Lattice Enumeration for Tower NFS: a 521-bit Discrete Logarithm Computation

Gabrielle De Micheli, Pierrick Gaudry, and Cécile Pierrot

Université de Lorraine, CNRS, Inria

Abstract. The Tower variant of the Number Field Sieve (TNFS) is known to be asymptotically the most efficient algorithm to solve the discrete logarithm problem in finite fields of medium characteristics, when the extension degree is composite. A major obstacle to an efficient implementation of TNFS is the collection of algebraic relations, as it happens in dimension greater than 2. This requires the construction of new sieving algorithms which remain efficient as the dimension grows. In this article, we overcome this difficulty by considering a lattice enumeration algorithm which we adapt to this specific context. We also consider a new sieving area, a high-dimensional sphere, whereas previous sieving algorithms for the classical NFS considered an orthotope. Our new sieving technique leads to a much smaller running time, despite the larger dimension of the search space, and even when considering a larger target, as demonstrated by a record computation we performed in a 521-bit finite field $\mathbb{F}_{p^6}$. The target finite field is of the same form than finite fields used in recent zero-knowledge proofs in some blockchains. This is the first reported implementation of TNFS.

1 Introduction

Context. While the post-quantum competition is ongoing, the discrete logarithm problem is still at the basis of the security of many currently-deployed public key protocols. Given a cyclic group G, a generator $g \in G$ and a target $h \in G$, solving the discrete logarithm problem in G means finding an integer $x \mod |G|$ such that $g^x = h$. The hardness of this problem depends on the group G and the two usual choices are the group of the invertible elements in a finite field and the group of points of an elliptic curve. This article deals with discrete logarithms in finite fields. In particular, as small characteristics finite fields are no longer considered because of the advent of quasipolynomial time algorithms [3, 15, 29], we focus on medium and large characteristics. For a finite field $\mathbb{F}_{p^n}$ we recall that the characteristic p is of medium size if $L_{p^n}(1/3) < p < L_{p^n}(2/3)$ and of large size if $p > L_{p^n}(2/3)$. [1] Equivalently, it means that the extension degree n is of bounded size with respect to the finite field order.

[1] We use the usual notation $L_Q(\alpha, c) = \exp((c + o(1))(\log Q)^\alpha (\log\log Q)^{1-\alpha})$, where $o(1)$ tends to 0 when Q tends to infinity.

NFS and TNFS. The Number Field Sieve (NFS) algorithm and its variants are the fastest known algorithms to solve the discrete logarithm problem in finite fields of medium and large characteristics. One of these variants is the Tower Number Field Sieve (TNFS), known to be asymptotically more efficient than a classical NFS for some fields when the extension degree is composite. TNFS exploits the algebraic structure of towers of number fields: the main difference with NFS comes from the representation of the target field $\mathbb{F}_{p^n}$. Whereas in the classical NFS setup, the finite field $\mathbb{F}_{p^n}$ is represented as the quotient field $\mathbb{F}_p[x]/(f)$ where f is a polynomial of degree n over $\mathbb{F}_p$, in the TNFS setup, we have $\mathbb{F}_{p^n} \cong \mathcal{R}/p\mathcal{R}$ where $\mathcal{R}$ is the ring defined as the quotient $\mathbb{Z}[\iota]/h(\iota)$, and $h \in \mathbb{Z}[\iota]$ is a degree n polynomial that remains irreducible modulo p.

Originally proposed by Schirokauer [36], TNFS was reinvestigated by Barbulescu, Gaudry, and Kleinjung [4] in 2015. They showed that the asymptotic complexity of TNFS in large characteristics is $L_{p^n}(1/3, \sqrt[3]{64/9})$, the same as for NFS. In medium characteristics, the complexity of TNFS is greater than $L_{p^n}(1/3)$ and thus this algorithm is only considered in the large case.

This algorithm was then modified by Kim, Barbulescu [27] and Jeong [28] to form the extended Tower Number Field Sieve (exTNFS), the variant being dedicated to composite extension degrees, *i.e.,* when $n = \eta\kappa$. This extended variant has an $L_{p^n}(1/3)$ complexity also in medium characteristics. In this case, the overall complexity of exTNFS can be as low as $L_{p^n}(1/3, \sqrt[3]{48/9})$ if there is a factor of n of the appropriate size (see Table 1). Both TNFS and exTNFS can be coupled with a multiple field variant – for any finite field – and a special variant – for some sparse characteristics only – giving each time a lower asymptotic complexity. We do not address these variants in this article.

Algorithm	Medium characteristic	Boundary	Large characteristic
NFS	96	48	64
TNFS	–	–	64
exTNFS	≥ 48	48	64

Table 1: Medium and large characteristics complexities of various algorithms, expressed as $L_{p^n}(1/3, \sqrt[3]{c/9})$, where c is the reported value in this table.

Towards an implementation of exTNFS. One can see from the complexities given in Table 1 that for NFS, medium characteristics are harder than large characteristics. This remains true for the multiple and special variants. However, a noticeable exception to this observation lies in the exTNFS algorithm. Indeed, when the degree n is composite, *i.e.,* $n = \eta\kappa$, the target finite field $\mathbb{F}_{p^n} = \mathbb{F}_{p^{\eta\kappa}}$ can be viewed as $\mathbb{F}_{P^\kappa}$ where P is a prime power of the same bitsize as p^η. Thus, the complexity of exTNFS in medium characteristics can be viewed similarly as the complexity of NFS at the boundary case between medium and large characteristics, leading to a smaller c constant in the L_{p^n}-notation. Hence we find

a lower complexity in medium characteristics than in large ones with exTNFS, which makes it a promising candidate for computational records in this area.

Let us assume we want to evaluate the security of a family of finite fields with fixed composite extension degree (for instance $n = 6$). These families often arise in pairing-based protocols. Evaluating the security of a concrete finite field in such a family is not an easy task, as we are not even able to tell beforehand whether NFS or exTNFS would be the fastest algorithm. Indeed, using a fixed extension degree asymptotically defines the characteristic as large, an area where the best discrete logarithm algorithm is NFS (not exTNFS). However, let us keep in mind that medium and large characteristics are notions defined for asymptotic sizes; as soon as we set a concrete target finite field it is not well understood how we should qualify its characteristic. In this work we underline that exTNFS shows real improvements with regards to current NFS computations for this family. Current record computations (e.g. for 400 or 500-bit finite fields) deal with areas where asymptotic analysis are not yet the relevant ones. We cannot easily extrapolate on current and deployed sizes (e.g. for more than 2000-bit finite fields) but our implementation of exTNFS provides practical insight on security parameters by showing its incredibly good behavior at lower sizes.

In the rest of this article, to simplify notations and to be coherent with the recent literature, we use TNFS as a short hand for ex-TNFS[2]. We do however assume the degree n of our target finite field is composite, thus considering specifically the extended variant.

Lattice enumeration for TNFS. Despite the fact that TNFS is promising, no implementation was done using this variant of NFS, up to this work. Indeed, so far, excluding the very small characteristics 2 and 3, all discrete logarithm record computations were performed using NFS, the special variant of NFS, or the Function Field Sieve – a method for small characteristics only.

A major obstacle to an efficient implementation of TNFS is the collection of algebraic relations where equations between small elements of number fields must be found. Indeed, whereas NFS requires sieving through $(a, b) \in \mathbb{Z}^2$ pairs, the tower setup sieves through $(a(\iota), b(\iota))$-pairs, *i.e.,* degree $\eta - 1$ polynomials with bounded coefficients. This requires the construction of sieving algorithms in a space of dimension $2\eta \geq 4$, which remain efficient as the dimension grows.

In dimension 2, Franke and Kleinjung [13] proposed in 2005 an efficient algorithm used in all previous records. For higher dimensions, after the pioneer work in $\mathbb{F}_{p^{12}}$ by Hayasaka et al. [22], the transition vectors method from Grémy [16] and a recursive plane method proposed by McGuire and Robinson [31] were tested in dimension 3 and used for record computations using NFS. However, the efficiency of their algorithms for even higher dimensions is questionable.

Our work. In this article, we introduce an efficient sieving algorithm for higher dimensions which allows us to implement TNFS and perform the first record computation with it. More specifically, we propose the following contributions.

[2] We use the same abuse in the abstract and title too.

1. *Sieving in a high dimensional sphere instead of an orthotope.* All sieving algorithms so far considered a product of intervals as search space $\mathcal{S}$. Indeed, whether a candidate relation is characterized by an (a, b)-pair or an $(a(\iota), b(\iota))$-pair with more than two coefficients, every coefficient is bounded separately in an interval $[-H_1^i, H_2^i]$ for $i = 1, 2, \cdots, d$ where d is the total number of coefficients. Hence, the search space considered is a d-orthotope of the form $\mathcal{S} = [-H_1^1, H_2^1] \times \cdots \times [-H_1^d, H_2^d]$. We argue that when $d \geq 3$, the shape of $\mathcal{S}$ must be adequately chosen. More precisely, we consider a d-sphere instead of a d-orthotope and explain why we believe this choice leads to a more efficient algorithm when the dimension grows.
2. *Adapting a lattice enumeration algorithm to the context of TNFS.* In order to fully exploit the new search space, we adapt a known lattice algorithm to the context of TNFS: Schnorr-Euchner's enumeration algorithm [38], that outputs the shortest vector of a lattice. We modify this algorithm in order to list all the vectors of a lattice $\mathcal{L}$ within a d-dimensional sphere S_d. Furthermore, a part of the common coefficients of the enumerated vectors are kept in memory during the algorithm, leading to a 10% reduction in the execution time. This algorithm remains competitive when the dimension grows and also provides an exhaustive search of all the vectors in $\mathcal{L} \cap S_d$, contrary to previous approaches.
3. *Analysis of the relation collection step in TNFS and duplicate relations.* We place this sieving algorithm in the context of the entire relation collection step. Sieving algorithms are usually combined with batch algorithms and ECM to provide the most efficient relation collection. We give details on this relation collection step, and give new insight on how to define and remove duplicate relations that arise in the context of TNFS.
4. *A 521-bit finite field record.* Our new lattice enumeration for the sieving step led to the first record computation of a discrete logarithm with TNFS, reaching a 521-bit finite field $\mathbb{F}_{p^6}$. Previous record on a finite field of the same shape reached a 423-bit finite field in January 2020. The choice of the extension degree was motivated by the use of such finite fields in pairing-based protocols, in particular in recent zero-knowledge proofs in some blockchains [5, 9]. Ultimately, as shown in Table 2, our algorithm is much faster than existing high-dimensional sieving algorithms, despite the larger dimension of the search space and the larger finite field.

Parameters	[17]	[31]	This work
Algorithm	NFS	NFS	TNFS
Field size (bits)	422	423	521
Sieving dimension	3	3	6
Sieving time	201,600	69,120	**23,300**

Table 2: Comparison of the relation collection step in core hours with [17] and [31] for finite fields of the form $\mathbb{F}_{p^6}$.

Outline. In Section 2, we recall the general setup of TNFS and in particular we focus on the steps that differ the most from the classical NFS setup. In Section 3, we focus on the relation collection step with the special-q method, and explain how to deal with duplicate relations. In Section 4, we describe our adaptation of Schnorr-Euchner's enumeration algorithm to the context of TNFS. We justify why we choose a d-sphere as sieving area and introduce an efficient way to compute the desired vectors of coefficients for the relations. Section 5 analyses the complexity of our sieving algorithm and compares the latter with pre-existing algorithms. In Section 6 we detail our complete discrete logarithm computation in a 521-bit finite field with extension degree 6. Finally, in Section 7 we study the potential savings brought by explicit automorphisms, and introduce the notion of Galois-compatible Schirokauer maps, relevant in this context.

2 The Tower Number Field Sieve

2.1 Mathematical setup

The classical tower of number fields that illustrates the TNFS setup considers the intermediate number field $\mathbb{Q}(\iota)$ where ι is a root of h, a polynomial over $\mathbb{Z}$ that remains irreducible modulo p. Above this number field are set the two number fields $K_1 = \mathbb{Q}(\iota)[x]/f_1(x)$ and $K_2 = \mathbb{Q}(\iota)[x]/f_2(x)$ where f_1, f_2 are irreducible polynomials over $\mathcal{R} = \mathbb{Z}[\iota]$ that share an irreducible factor φ modulo the unique ideal $\mathfrak{p}$ over p in $\mathbb{Q}(\iota)$. We write $\mathcal{O}_i$ the ring of integers of K_i and α_i a root of f_i in K_i for $i = 1, 2$. This construction is illustrated in Figure 1. Because of the

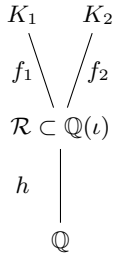

Fig. 1: Tower of Number Fields used in TNFS.

conditions on the polynomials h, f_1 and f_2, there exist two ring homomorphisms from $\mathcal{R}[x] = \mathbb{Z}[\iota][x]$ to the target finite field $\mathbb{F}_{p^n}$ through the number fields K_1 and K_2. This allows to build a commutative diagram as shown in Figure 2. The extension degree n is assumed to be composite, and we write $n = \eta\kappa$. In this setting, h is of degree η, and f_1 and f_2 have degree at least κ, so that the degree of their common factor φ is exactly κ. For simplicity, we will assume that f_1 and f_2 are defined over $\mathbb{Z}$, since it is the case in our record computation; this is only possible when κ and η are coprime.

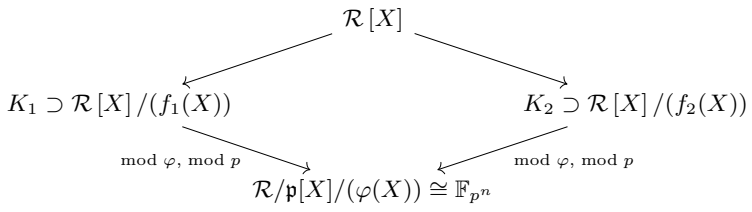

Fig. 2: Commutative diagram of Tower NFS.

2.2 A step by step walk through TNFS

The TNFS algorithm follows similar steps as any index calculus algorithm. The main differences with the Number Field Sieve algorithm take place in the polynomial selection step and the collection of relations.

Polynomial selection. Unlike NFS which uses only two polynomials f_1 and f_2 to define the number fields, three polynomials must be selected for this algorithm, namely h, f_1 and f_2. The polynomial h must be of degree η and irreducible modulo p to ensure the uniqueness of the ideal $\mathfrak{p}$ over p in $\mathcal{R}$. Ideally one would choose a unitary h with small coefficients and such that the inverse of the Dedekind zeta function evaluated at 2 (implemented in Sage for example) is close to 1. Indeed, as we will see in Section 3.4, this is related to non-coprime ideals that produce equivalent relations which are useless for the linear algebra step. Moreover, the proportion of coprime ideals is given by

$$\prod_{\mathfrak{q} \text{ prime ideal in } \mathbb{Q}(\iota)} \left(1 - \frac{1}{N(\mathfrak{q})^2}\right) = \frac{1}{\zeta_{\mathbb{Q}(\iota)}(2)},$$

where $\zeta_{\mathbb{Q}(\iota)}$ is the Dedekind zeta function. Hence, in order to reduce the amount of duplicate relations produced because of these non-coprime ideals, one can choose an h with $\zeta_{\mathbb{Q}(\iota)}(2)$ as close to 1 as possible.

The polynomials f_1 and f_2 are selected to fit the mathematical setting of Section 2.1. One can use NFS polynomial selections such as the Conjugation, JLSV or Sarkar-Singh's methods [2,24,35], not recalled here. The polynomials we use for our 521-bit computation come from the Conjugation method. The choice of the polynomial selection can be made as follows. We know that JLSV produces polynomials f_1 and f_2 of same degree, say d, and with coefficients of size $O(\sqrt{p})$ for both polynomials. On the other hand, the Conjugation method has more unbalanced parameters. The polynomial f_1 has a larger degree $2d$ and small coefficient in $O(1)$ and f_2 has degree d (the same as for JLSV) and coefficients of size $O(\sqrt{p})$. Thus in order to decide which method to use, one should compare the size of the norms $N_1(\phi)$ and $N_2(\phi)$ for elements $\phi \in \mathcal{R}[x]$. If $N_1(\phi) < N_2(\phi)$, then one should choose the Conjugation method. However, if $N_2(\phi) < N_1(\phi)$,

then one should choose the JLSV method since the polynomial f_2, which shares the same properties as polynomials outputted from JLSV, has the smaller norm.

In NFS, the quality of the polynomials can be refined with a quantity known as the Murphy-α value. See [20] for details about Murphy-α adapted to TNFS.

Relation collection. The goal of the relation collection step is to select among the set of linear polynomials $\phi(x, \iota) = a(\iota) - b(\iota)x \in \mathcal{R}[x]$ at the top of the diagram the candidates which produce a relation. A relation is found if the polynomial $\phi(x, \iota)$ mapped to K_1 and K_2 factors into products of ideals of small norms in both number fields. The ideals of small norms that occur in these factorizations constitute the factor basis $\mathcal{F}$. More precisely, we define it as $\mathcal{F} = \mathcal{F}_1 \cup \mathcal{F}_2$ with

$$\mathcal{F}_i(B) = \{\text{prime ideals of } \mathcal{O}_i \text{ of norm} \leq B, \text{ whose inertia degree over } \mathbb{Q}(\iota) \text{ is } 1\},$$

for $i = 1, 2$.

Remark 1. The prime ideals in the factorization of $(a(\iota) - b(\iota)\alpha_i)O_i$ can have degree at most $\deg \phi$. Thus the ideals in the factor basis have degree at most 1 in the variable x. However, in the specific context of TNFS, the polynomial ϕ is a polynomial in two variables: x and ι. Thus, these ideals can have degree in ι up to degree of h. We will however only consider ideals of degree 1 in ι. Indeed, in theory, we only lose a constant factor in the smoothness probabilities when excluding the ideals of degree greater than 1. In practice, if we encounter a prime ideal that has a degree greater than 1 in the variable ι, we keep the relation. Otherwise, we would throw away too many relations. The same holds for ideals dividing the index ideal $[\mathcal{O}_i : \mathcal{O}_i[\alpha_i]]$ but there are only finitely many and they can be handled separately.

The representation of these ideals of degree 1 in the context of TNFS is summarized in [4, Proposition 1], which we recall below.

Proposition 1 (From [4], Proposition 1). *Let $\mathbb{Q}(\iota)$ be a number field and let $\mathcal{O}_\iota$ be its ring of integers. Let f be a monic irreducible polynomial in $\mathbb{Q}(\iota)[x]$, and denote by α one of its roots. We denote by $K = \mathbb{Q}(\alpha, \iota)$ the corresponding extension field, and $\mathcal{O}_f$ its ring of integers.*

If $\mathfrak{q}$ is a prime ideal of $\mathcal{O}_\iota$ not dividing the index-ideal $[\mathcal{O}_f : \mathcal{O}_\iota[\alpha]]$, then the following statement holds.

1. *The prime ideals of $\mathcal{O}_f$ above $\mathfrak{q}$ are all the ideals of the form*

$$\mathfrak{Q} = (\mathfrak{q}, T(\alpha)),$$

   where $T(x)$ are the lifts to $\mathcal{O}_\iota[x]$ of the irreducible factors of f in $\mathcal{O}_\iota/\mathfrak{q}[x]$. Moreover, $\deg \mathfrak{Q} = \deg T$.
2. *If $a(t), b(t) \in \mathbb{Z}[t]$ are such that $\mathfrak{q}$ divides $N_{K/\mathbb{Q}(\iota)}(a(\iota) - b(\iota)\alpha)$ and $a(\iota)\mathcal{O}_\iota + b(\iota)\mathcal{O}_\iota = \mathcal{O}_\iota$, then the unique ideal of $\mathcal{O}_f$ above $\mathfrak{q}$ which divides $a(\iota) - b(\iota)\alpha$ is $\mathfrak{Q} = (\mathfrak{q}, \alpha - r(\iota))$ with $r \equiv a(\iota)/b(\iota) \pmod{\mathfrak{q}}$.*

To verify the B-smoothness on each side, one needs to evaluate the norms $N_i(a(\iota) - b(\iota)\alpha_i)$ for $i = 1, 2$. To do so we recall that when the polynomials f_i are monic, these norms are integers that can be computed thanks to resultants as

$$N_i(a(\iota) - b(\iota)\alpha_i) = \mathrm{Res}_t(\mathrm{Res}_x(a(t) - b(t)x, f_i(x)), h(t)).$$

This allows to verify the B-smoothness over integer values. These formulas must be adapted when the polynomials f_i are not monic by including the leading coefficient raised to the degree of the polynomial. The relation collection step stops when we have enough relations to construct a system of linear equations that may be full rank. The unknowns of these equations are the *virtual* logarithms of the ideals of the factor basis.

Linear algebra. A good feature of the linear system created is that the number of non-zero coefficients per line is very low. This allows to use sparse linear algebra algorithms such as the block variant of Wiedemann's algorithm [39], for which parallelization is partly possible. The output of this step is a kernel vector corresponding to the virtual logarithms of the ideals in the factor basis. There is no difference for this step between TNFS and NFS.

Individual discrete logarithm. The final step of TNFS consists in finding the discrete logarithm of the target element. This step is subdivided into two substeps: a smoothing step and a descent step. The smoothing step is an iterative process where the target element t is randomized by considering $s = g^e t \in \mathbb{F}_{p^n}^*$ for an exponent e chosen uniformly at random. Values for e are tested until s lifted back to one of the number fields K_i is B_i-smooth for a smoothness bound $B_i > B$.

The second step consists in decomposing every factor of the lifted value of s, in our case prime ideals with norms less than a smoothness bound B_i (but usually greater than B) into elements of the factor basis for which we now know the virtual logarithms. This process creates descent trees where the root is an ideal coming from the smoothing step and the nodes are ideals that get smaller and smaller as they go deeper. The leaves are ultimately elements of the factor basis. The edges of the tree are defined as follows: for every node, there exists an equation between the ideal of the node and all the ideals of its children.

In this work, we consider an improvement given by Guillevic in [19, Algorithm 5] for the smoothing step, that is useful in the context of TNFS only. The goal is to improve the smoothness probability of the lift of $s \in \mathbb{F}_{p^n}^*$ to K_i by constructing an adequate lattice whose reduced vectors define elements of K_i with potentially small norms which is precisely the potential lifts of s we are looking for.

3 Focus on the relation collection

In TNFS the relation collection step requires sieving in dimension $2\eta \geq 4$, which is the number of coefficients involved in ϕ. We start by dividing the set of polynomials ϕ into multiple subsets and then we present different algorithms to successively select the candidates in each of these subsets.

3.1 The special-q setup

The relation collection phase looks at a set of linear polynomials $\phi(x, \iota) = a(\iota) - b(\iota)x \in \mathcal{R}[x]$ where a, b are polynomials of degree $\deg h - 1$ with $\deg h = \eta$ and bounded coefficients, and tries to identify which are going to produce doubly-smooth norms, *i.e.,* for which pair $(a(\iota), b(\iota))$ the norms $N_1\left(a(\iota) - b(\iota)\alpha_1\right)$ and $N_2\left(a(\iota) - b(\iota)\alpha_2\right)$ factor into small primes. To reduce the time of the sieving stage, Pollard [34] suggested to divide the set of all polynomials ϕ, commonly called the search space, into multiple subsets. This task also requires a large amount of memory and Pollard's idea allows to parallelize the process, thus improving practical computation time. This corresponds to the so-called *special-q method*. This method regroups polynomials into groups such that $\phi(\alpha_1, \iota)$ (or $\phi(\alpha_2, \iota)$ depending on whether we put the special-q on the f_1-side or the f_2-side) share a common factor: the ideal $\mathfrak{Q}$, above a prime q, hence the name. Thus, when talking about a sieving algorithm, we usually consider a fixed special-q ideal $\mathfrak{Q}$, and select good polynomials ϕ in the corresponding subset. This idea of using special-q's increases the smoothness probability on the side where divisibility by $\mathfrak{Q}$ is forced, since the norm is already divisible by q. Furthermore this provides a natural parallelization where each work-unit corresponds to a special-q.

Let $\underline{\phi}$ denote the (row-) vector of coefficients of the polynomial $\phi(x, \iota)$, *i.e.* the vector $\underline{\phi} = (a_0, \cdots, a_{\eta-1}, b_0, \cdots, b_{\eta-1}) \in \mathbb{Z}^{2\eta}$. Let us consider a special-q ideal $\mathfrak{Q}$ of degree 1 in K_i of the form $\mathfrak{Q} = \langle q, \iota - \rho_\iota, x - \rho_x \rangle$, where q is a prime number, ρ_ι is a root of h modulo q, and ρ_x is a root of f_i modulo q. One could also consider ideals of degree greater than 1, but special-q of degree 1 are the most common among ideals of bounded norms and thus we restrict to this case.

Proposition 2. *The set of polynomials ϕ such that the corresponding principal ideal in K_i is divisible by $\mathfrak{Q}$ form a lattice that we call the $\mathfrak{Q}$-lattice $\mathcal{L}_\mathfrak{Q}$.*

The latter can be made explicit as follows.

$$\mathcal{L}_\mathfrak{Q} = \{(a_0, \cdots, a_{\eta-1}, b_0, \cdots, b_{\eta-1}) \in \mathbb{Z}^{2\eta} : \sum_{k=0}^{\eta-1} \left(a_k \iota^k - b_k \iota^k \alpha_i\right) \equiv 0 \pmod{\mathfrak{Q}}\}$$

where $i = 1, 2$ depending on the side we consider. A basis $B_\mathfrak{Q}$ of this lattice can be expressed as seen in Figure 3.

The determinant of this lattice is $q^{\deg \phi_h}$, where ϕ_h is an irreducible factor of $h \pmod p$. In our case $\phi_h = \iota - \rho_\iota$ because we only consider special-q ideals of degree 1 and so the determinant is simply q. The lattice dimension is 2η.

For example, with $\eta = 3$, it is generated by the rows of the matrix given in Figure 4. where the first 3 columns correspond to the coefficients of $a(\iota) = a_0 + a_1\iota + a_2\iota^2$, and the last 3 columns to the coefficients of $b(\iota) = b_0 + b_1\iota + b_2\iota^2$. For example, the 5^{th} row corresponds to the polynomial $\phi_5(x, \iota) = -\rho_x\iota + \iota x$, and $\underline{\phi}_5 = (0, \rho_x, 0, 0, 1, 0)$ is indeed in $\mathcal{L}_\mathfrak{Q}$. The lattice $\mathcal{L}_\mathfrak{Q}$ has dimension $d = 2\eta = 6$ and determinant q in this example.

$$
\begin{array}{c|c|c}
(a,b) & a_0 \quad a_1 \quad \cdots \; a_{\eta-2} \quad a_{\eta-1} & b_0 \; b_1 \; \cdots \; b_{\eta-1} \\
\hline
(q,0) & \begin{pmatrix} q & 0 & & & 0 \\ -\rho_\iota & 1 & 0 & & \\ 0 & -\rho_\iota & 1 & 0 & \\ & & \ddots & \ddots & \\ & & & -\rho_\iota & 1 \\ \rho_x & 0 & & & \\ 0 & \rho_x & 0 & & \\ & & \ddots & & \\ & & & \rho_x & \end{pmatrix} & \begin{pmatrix} & & & \\ & & & \\ & & & \\ & & & \\ & & & \\ 1 & & & \\ & 1 & & \\ & & \ddots & \\ & & & 1 \end{pmatrix} \\
(\iota-\rho_\iota,0) & & \\
(\iota(\iota-\rho_\iota),0) & & \\
\vdots & & \\
(\iota^{\eta-2}(\iota-\rho_\iota),0) & & \\
(\rho_x,1) & & \\
(\iota\rho_x,\iota) & & \\
\vdots & & \\
(\iota^{\eta-1}\rho_x,\iota^{\eta-1}) & &
\end{array}
= B_{\mathfrak{Q}}.
$$

Fig. 3: A basis $B_{\mathfrak{Q}}$ of the lattice $\mathcal{L}_{\mathfrak{Q}}$.

$$
B_{\mathfrak{Q}} = \begin{pmatrix} q & 0 & 0 & 0\,0\,0 \\ -\rho_\iota & 1 & 0 & 0\,0\,0 \\ 0 & -\rho_\iota & 1 & 0\,0\,0 \\ -\rho_x & 0 & 0 & 1\,0\,0 \\ 0 & -\rho_x & 0 & 0\,1\,0 \\ 0 & 0 & -\rho_x & 0\,0\,1 \end{pmatrix},
$$

Fig. 4: Example with $\eta = 3$.

Each unit of computation targets one special-q ideal $\mathfrak{Q}$ and searches for polynomials $\phi(x,\iota)$ with $\underline{\phi} \in \mathcal{L}_{\mathfrak{Q}}$ leading to relations, *i.e.,* for which both sides are smooth. In order to explore the lattice $\mathcal{L}_{\mathfrak{Q}}$, we first LLL-reduce the basis $B_{\mathfrak{Q}}$, and then consider linear combinations with small coefficients of these new basis elements. This allows us to focus on polynomials where one of the norms on one side is known to be divisible by q, thus increasing the probability of it being smooth. More precisely, let $M_{\mathfrak{Q}}$ be an LLL-reduced basis of $\mathcal{L}_{\mathfrak{Q}}$. We study the (row-) vectors $\mathbf{c}$ of coefficients such that $\underline{\phi} = \mathbf{c} \cdot M_{\mathfrak{Q}}$, potentially leads to a relation. This is done using sieving algorithms.

3.2 Constructing the double-divisibility lattice $\mathcal{L}_{\mathfrak{Q},\mathfrak{p}}$

We concentrate on vectors $\mathbf{c}$ that belong to a sieving region $\mathcal{S}$. Traditionally, $\mathcal{S}$ is an ℓ_∞-ball, however in this work we consider the ℓ_2-norm. Section 4.2 explains this preference. In order to efficiently detect the vectors $\mathbf{c}$ giving elements of smooth norms, one can perform an Eratosthenes-like sieving, quickly marking all vectors $\mathbf{c}$ in $\mathcal{S}$ leading to a norm on the f_1-side (or the f_2-side) that is divisible by a small prime p. Repeating this sieve for many primes p allows to detect the most promising vectors $\underline{\phi}$, those for which the norm is divisible by many small primes. To do so, we proceed as for the divisibility by $\mathfrak{Q}$.

Let $\mathfrak{p}$ be a prime ideal of norm p in K_i of the form $\mathfrak{p} = \langle p, \iota - r_\iota, x - r_x \rangle$, where r_ι is a root of h modulo p and r_x is a root of f_i modulo p. The second statement of [4, Proposition 1] can be reformulated for this specific context.

Proposition 3. *The principal ideal generated by $\phi(x, \iota)$ in K_i is divisible by $\mathfrak{p}$ if and only if $\phi(r_x, r_\iota) \equiv 0 \mod p$.*

Let $\mathbf{U}_\mathfrak{p}$ be the (row-) vector of size 2η defined by

$$\mathbf{U}_\mathfrak{p} = \left(1, r_\iota \bmod p, \cdots, r_\iota^{\eta-1} \bmod p, r_x, r_x r_\iota \bmod p, \cdots, r_x r_\iota^{\eta-1} \bmod p\right).$$

Then similarly as before, we can translate the divisibility property of the ideal of Proposition 3: the divisibility by $\mathfrak{p}$ is equivalent to the condition $\underline{\phi} \cdot \mathbf{U}_\mathfrak{p}^\intercal \equiv 0 \mod p$. Recall that ϕ is taken in a subset of the search space so that the ideal generated by ϕ is divisible by $\mathfrak{Q}$, namely its coefficients are written as $\underline{\phi} = \mathbf{c} \cdot M_\mathfrak{Q}$. Taking into account the divisibility by $\mathfrak{Q}$ and by $\mathfrak{p}$ yields the condition on $\mathbf{c}$:

$$\mathbf{c} \cdot M_\mathfrak{Q} \mathbf{U}_\mathfrak{p}^\intercal \equiv 0 \mod p. \tag{1}$$

The product $M_\mathfrak{Q} \mathbf{U}_\mathfrak{p}^\intercal$, reduced modulo p and normalized so that its first coordinate is 1, is expressed as $M_\mathfrak{Q} \mathbf{U}_\mathfrak{p}^\intercal \equiv \lambda \, (1, \alpha_1, \alpha_2, \ldots, \alpha_{\eta-1})^\intercal \mod p$, with $\lambda > 0$. Since $M_\mathfrak{Q}$ and $\mathbf{U}_\mathfrak{p}$ are known, we explicitly compute the values α_i. This assumes the first coordinate is non-zero. Otherwise, one must either adapt the construction of $M_{\mathfrak{Q},\mathfrak{p}}$ below or skip the ideal $\mathfrak{p}$ during sieving. Finally, the set of vectors $\mathbf{c}$ verifying Equation (1) is the lattice $\mathcal{L}_{\mathfrak{Q},\mathfrak{p}}$ generated by the rows of the matrix

$$M_{\mathfrak{Q},\mathfrak{p}} = \begin{pmatrix} p & 0 & 0 & 0 \cdots 0 \\ -\alpha_1 & 1 & 0 & 0 \cdots 0 \\ -\alpha_2 & 0 & \ddots & 0 \cdots 0 \\ \vdots & 0 & 0 & \ddots 0 \\ -\alpha_{\eta-1} & 0 & 0 & 0 \cdots 1 \end{pmatrix}.$$

In the end, since $M_{\mathfrak{Q},\mathfrak{p}}$ is explicitly known, we can compute the coefficients $\underline{\phi} = \mathbf{c} \cdot M_\mathfrak{Q}$ of the polynomials ϕ. This is possible as soon as we are able to enumerate the short vectors $\mathbf{c}$ in this lattice which is the aim of Section 4. This procedure, which is called enumeration, is done for all prime p from 2 up to a predefined bound $p_{\max}$, forming a so called sieving algorithm. Sieving allows to detect quickly vectors $\mathbf{c}$ that belong to several $M_{\mathfrak{Q},\mathfrak{p}}$ for various $\mathfrak{p}$. The corresponding polynomials ϕ are good candidates that potentially give relations as they provide by construction ideals that are already divisible by $\mathfrak{Q}$ and several ideals $\mathfrak{p}$. Hence we keep them for the next selection phase.

3.3 Combining three algorithms

Other algorithms are used, either to directly detect polynomials leading to doubly-smooth norms, or to work as complementary algorithms one after another as a sequence of filters to determine and only keep the promising candidates at

each step (as done with the enumeration above). These algorithms either find and extract smooth parts of the norms, or completely factor them. The family of sieving algorithms [16,31], batch algorithms [6, Algorithm 2.1] and ECM [8,30] are examples of such methods used in factorization and DLP computations.

Batch. Batch algorithms are quasi-linear time algorithms that test the smoothness of a list of integers. Multiple variants exist depending on the required output. The first two variants take as input the list of integers and a list of primes up to some bound: batch trial division outputs the list of primes p that divide the integers and batch smooth part extraction only returns the smooth part of the integers, *i.e.,* the product of the primes that divide the integers. Finally batch coprime factorization only takes as input the list of integers and returns the factors of each integer using repeated gcd operations.

In the context of DLP computations, the batch smooth part extraction algorithm is favored since it has the lowest complexity and is enough to test for potential smooth candidates. More precisely, the algorithm takes as input a list of prime numbers $P = \{p_1, \cdots, p_{\text{batch}}\}$ and a list of integers $N = \{n_1, \cdots, n_m\}$ and outputs for each $n_i \in N$ its smooth part, *i.e.,* the product of its prime factors in P. The algorithm uses product trees and remainder trees to efficiently compute the $\gcd(n_i, \prod_j p_j)$ for all i. We refer to [6, Algorithm 2.1] for the complete algorithm.

The batch smooth part extraction has complexity $O(M(B)\log B)$ where $M(n)$ is the cost of multiplication of integers of size n bits and B is the number of bits of $\max(\prod p_j, \prod n_i)$.

Elliptic Curve Factorization Method. The elliptic curve factorization method [30], commonly known as ECM is an elliptic curve-based integer factorization algorithm that runs in sub-exponential time. The algorithm takes as input a composite integer n and outputs small factors of n. The algorithm relies on Hasse's theorem that states that if p is a prime, then the group order of an elliptic curve over $\mathbb{Z}/p\mathbb{Z}$ is $p + 1 - t$ where $|t| \leq 2\sqrt{p}$. The algorithm picks a random elliptic curve $E(\mathbb{Z}/n\mathbb{Z})$ and a point P on it and computes the scalar multiplication $Q = kP$ where $k = \prod p_i$ for primes p_i up to some bound. Let p be one of the factors of n which we want to find. If the order of the curve E over $\mathbb{Z}/p\mathbb{Z}$ divides k, then the z-coordinate of the point Q taken mod p is equal to zero and can thus be recovered by computing $\gcd(z, n) = p$. The running-time of ECM is $O(L_p(1/2, \sqrt{2}+o(1))M(\log n))$, where p is the smallest prime factor of n.

These algorithms all have different complexities and properties and thus cannot be used on the same amount of input norms N_i. ECM is for example much more costly than sieving. Hence, applying it to all norms N_i is far from optimal. On the other hand, sieving is a much less costly algorithm per candidate, and thus can be used to find the small factors (up to $p_{\max}$) of a large number of norms of structured candidates. This is why the relation collection step usually starts with a sieving algorithm with input all candidates $(a(\iota), b(\iota))$ pairs. ECM

is then used to guarantee that the norms of promising candidates are indeed B-smooth by checking the larger prime factors. Batch smoothness can be added in between sieving and ECM or as a substitution of one of them to further optimize the overall cost. It is less costly than ECM and thus can be used to pre-select promising candidates but more costly than sieving and thus cannot be run on the entire set of candidates. It extracts prime factors up to a bound p_{batch} such that $p_{\max} < p_{\text{batch}} < B$. Table 3 describes the properties of these algorithms.

Properties	Sieving	Batch	ECM
Input candidates	Numerous and structured	Numerous	Few
Prime factors extracted	Small	Small or medium	Large
RAM	Very large	Large	Tiny
Cost per candidate	Small	Medium	High

Table 3: Properties of the different relation collection algorithms

The relation collection is thus seen as a sequence of filters, each taking a certain amount of candidates as input, and selecting *survivors* based on a criterion. These survivors are then the input to the next filter. The selection of survivors is usually based on the size of the cofactor, which we now define.

Definition 1 ($\mathcal{A}$-cofactor). *Let N be a positive integer and consider $P = \prod_i p_i$ where the p_i are the prime factors of N extracted by Algorithm $\mathcal{A}$. Then the $\mathcal{A}$-cofactor of N is $C_{\mathcal{A}}(N) = N/P$.*

Remark 2. When $\mathcal{A}$ is sieving or ECM, the algorithm returns the primes p_i and when $\mathcal{A}$ is batch smoothness, the algorithm directly returns the product P.

For a fixed $\mathcal{A}$-cofactor threshold $\mathcal{T}_{\mathcal{A}}$, the survivors are the candidates selected if their norm N satisfies $C_{\mathcal{A}}(N) \leq \mathcal{T}_{\mathcal{A}}$ (there can be such a condition on both sides if sieving is done on both of them). This entire procedure is summarized in Figure 5.

Remark 3. Note that the bounds $\mathcal{T}_{\mathcal{A}}$ are not smoothness bounds. However, they allow to select promising candidates. Indeed, if the cofactor of a norm is small, the probability of the norm being smooth is higher.

Finally, the complete relation collection is given in Algorithm 1. Note that on line 7, we remove duplicate relations, as explained in the next Section.

3.4 Filtering through equivalent relations

When sieving through all the pairs of candidates it is sometimes the case that two pairs $(a(\iota), b(\iota))$ and $(a'(\iota), b'(\iota))$ provide the same relation, *i.e.,* they correspond to two linear equations that provide the same information on the virtual logarithms of the elements of the factor basis involved.

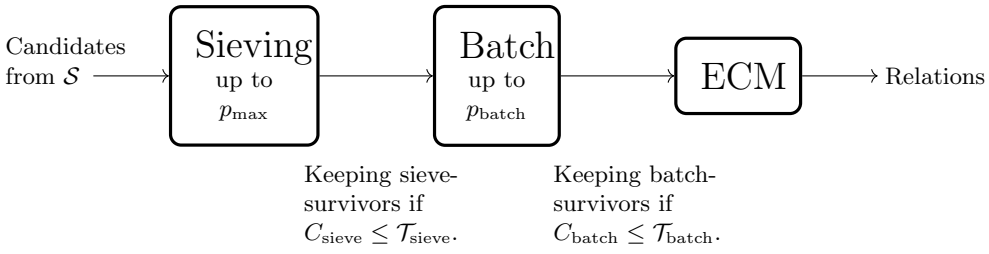

Fig. 5: Sequence of algorithms for relation collection.

Algorithm 1 Relation collection for a given special-q with sieving, batch and ECM

Input: A prime ideal $\mathfrak{Q}$, a sieving region $\mathcal{S}$
Output: A list of relations.

1: Construct the lattice $\mathcal{L}_{\mathfrak{Q}}$ and LLL-reduce it.
2: **for** each prime ideal $\mathfrak{p}$ in $\mathcal{O}_1$ (or $\mathcal{O}_2$) up to norm $p_{\max}$ **do**
3: Construct the lattice $\mathcal{L}_{\mathfrak{Q},\mathfrak{p}}$
4: Enumerate all vectors $\mathbf{c}$ in $\mathcal{L}_{\mathfrak{Q},\mathfrak{p}} \cap \mathcal{S}$.
5: For each $\mathbf{c}$, keep track of the size of the factor p with a sieving table.
6: For promising $\mathbf{c}$, for which the product of the factors p is large, compute approximations of the norms N_1, N_2 and identify vectors with sieve-cofactor smaller than $\mathcal{T}_{\text{siev}}$. They are called sieve-survivors.
7: Remove duplicates.
8: Run batch algorithm with input the (exact) norms N_1 and N_2 of the sieve-survivors and primes up to p_{batch}. Keep batch-survivors whose batch-cofactor is smaller than $\mathcal{T}_{\text{batch}} < \mathcal{T}_{\text{siev}}$.
9: Run ECM on the batch-survivors to identify all the doubly-B-smooth norms.
10: **return** Vectors with doubly-B-smooth norms

Removing duplicates is common in factoring and DLP computations. Let us start by identifying three different types of duplicates. Because these definitions apply in both NFS and TNFS, we use the terminology (a, b) to either define a classical $(a, b) \in \mathbb{Z}^2$ pair in NFS or $(a(\iota), b(\iota)) \in \mathcal{R}[x]$ in TNFS.

Definition 2 (Duplicates). *A duplicate relation refers to a pair (a, b) such that there exists another pair (a', b') that leads to the same relation. We distinguish three types of duplicates:*

- *We refer to **special-q-duplicates** when a relation with ideal factorization $(a - b\alpha_i)\mathcal{O}_i = \prod_j \mathfrak{Q}_j^{e_j}$ involves several prime ideals $\mathfrak{Q}_j$ that occur in the set of special-q's considered. In other words, more than one special-q units of computation provide the same relation.*
- *If (a, b) generates a relation for a fixed special-q, then a K_h**-unit-duplicate** refers to the pair (ua, ub), for $u \in \mathcal{O}_{K_h}^*$, where u is a small enough unit of K_h.*

– *If (a, b) generates a relation for a fixed special-q, then a ζ_2-**duplicate** refers to the pair $(\lambda a, \lambda b)$, for $\lambda \in \mathcal{O}_{K_h} \setminus \mathcal{O}_{K_h}^*$, where λ is small enough.*

Remark 4. The special-q-duplicates are considered over the entire set of relations, meaning for all special-q considered, whereas the other two types of duplicates concern a fixed special-q. The methods to remove them thus differ.

Duplicate relations generate identical or nearly identical lines in the linear system of equations. As the cost of solving the system grows with its dimension, we want to get rid of all the unnecessary lines. The related matrix is encoded as a list of prime ideal factors for each relation. So in theory, a simple solution would be to remove identical lines in this file before the linear algebra step.

However, in practice generating duplicate relations is costly. Indeed, they generate more hits during the enumeration of the vectors in $\mathcal{L}_{\mathfrak{Q},\mathfrak{p}} \cap \mathcal{S}$ (line 4 in Algorithm 1), they imply more sieve-survivors for which we must compute exact norms (line 8), and this finally results in more batch-survivors and hence a more costly ECM algorithm (line 9). To minimize these extra computations, it is thus convenient to get rid of the duplicates that can be identified as fast as possible. However, the same strategy cannot be applied for each type of duplicates.

Indeed, the special-q duplicates can only be detected once we know the entire factorization of the norms, meaning after running ECM. Moreover, special-q computation units are often run in parallel and thus there is little hope to be able to detect any special-q duplicates before the end of the relation collection phase. These duplicates are thus removed just before the linear algebra step.

On the other hand, K_h-unit-duplicates and ζ_2-duplicates are *local* to a special-q and can be detected at an earlier stage. Yet there is a trade-off between the extra cost of having duplicates and the cost of analyzing whether a pair (a, b) yields a duplicate relation. In our Algorithm 1, we chose to remove duplicates before running the batch algorithm. Let us now explain how to remove them.

Classical strategy for removing duplicates. A classical trick used in NFS is to reduce the search space by enforcing a positive sign to the first coordinate a. Indeed, when looking at K_h-unit-duplicates, we are concerned with elements $u \in \mathcal{O}_{K_h}^*$ and in the classical NFS setup, $\mathcal{O}_{K_h}^* = \mathbb{Z}^* = \{-1, 1\}$. Enforcing $a > 0$ reduces the search space by a factor 2 and avoids all unit-duplicates.

The situation is more complicated in TNFS as the number of units to consider is greater than 2. It is still possible to restrict to positive coefficients in order to avoid duplicates resulting from the units $\{\pm 1\}$ and we see in Section 4 that the enumeration algorithm indeed only considers half of the vectors in $\mathcal{L}_{\mathfrak{Q},\mathfrak{p}} \cap \mathcal{S}$. However, we are left with the following open question. Is there a systematic way to identify and thus remove duplicates generated from units other than ± 1? The difficulty of answering this question comes not only from the large number of units but also from the fact that the units must be small enough in order to produce a relation. Indeed, if u is too large, then (ua, ub) will be outside the sieving region and we do not have to worry about it.

Before giving more details on how we deal with these problematic K_h-unit-duplicates and the ζ_2-duplicates, we give our methodology to identify them. We

remind the reader that removing duplicates happens before the batch smoothness and thus we are looking at the set of sieve-survivors.

Our strategy to identify K_h-unit-duplicates and ζ_2-duplicates. For each pair (a, b) that is a sieve-survivor, we compute the value $k := a/b \pmod h \in K_h$, and store it in a hash table. If (a, b) and (a', b') are either K_h-unit-duplicates or ζ_2-duplicates, then they have the same index k. The hash table allows us to quickly identify if a given pair (a', b') is a duplicate of a previously seen (a, b) pair.

Remark 5. This method is done "locally" for every special-q. Indeed, one could think of adapting the idea of a general hash table regrouping relations from every special-q considered but the memory cost would be exceedingly high.

This method also justifies the choice of where in Algorithm 1 we test for duplicates. Indeed, computing k is not cost-free thus we want to avoid this computation for every pair (a, b) outputted by the enumeration algorithm. It is however less costly than computing an exact norm, so we compute it before.

However, the method brings forth the following issue. Duplicates can be seen as an equivalence class from which we want to select a unique representative. This representative of the class should be the "smallest" pair (a, b), meaning the (a, b)-pair which leads to the smallest norms. Indeed, a larger (a, b)-pair adds non-zero coefficients in the matrix of the linear system of relations and thus slows down the linear algebra step. For example, considering the $(\lambda a, \lambda b)$-pair, we have $N_i(\lambda a, \lambda b) = N_i(a, b)N_{K_h}(\lambda)$ for $i = 1, 2$ with the additional term $N_{K_h}(\lambda)$ with respect to the (a, b)-pair. This additional term yields extra ideals in the prime ideal decomposition, thus non-zero coefficients in the matrix. Our method does not necessarily keep the smallest pair. Indeed, if $(\lambda a, \lambda b)$ for $\lambda \in \mathcal{O}_{K_h}$ is already in the hash table, and if the algorithm sees the pair (a, b) afterwards, it will discard it and keep $(\lambda a, \lambda b)$.

The removal of special-q duplicates is easier when the representative of a duplicate class is in its canonical form. Indeed, special-q duplicates are removed by simply comparing the lines in the file that encodes the relations. Thus if different special-q's produce the same relation but each keep a different representative, say (a, b) for one and $(\lambda a, \lambda b)$ for the other, then their prime ideal decomposition will differ by some factors corresponding to $N_{K_h}(\lambda)$ and thus the duplicate will be kept. To identify the "smallest" (a, b)-pair in a ζ_2-duplicates class of equivalence, the most intuitive idea is to consider the notion of a primitive pair.

Definition 3. *A pair (a, b) is primitive if there exists no $\lambda \in \mathcal{O}_{K_h} \setminus \mathcal{O}_{K_h}^*$ such that $a = \lambda a'$ and $b = \lambda b'$ with $a', b' \in \mathcal{O}_{K_h}$.*

In NFS, we simply keep the (a, b)-pairs such that $\gcd(a, b) = 1$. The situation is more problematic in TNFS as the notion of gcd exists at the level of ideals, but not for $a(\iota)$ and $b(\iota)$. Consequently we propose to detect non-primitive pairs by computing the gcd of their norms: if $\gcd(N_1(a, b), N_2(a, b)) = 1$, then the (a, b)-pair is primitive. Indeed if one considers $(\lambda a, \lambda b)$ which is clearly non-primitive, we have

$$\gcd\left(N_1(\lambda a, \lambda b), N_2(\lambda a, \lambda b)\right) \geq N_{K_h}(\lambda)^{\min(\deg f_1, \deg f_2)} \neq 1.$$

Because we store and compare pairs in our hash table, we are left with a unique representative per class but we cannot be certain that this pair is primitive. Indeed, the method that computes the indices k keeps the first pair it encounters with that index and ignores all the following regardless of their primitiveness. Hence, on line 7 of Algorithm 1, if an (a, b)-pair survives the K_h-unit duplicates and ζ_2-duplicates elimination, we check whether our representative is primitive and if not, try to make it so, using Algorithm 2.

Algorithm 2 Primitive representative for each class of duplicates

Input: (a, b)-pair corresponding to a sieve-survivor
Output: primitive (a, b)-pair corresponding to the same sieve-survivor, or Fail

```
1: Compute gcd(N_1(a,b), N_2(a,b))
2: if gcd(N_1(a,b), N_2(a,b)) = 1 then
3:     Return (a,b)-pair.
4: else
5:     for each prime ℓ| gcd(N_1(a,b), N_2(a,b)) do
6:         Try to find β in O_{K_h} of norm ℓ such that a/β and b/β are in O_{K_h}.
7:         if Such a value β is found then
8:             a ← a/β and b ← b/β
9:             Recompute gcd(N_1(a,b), N_2(a,b))
10:            if gcd(N_1(a,b), N_2(a,b)) = 1 then
11:                Return new (a,b)-pair
12:        else
13:            Return Fail
```

Remark 6. A few comments can be made on the above algorithm.

- In Algorithm 2 we use the fact that if $\gcd(N_1(a, b), N_2(a, b)) = 1$, then the (a, b)-pair is primitive. We actually have an equivalence if the number field K_h is principal. In particular, in our computation, the field K_h is principal which ensures we do not throw away too many relations by using Algorithm 2.
- The elements of K_h of norm ℓ are hardcoded in our code for our specific computation for primes ℓ up to 43.
- If for a given (a, b)-pair, once we have reached $\ell = 43$ and the gcd is still not 1, we simply remove the relation. Therefore the algorithm may still fail, even though K_h is principal.
- Finding the corresponding primitive pair can be done in an easier way with PARI or Sage by solving norm equations to find β in line 7. We avoid this in our code to minimize the dependencies to other libraries. Improving this algorithm and making it robust to non-principal K_h is left for future work.

We have presented how to detect ζ_2-duplicates and K_h-unit duplicates. For ζ_2-duplicates we can keep a unique representative and make sure that representative is in a primitive form. Unfortunately, Algorithm 2 does not work for

finding a unique representative with respect to K_h-unit duplicates. Indeed, if γ is a unit, then $N_{K_h}(\gamma) = 1$. In this case, we simply rely on the prime ideal decomposition which is unique in an equivalence class of K_h-unit duplicates.

4 Relation collection with lattice enumeration

Recall we can select polynomials ϕ that are good candidates that lead to potential relations as soon as we enumerate all vectors $\mathbf{c}$ in $\mathcal{L}_{\mathfrak{Q},\mathfrak{p}} \cap \mathcal{S}$. Different enumeration techniques exist in the literature which depend on the shape of the sieving region $\mathcal{S}$ and the dimension d of the lattice $\mathcal{L}_{\mathfrak{Q},\mathfrak{p}}$. For NFS, usually $d = 2$ since $(a, b) \in \mathbb{Z}^2$ are not polynomials. Higher dimensions can also be considered in theory to target medium characteristics finite fields. When $d = 2$, thus for previous records using NFS, the sieving method of Franke and Kleinjung [13] is very efficient. However, in this article, we focus on methods that can be used in higher dimensions. Indeed, as shown above, for TNFS we have $d = \dim M_{\mathfrak{Q},\mathfrak{p}}$. Taking the polynomials $a(\iota)$ and $b(\iota)$ of degree $\deg h - 1$ leads to $d = 2 \times \deg h$ hence $d \geq 4$. There exist two competitive algorithms that can be used when $d \geq 3$: the transition vectors method [16] and the recursive hyperplane one [31], see Section 4.1 for these algorithms. They both use as a sieving space a d-orthotope whereas in this work we consider a d-sphere. Section 4.2 justifies our choice. We use the notation $\mathcal{S} = S_d(R)$ to indicate we are working in a d-sphere of radius R or simply S_d to lighten the notation when possible. Section 4.3 describes our new algorithm adapted for TNFS.

4.1 Existing algorithms to enumerate vectors in $\mathcal{L}_{\mathfrak{Q},\mathfrak{p}} \cap \mathcal{S}$

We start by giving a short refresher of two other recent competitive methods that can be used when $d \geq 3$: the transition vectors method [16] and the recursive hyperplane one [31].

Transition vectors for lattice sieving in [16]. In 2018, Grémy suggested a sieving algorithm inspired by Franke-Kleinjung's algorithm in dimension 2 but extended to higher dimensions. Let $\mathcal{S}$ be the sieving space considered, in this case, a d-orthotope defined as the product of intervals $\mathcal{S} = [H_0^m, H_0^M[\times \cdots, [H_{d-1}^m, H_{d-1}^M[$ for fixed bounds H_k^m, H_k^M.

The key notion used by Grémy to enumerate vectors of a lattice $\mathcal{L}$ is the notion of transition-vectors, allowing to jump from vector to vector in order to reach all elements in $\mathcal{L}_{\mathfrak{Q},\mathfrak{p}} \cap \mathcal{S}$. The transition-vectors are divided into d subsets $T_1, \cdots, T_d$, with T_k the set of k-transition-vectors for $k = 1, \cdots, d$. The latter have at least a non-zero k-coordinate and the last $d - k$ coordinates all equal to 0. The algorithm starts from $(0, 0, \cdots, 0) \in \mathcal{L}_{\mathfrak{Q},\mathfrak{p}}$ and enumerates all vectors in $\mathcal{L}_{\mathfrak{Q},\mathfrak{p}} \cap \mathcal{S}$ by adding or subtracting transition-vectors. It starts with vectors of T_1 until it reaches the edges of $\mathcal{S}$, then looks at additions (or subtractions) of vectors of T_2 etc, increasing from 1 to d step by step.

More precisely, a k-transition vector $\mathbf{t}$ is a non-zero vector of the lattice $\mathcal{L}$ such that there exists vectors $\mathbf{v}$ and $\mathbf{v}'$ in $\mathcal{L} \cap \mathcal{S}$ satisfying $\mathbf{v}' = \mathbf{v} + \mathbf{t}$ with the following two conditions: the last $d-k$ coordinates of $\mathbf{v}$ and $\mathbf{v}'$ are the same, and the coordinate v'_k is the smallest possible value greater than v_k. The addition (similarly subtraction) of k transition-vectors is illustrated below.

$$\begin{aligned} \mathbf{v} + \mathbf{t} &= (v_1, v_2, \cdots, v_k, v_{k+1}, \cdots, v_d) \\ &\quad + (*, *, \cdots, *, 0, \cdots, 0) \\ = \quad \mathbf{v}' &= (v'_1, v'_2, \cdots, v'_k, v_{k+1}, \cdots, v_d) \end{aligned}$$

The enumeration algorithm reaches all vectors in the intersection by adding and subtracting k-transition vectors for $k = 1, 2, \cdots, d$. The algorithm starts with the nul vector (which trivially belongs to $\mathcal{L}$) and adds (and subtracts) k-transition vectors until the edges of the intervals $[H_k^m, H_k^M[$ are reached.

In most cases, producing the entire set T is not possible, and thus the notion of transition-vectors is relaxed into nearly-transition-vectors. This variant is effective, but no longer reaches all vectors. A fall-back strategy is then considered when the algorithm fails to find an appropriate nearly-transition vector.

In dimension 4, this method seems to have sufficient prospects of success. However, even with the relaxed variant, experiments ran in dimension 6 in [16] point to the limits of this method due to the poor quality of the nearly-transition-vectors and the number of calls required to the dedicated fall-back strategy. [16] concluded that "*using cuboid search is probably a too hard constraint that implies the hardness or even an impossibility for the sieving process*".

Recursive lattice sieving through hyperplanes in [31]. In 2020, McGuire and Robinson also proposed an enumeration algorithm in dimension 3 or higher. The sieving area is again a d-orthotope $\mathcal{S} = [0, H[\times [-H, H] \cdots \times [-H, H[$ for a fixed bound H. To enumerate all the vectors in $\mathcal{L}_{\mathfrak{Q},\mathfrak{p}} \cap \mathcal{S}$ the main idea consists in dividing the search space into hyperplanes, and enumerating in each of them. Minimizing the number of hyperplanes to visit is done by adequately choosing a "ground" hyperplane and then considering translations of it.

More precisely, in dimension 3 the "ground" plane G_0 is defined as a plane spanned by the two shortest vectors $\mathbf{c}_1, \mathbf{c}_2$ of $\mathcal{L}_{\mathfrak{Q},\mathfrak{p}}$ through the origin. Because of the small dimension of $\mathcal{L}_{\mathfrak{Q},\mathfrak{p}}$, these shortest vectors are easily found with LLL. One then enumerates every point in $G_0 \cap \mathcal{S}$ before moving to the next translated plane: $G_1 = G_0 + \mathbf{c}_3, G_2 = G_0 + 2\mathbf{c}_3, \cdots, G_k = G_0 + k\mathbf{c}_3$ until a k is reached such that $G_k \cap \mathcal{S} = \emptyset$. For each translated plane, one enumerates points in $G_k \cap \mathcal{S}$.

As we understand it, these short vectors serve a similar purpose as Grémy's transition-vectors: the aim is to choose relevant vectors to add (or subtract) to others while being as exhaustive as possible. Similarly to [16], the enumeration here is not completely exhaustive. Indeed, in [31], the authors report consistently missing around 1.8 % of the lattice points per special-q due to corner cases.

The main speed gain of this algorithm compare to Grémy's algorithm is not so much in the enumeration done in each translated hyperplane, but in the number of hyperplanes that are being considered. Indeed, taking the largest vectors

of the basis for the direction of the hyperplane translations allows to minimize the number of hyperplanes required to cover $\mathcal{S}$.

Algorithm 3 Recursive version of enumeration algorithm from [31]

Input: the basis of a lattice $\mathcal{L}$ of dimension d, a sieving region $\mathcal{S}$.
Output: list L of vectors in $\mathcal{L} \cap \mathcal{S}$

def enum(d, $\mathcal{S}$, $[\mathbf{b}_1, \mathbf{b}_2, \cdots, \mathbf{b}_d]$)

$L = \{\}$
if $d \neq 1$ **then**
  $k = 0$
  $P = \text{plane}(\mathbf{0}, \mathbf{b}_1, \mathbf{b}_2, \cdots, \mathbf{b}_{d-1})$
  $e_{\max} = \max\{e \in \mathbb{N} : \mathcal{S} \cap (P - e \cdot \mathbf{b}_d) \neq \emptyset\}$
  $G_0 = P - e_{\max} \cdot \mathbf{b}_d$
  while $G_k \cap \mathcal{S} \neq \emptyset$ **do**
    $L' \leftarrow \text{enum}(d-1, G_k \cap \mathcal{S}, [\mathbf{b}_1, \mathbf{b}_2, \cdots, \mathbf{b}_{d-1}])$
    Append L' to L
    $k = k + 1$
    $G_{k+1} = G_k + \mathbf{b}_d$
if $d = 1$ **then**
  Find $p_0 \in \text{plane}(\mathbf{0}, \mathbf{b}_1) \cap \mathcal{S}$ with linear programming
  Add p_0 to L
  $e_{\max} = \max\{e \in \mathbb{N} : \mathcal{S} \cap (p_0 - e \cdot \mathbf{b}_1) \neq \emptyset\}$
  Define $P_0 = p_0 - e_{\max} \cdot \mathbf{b}_1$
  while $P_0 \cap S \neq \emptyset$ **do**
    $P_0 = P_0 + \mathbf{b}_1$
    Add P_0 to L
return L.

Pseudo-code for dimension 3 (only) is given in [31]. Although the authors state their algorithm can be extended to higher dimension, we wonder whether it remains efficient when $d \geq 3$. We write in Algorithm 3 a pseudo-code of our understanding of how their method can be adapted for any dimension d. One difficulty we see is finding $e_{\max}$ when enumerating in $G_k \cap \mathcal{S}$, which increases with d and the task can become too expensive very quickly. Indeed, finding $e_{\max}$ can be done using integer linear programming, which is doable in low dimension but should be very hard (or at least more costly than desired) as d grows.

4.2 Why do we choose a *d*-sphere?

Let d be the dimension of the sieving space. Consider a d-sphere S_d and a d-orthotope C_d of equal volume. The number of vectors $\mathbf{c}$ of $\mathcal{L}_{\mathfrak{Q},\mathfrak{p}} \cap \mathcal{S}$ to enumerate is thus approximately the same if we consider $\mathcal{S}$ to be S_d or C_d. Let us assume that the size of the norms is only dependent on the size of the coordinates of the vectors $\mathbf{c}$. We now argue that using S_d instead of C_d leads to smaller norms.

Recall that the volume of a d-sphere is given by

$$V_d(R) = \frac{\pi^{d/2} R^d}{\Gamma(d/2+1)},$$

and the volume of a d-hypercube of fixed length L is L^d. We use a d-hypercube instead of a d-orthotope to simplify the presentation. In order to have the same sieving volume, *i.e.*, $V_d(R) = L^d$ we must have

$$R = L \cdot \Gamma(d/2+1)^{1/d} \cdot \pi^{-1/2}.$$

For the hypercube, the length of half the diagonal (from the center) is given by $D = L \cdot \sqrt{d}/2$. The distance between the vertices of the hypercube and the sphere is expressed as $D - R = L \cdot \sqrt{d}/2 - L \cdot \Gamma(d/2+1)^{1/d} \cdot \pi^{-1/2}$ and from this last equality we see

$$\lim_{d \to \infty} (D - R) = \infty.$$

Let $P_d = C_d \setminus S_d$ and $Q_d = S_d \setminus C_d$. The quantity $D - R$ represents an upper bound on the distance from the origin to points in P_d, which would correspond to the largest norms. Hence, if we want to consider smaller norms, when $d \to \infty$ it is more advantageous to consider points in Q_d, and thus choosing S_d rather than C_d is a more suitable choice. This is illustrated in Figure 6.

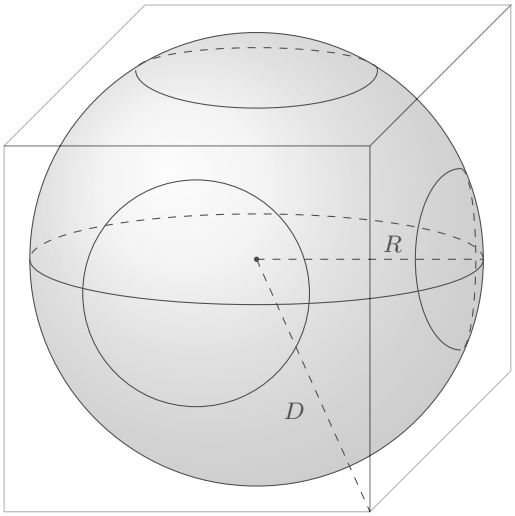

Fig. 6: Hypercube and d-sphere for $d = 3$ of equal volume.

4.3 Schnorr-Euchner's enumeration algorithm for TNFS

In order to find potential relations, recall that we enumerate all the vectors of bounded norms in the lattice $\mathcal{L}_{\mathfrak{Q},\mathfrak{p}}$, where $\mathcal{L}_{\mathfrak{Q},\mathfrak{p}}$ translates the notion of divisibil-

ity by an ideal $\mathfrak{p}$ and a special-q ideal $\mathfrak{Q}$. By enumerating vectors in $\mathcal{L}_{\mathfrak{Q},\mathfrak{p}} \cap S_d(R)$ for many different $\mathfrak{p}$ (each generating a different $\mathcal{L}_{\mathfrak{Q},\mathfrak{p}}$) one can identify vectors divisible by many $\mathfrak{p}$'s and thus more likely to correspond to B-smooth norms.

Let us fix $\mathfrak{p}$ and a special-q ideal $\mathfrak{Q}$. Given an LLL-reduced basis $\{\mathbf{b}_1, \cdots, \mathbf{b}_d\}$ of $\mathcal{L}_{\mathfrak{Q},\mathfrak{p}}$ and the radius R of a d-sphere S_d which corresponds to the sieving area, we propose to find these vectors thanks to an adaptation of Schnorr-Euchner's enumeration algorithm [38]. We choose to follow [38] instead of Fincke-Pohst-Kannan's algorithm [12, 26] as it appears more efficient operation-wise.

Description of the algorithm. Schnorr-Euchner's algorithm constructs an enumeration tree of depth d where the leaves correspond to our vectors in $\mathcal{L}_{\mathfrak{Q},\mathfrak{p}} \cap S_d$, *i.e.,* the vectors $\mathbf{c} = \sum_{i=1}^{d} v_i \mathbf{b}_i$ that satisfy $||\mathbf{c}|| \leq R$.

To construct the tree, the algorithm considers projections of the lattice $\mathcal{L}_{\mathfrak{Q},\mathfrak{p}}$. Since the norm of vectors cannot increase under orthogonal projections, the enumeration algorithm proceeds recursively by looking at the orthogonal projections π_k on the set $\{\mathbf{b}_1, \cdots, \mathbf{b}_{k-1}\}^{\perp}$ for decreasing values of k and where π_1 is the identity.

More precisely, using the Gram-Schmidt relation $\mathbf{b}_i = \mathbf{b}_i^* + \sum_{j=1}^{i-1} \mu_{i,j} \mathbf{b}_j^*$, a vector $\mathbf{c}$ can be re-written as

$$\mathbf{c} = \sum_{i=1}^{d} \left(v_i \mathbf{b}_i^* + \sum_{j=1}^{i-1} v_i \mu_{i,j} \mathbf{b}_j^* \right) = \sum_{j=1}^{d} \left((v_j + \sum_{i=j+1}^{d} \mu_{i,j} v_j) \mathbf{b}_j^* \right),$$

where the vectors $\mathbf{b}_i^*$ correspond to the Gram-Schmidt orthogonalization of the basis vectors $\mathbf{b}_i$ and the $\mu_{i,j}$ are the Gram-Schmidt coefficients. The internal nodes of the tree then correspond to projections of the vector $\mathbf{c}$. The projection of the vector $\mathbf{c}$ for a given $k = 1, 2, \cdots, d$ is given by

$$\pi_k(\mathbf{c}) = \sum_{j=1}^{d} \left((v_j + \sum_{i=j+1}^{d} \mu_{i,j} v_i) \pi_k(\mathbf{b}_j^*) \right) = \sum_{j=k}^{d} \left((v_j + \sum_{i=j+1}^{d} \mu_{i,j} v_i) \mathbf{b}_j^* \right),$$

At each level k of the tree, the algorithm verifies that $||\pi_k(\mathbf{c})|| \leq R$ which can be reduced to enumerating admissible values of v_k that lie in a bounded interval. Indeed, searching for all plausible v_k coefficient such that $||\pi_k(\mathbf{c})||^2 \leq R^2$ reduces to enumerating in the interval given by Equation 2.

$$\left| v_k + \sum_{i=k+1}^{n} \mu_{i,k} v_i \right| \leq \frac{\sqrt{R^2 - \sum_{j=k+1}^{n} (v_j + \sum_{i=j+1}^{n} \mu_{i,j} v_i)^2 ||\mathbf{b}_j^*||^2}}{||\mathbf{b}_k^*||}. \quad (2)$$

If the condition $||\pi_k(\mathbf{c})||^2 \leq R^2$ is satisfied, the algorithm explores the descendants (*i.e.,* the algorithm moves to the $k-1$ level). Otherwise, the subtree is ignored. Once we reach the leaves, and the condition is still true, we have found a vector in $\mathcal{L}_{\mathfrak{Q},\mathfrak{p}} \cap S_n$. The algorithm visits half the nodes since if $\mathbf{c} \in \mathcal{L}_{\mathfrak{Q},\mathfrak{p}}$ then $-\mathbf{c} \in \mathcal{L}_{\mathfrak{Q},\mathfrak{p}}$. The pseudo-code for the full algorithm in our context is given in Algorithm 4.

Efficiently computing the vectors $\boldsymbol{c} = \boldsymbol{v} \cdot M_{\mathfrak{Q},\mathfrak{p}}$. The algorithm works with the coefficient vectors $\mathbf{v} = (v_1, \cdots, v_d)$. However, in the end, we do not want the combinations $\mathbf{v}$, but the vectors

$$\mathbf{c} = \mathbf{v} \cdot M_{\mathfrak{Q},\mathfrak{p}} = \sum_{i=1}^{d} v_i \mathbf{b}_i.$$

Computing these vectors $\mathbf{c}$ can either be done naively, at the leaf level by explicitly computing $\mathbf{c} = \sum_{i=1}^{d} v_i \mathbf{b}_i$ for each leaf, or one can keep track of a partial sum $\sum_{i=t}^{d} v_i \mathbf{b}_i$ for a fixed value t chosen as input to the algorithm and update the quantity $v_i \mathbf{b}_i$ once a v_i is changed during the algorithm, *i.e.,* once the algorithm visits a new internal node in levels t to d. We opt for the second option as it reduces the overall cost of enumeration.

More precisely, let $\texttt{common\_part} = \sum_{i=t}^{d} v_i \mathbf{b}_i$, where each $v_i \mathbf{b}_i$ is stored in a variable. Each time the algorithm visits a new internal node, thus updates v_i for a given $i = t, \cdots, d$, the algorithm updates `common_part` by subtracting the current $v_i \mathbf{b}_i$, computing the new $v_i \mathbf{b}_i$ with the new value of v_i and adding it back to `common_part`. Once at the leaf, in order to compute the vector $\mathbf{c}$, it remains to compute $\mathbf{c} = \sum_{i=1}^{t-1} v_i \mathbf{b}_i + \texttt{common\_part}$.

For most values of p we are concerned about, we use this optimized code with $t = 2$, thus updating all values $v_i \mathbf{b}_i$ during the algorithm, except at the leaf level, and finally computing $\mathbf{c} = v_1 \mathbf{b}_1 + \texttt{common\_part}$. When p becomes large and few leaves are found, it can be less efficient to choose $t = 2$, and thus one selects the appropriate $t > 2$ in order to optimize the number of operations performed for this computation. More details are given in the Section 5 below and the pseudo-code for the optimized enumeration algorithm is given in Algorithm 4.

Remark 7. This optimization makes sense in this specific context where our lattices are of small dimension and often dense (in particular for small primes). This would not translate well for general lattices of larger dimensions or if only a handful of small vectors were output.

5 Analysis of the enumeration algorithm

5.1 Number of leaves, nodes and enumeration cost

We now estimate the cost of our enumeration algorithm. This implies having an estimate of the number of nodes and leaves in the enumeration tree. This estimate is derived using the Gaussian heuristic. In order to do so, it is necessary to analyze the geometry of the input lattice $\mathcal{L}_{\mathfrak{Q},\mathfrak{p}}$. In particular, we are interested in the ratio between the norms of two consecutive Gram-Schmidt vectors of the (reduced) lattice. Indeed, to count the number of nodes in the enumeration tree, we need to compute the volume of the projected lattices which is given by the product of the norms of the Gram-Schmidt vectors.

The dimension of the lattices we consider is small, *i.e.,* precisely 6 in our computation but plausible dimensions are 4, 6 or 8 for other realistic targets.

Algorithm 4 Optimized enumerating $\mathcal{L}_{\mathfrak{Q},\mathfrak{p}} \cap S_d$

Input: LLL-reduced basis $\{\mathbf{b}_1, \cdots, \mathbf{b}_d\}$ of $\mathcal{L}_{\mathfrak{Q},\mathfrak{p}}$, radius R of d-sphere S_d, variable t for optimization.
Output: List K of vectors $\mathbf{c} \in \mathcal{L}_{\mathfrak{Q},\mathfrak{p}} \cap S_d(R)$.

Pre-computation: compute all Gram-Schmit coefficients $\mu_{i,j}$ for $i < j$ and the norms of the Gram-Schmidt vectors $||\mathbf{b}_i^*||^2$ for all $i \leq d$.
$K \leftarrow \{\}$, $\sigma \leftarrow (0)_{(d+1)\times d}$, $r_0 = 0, r_1 = 1, \cdots, r_d = d$, $v_1 = 1$, $v_2 = \cdots = v_d = 0$.
$\rho_1 = \rho_2 = \rho_{d+1} = 0$ ▷ with $\rho_k = ||\pi_k(\mathbf{c})||^2$
$c_1 = \cdots = c_d = 0$ ▷ with $c_k = \sum_{i=k+1}^{d} \mu_{i,k} v_i$
$w_1 = \cdots = w_d = 0$
`last_nonzero` $= 1$, `common_part` $= v_t\mathbf{b}_t + \cdots + v_d\mathbf{b}_d$
$k = 1$
while true **do**
    $\rho_k = \rho_{k+1} + (v_k - c_k)^2 \, ||\mathbf{b}_k^*||^2$
    if $\rho_k \leq R^2$ **then**
        if $k = 1$ **then**
            $\mathbf{c} = \sum_{i=1}^{t-1} v_i\mathbf{b}_i +$ `common_part` ▷ opt. computation of $\mathbf{c}$
            $K \leftarrow K \cup \mathbf{c}$
            if `last_nonzero` $= 1$ **then**
                Skip ▷ this generates ζ_2-duplicates
            else
                if $v_k > c_k$ **then** $v_k \leftarrow v_k - w_k$
                else
                    $v_k \leftarrow v_k + w_k$
                $w_k \leftarrow w_k + 1$
        else
            $k \leftarrow k - 1$ ▷ we go down the tree
            $r_k \leftarrow \max(r_k, r_{k+1})$
            for $i = r_{k+1}$ to $k + 2$ **do**
                $\sigma_{i,k} \leftarrow \sigma_{i+1,k} + v_i\mu_{i,k}$
            $c_k \leftarrow -\sigma_{k+1,k}$
            $v_k = \lceil c_k \rfloor$, $w_k = 1$.
            if $k = \ell$ for $\ell = t, \cdots, d$ **then**
                Re-compute `common_part` by updating $v_\ell\mathbf{b}_\ell$.
    else
        $k \leftarrow k + 1$ ▷ going back up the tree.
        if $k = d + 1$ **then**
            return K ▷ we find no more solutions
        $r_k \leftarrow k$
        if $k \geq$ last_nonzero **then**
            last_nonzero $\leftarrow k$
            $v_k \leftarrow v_k + 1$
            if $k = \ell$ for $\ell = t, \cdots, n$ **then**
                Re-compute `common_part` by updating $v_\ell b_\ell$.
        else
            if $v_k > c_k$ **then** $v_k \leftarrow v_k - w_k$
                if $k = \ell$ for $\ell = t, \cdots, n$ **then**
                    Re-compute `common_part` by updating $v_\ell b_\ell$.
            else
                $v_k \leftarrow v_k + w_k$
                if $k = \ell$ for $\ell = t, \cdots, n$ **then**
                    Re-compute `common_part` by updating $v_\ell b_\ell$.
            $w_k \leftarrow w_k + 1$

Because of these small dimensions, we observe that classical analyses of lattice reduction algorithms do not hold. For example, an expected lower bound β on the ratio $||\mathbf{b}_{i+1}^*||^2/||\mathbf{b}_i^*||^2$ was observed in [33] for vectors outputted from a reduction algorithm. The constant β depends on the reduction algorithm considered and in the case of LLL, we have $\beta = 1/(\delta - \eta^2)$. Sage's default LLL implementation uses $\delta = 0.99$ and $\eta = 0.501$, thus $\beta = 1.35$. This value is obtained for random bases. Our lattices $\mathcal{L}_{\mathfrak{Q},\mathfrak{p}}$ are however not random. We thus experimentally measured that for 6-dimensional lattices, the ratio $||\mathbf{b}_{i+1}^*||/||\mathbf{b}_i^*||$ is smaller than expected, hence we introduce the following heuristic.

Heuristic 1 *For 6-dimensional lattices $\mathcal{L}_{\mathfrak{Q},\mathfrak{p}}$, $||\boldsymbol{b}_{i+1}^*||/||\boldsymbol{b}_i^*|| \approx 1.09$ on average.*

In what follows, we need to estimate the number of lattice vectors in a sphere. For this, we rely on the Gaussian heuristic, which tells us that the number of points belonging to the intersection of a lattice $\mathcal{L}$ and a set $\mathcal{S}$ is roughly the ratio of the volumes, *i.e.,* $\mathrm{vol}(\mathcal{S})/\mathrm{vol}(\mathcal{L})$. This heuristic was suggested to analyze enumeration algorithms in [21] and experimentally confirmed to be accurate in [14] for random lattices.

Number of leaves. The volume of a full-rank d-dimensional lattice $\mathcal{L}$ is given by $\det(\mathcal{L}) = \prod_{i=1}^{d} ||\mathbf{b}_i^*||$ and in our case the volume of $\mathcal{L}_{\mathfrak{Q},\mathfrak{p}}$ is p. Using the Gaussian heuristic, and taking into account the fact that we visit only half of the tree, the number of leaves is thus given by

$$\Xi_{\text{leaves}} \approx \frac{1}{2}\frac{\mathrm{vol}(S_d(R))}{\det(\mathcal{L}_{\mathfrak{Q},\mathfrak{p}})} = \frac{R^d \pi^{d/2}}{2\Gamma(d/2+1)p}.$$

Number of nodes. Let Ξ_k denote the number of nodes at level k which corresponds to the number of points in $\pi_k(\mathcal{L}_{\mathfrak{Q},\mathfrak{p}}) \cap S_k(R)$. From the Gaussian heuristic and dividing by 2 for the half-tree, we have $\Xi_k = |\pi_k(\mathcal{L}_{\mathfrak{Q},\mathfrak{p}}) \cap S_{d-k+1}(R)|$ and

$$\Xi_k = \mathrm{vol}(S_{d-k+1}(R))/(2 \cdot \mathrm{vol}(\pi_k(\mathcal{L}_{\mathfrak{Q},\mathfrak{p}}))).$$

The volume of the projected lattice $\pi_k(\mathcal{L}_{\mathfrak{Q},\mathfrak{p}})$ is $\prod_{i=k}^{d} ||\mathbf{b}_i^*||$, and we can use Heuristic 1 to estimate it. We get

$$\mathrm{vol}(\pi_k(\mathcal{L}_{\mathfrak{Q},\mathfrak{p}})) \approx ||\mathbf{b}_1||^{d-k+1}(1.09)^{\sum_{i=k-1}^{d-1} i} \approx ||\mathbf{b}_1||^{d-k+1}(1.09)^{0.5(d-k+1)(d+k-2)}.$$

Since for $k = 1$ we have $\mathrm{vol}(\pi_1(\mathcal{L}_{\mathfrak{Q},\mathfrak{p}})) = p$, we set $||\mathbf{b}_1|| \approx p^{1/d}/(1.09)^{(\sum_{i=1}^{d-1} i)/d}$. We then have

$$\mathrm{vol}(\pi_k(\mathcal{L}_{\mathfrak{Q},\mathfrak{p}})) \approx p^{(d-k+1)/d}(1.09)^{\sum_{i=k-1}^{d-1} i - ((d-k+1)/d)\sum_{i=1}^{d-1} i}$$

that leads to $\mathrm{vol}(\pi_k(\mathcal{L}_{\mathfrak{Q},\mathfrak{p}})) = p^{(d-k+1)/d}(1.09)^{0.5(d-k+1)(k-1)}$. We therefore get

$$\Xi_k \approx \frac{R^{d-k+1}\pi^{(d-k+1)/2}}{2 \cdot \Gamma((d-k+1)/2+1) \cdot p^{(d-k+1)/d} \cdot (1.09)^{0.5(d-k+1)(k-1)}}.$$

Finally, the total number of nodes is $\Xi = \sum_{k=1}^{d} \Xi_k$. Experimental verification of these formulae are provided in Section 6.

Running time of enumeration. The running time of our enumeration algorithm is given by the number of nodes Ξ times the number of operations per node. At each node, the algorithm performs 7 arithmetic operations on average to compute and update the linear combinations $\mathbf{v}$. In addition, one must also compute the vector $\mathbf{c} = \mathbf{v} \cdot M_{\mathfrak{Q},\mathfrak{p}} = \sum_{i=1}^{d} v_i \mathbf{b}_i$. As mentioned above, this can either be done naively at the leaf level by explicitly computing $\mathbf{c} = \sum_{i=1}^{d} v_i \mathbf{b}_i$ for each leaf, which costs $2d^2 - 1$ extra operations per leaf.

Or, one uses $\texttt{common\_part} = \sum_{i=t}^{d} v_i \mathbf{b}_i$. Each time the algorithm visits a new internal node in the levels t up to d, thus updates v_i for a given $i = t, \cdots, d$, the algorithm performs $4d - 1$ operations: in order to update `common_part`, we subtract the current $v_i \mathbf{b}_i$ (d operations), compute the new $v_i \mathbf{b}_i$ ($2d - 1$ operations) with the new value of v_i and add it back to `common_part` (again, d operations).

Once at the leaf, in order to compute the vector $\mathbf{c}$, it remains to perform $(t-1)(2d-1) + t - 1$ operations, $\mathbf{c} = v_1\mathbf{b}_1 + \cdots + v_{t-1}\mathbf{b}_{t-1} + \texttt{common\_part}$. In summary, we have for the additional cost of computing the vector $\mathbf{c}$

$$\texttt{Comp c naively} = \Xi_{\text{leaves}} \times (2d^2 - 1)$$

but using `common_part`, the cost of computing the vector $\mathbf{c}$ is $\texttt{Comp c opt} = \#\text{ int. nodes}_{t \to d} \times (4d-1) + \Xi_{\text{leaves}} \times ((t-1)(2d-1) + t - 1)$ so

$$\texttt{Comp c opt} = \left(\sum_{i=t}^{d} \Xi_i \right) (4d-1) + \Xi_{\text{leaves}}((t-1)(2d-1) + t - 1).$$

We experimentally verify that the optimized code results in less operations than the naive one to compute all the vectors c for all but too large values of p when choosing $t = 2$. When p becomes too large and there aren't many leaves, the optimized code uses more operations than the naive one. One easy way to resolve this is to increase the value of t in the definition of `common_part`. However, this occurs when p is large enough that the predominant cost is in generating the lattice $\mathcal{L}_{\mathfrak{Q},\mathfrak{p}}$ and not in the enumeration algorithm. Finally, the total cost of enumeration on average is thus equal to:

$$\texttt{Cost enum} = 7 \times \Xi + \texttt{Comp c opt}.$$

Number of leaves per node. The number of leaves per node is given by $r = \Xi_{\text{leaves}}/\Xi$ as a function of p. This ratio r captures the behavior of our algorithm. The higher r is, the more efficient our algorithm becomes: we want this ratio to remain high as internal nodes correspond to (necessary) operations which do not produce any information as lattice vectors are seen only at the leaf level. When p increases, r decreases, as illustrated in Figure 8 for parameters specific to our computation. Indeed, the probability of a norm being divisible by a small prime is higher than for larger primes. Hence for small primes, r is close to 1. This explains why we enumerate on small primes first, and switch to batch algorithms for larger primes.

Comparing enumeration and construction of the lattice. Figure 7 illustrates the variation of the number of operations for the construction of the basis $M_{\mathfrak{Q},\mathfrak{p}}$ for the lattice $\mathcal{L}_{\mathfrak{Q},\mathfrak{p}}$ and the enumeration algorithm for increasing values of p and a fixed special-q. When p is small, the enumeration algorithm is more costly (in terms of number of operations) than constructing the lattice itself. However, when p becomes large enough, constructing the basis $M_{\mathfrak{Q},\mathfrak{p}}$ becomes much more costly. The intersection point varies depending on the radius R and can be chosen to be close to $p_{\max}$.

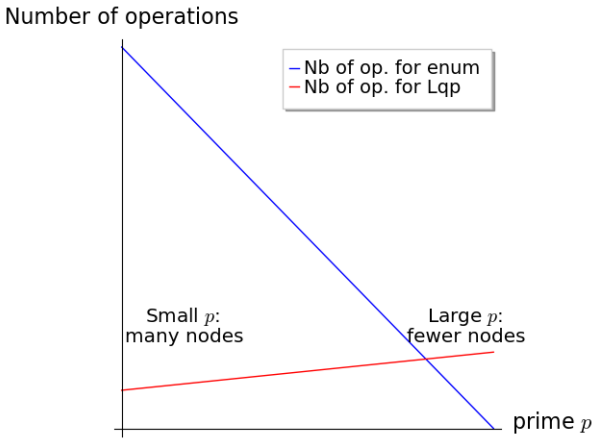

Fig. 7: Sketch of number of operations for enumeration and constructing a basis of $\mathcal{L}_{\mathfrak{Q},\mathfrak{p}}$ as a function of p. This plot illustrates the behaviour of these two costs.

5.2 Overall complexity of relation collection

The total cost of Algorithm 1 is the sum of the cost of constructing $\mathcal{L}_{\mathfrak{Q},\mathfrak{p}}$, the cost of enumerating in $\mathcal{L}_{\mathfrak{Q},\mathfrak{p}} \cap S$, and the costs of batch algorithm on the sieve-survivors and ECM on the batch-survivors. In order to optimize the overall complexity, it is important to correctly set the many parameters that come into play during this step. In particular, one must decide the size of (many) fixed parameters: the radius R, the smoothness bound B, the range of special-q's to consider, the bounds $p_{\max}$, p_{batch} and the balance between sieving, batch smoothness and ECM based and the size of the cofactors.

5.3 Comparison with previous methods

Our enumeration algorithm vs [16]. Grémy's algorithm uses a d-orthotope as sieving space, whereas we consider a d-sphere. As explained previously, we

believe that as the dimension increases, it is more efficient to sieve in a d-sphere as opposed to a d-orthotope. The number of nearly-transition vectors required in [16] for the algorithm to enumerate most of the vectors also increases with the dimension. These nearly-transition vectors are generated during the initialization of the enumeration procedure using various strategies. Moreover, [16] states that in dimension 6, the number of calls to the fall-back strategy is important, indicating that the nearly-transition-vectors are of poor quality, and thus the algorithm requires the use of skew-small-vectors (also to be computed). Finally, Grémy's algorithm is not exhaustive in its search of vectors in $\mathcal{L}_{\mathcal{Q},\mathfrak{p}} \cap \mathcal{S}$, and as the dimension increases, in addition to what was mentioned just before, we suspect the percentage of missing vectors increases as well.

Our enumeration algorithm vs [31]. Similarly as Grémy's algorithm, this algorithm also uses a d-orthotope as sieving space. Moreover, the algorithm presented in [31] is very similar to the classical enumeration algorithm of Fincke-Pohst-Kannan (FPK) [12, 26] adapted to a rectangular sieving region. One important cost in both FPK and this algorithm is finding the initial point in each plane from which the enumeration starts. In [31], this is done with linear programming. Every time the algorithm changes hyperplane, an integer linear programming problem must be solved. This does not add much complexity to the algorithm, but its cost is non-negligible with respect to the rest of the operations performed, and increases with the dimension. Our algorithm is based on Schnorr-Euchner's variant which starts its enumeration of a given interval at its center. This avoids the computation of the edge of the interval at each level as required in FPK or the linear programming cost.

Moreover, as the dimension grows, so does the number of hyperplanes. Thus, we believe that the algorithm would struggle to be competitive when this number becomes too large and a linear program must be solved for each hyperplane.

Finally, our algorithm is exhaustive by construction, and thus enumerates every single vector in $\mathcal{L}_{\Omega,\mathfrak{p}} \cap S$. As mentioned previously, the algorithm in [31] encounters boundary issues when the planes intersect only the corners of the sieving region. The loss is reasonable in dimension 3 but may become more and more problematic as the dimension grows.

6 A 521-bit computation

6.1 A target from the pairing world

Considering finite fields with composite extensions is highly motivated by pairing-based cryptography. The security of pairing-based protocols relies on both the discrete logarithm problem in the curve and in the finite field. MNT curves [32] are pairing-friendly elliptic curves with small embedding degree 6, meaning that the security of the related pairing-based protocols relies on the discrete logarithm problem in $\mathbb{F}_{p^6}^*$ for a prime p. Our target is precisely $\mathbb{F}_{p^6}^*$ with a 87-bit prime. MNT curves were introduced in the early 2000's but are back into the

spotlight due to the recent arising of zk-SNARKS within the zero-knowledge community, that brings other needs and other uses. For instance, the need of cycles of curves[3] in zk-SNARKS, that are currently only available with MNT curves explains why MNT-6 curves are so useful, as explained in Guillevic's blog-post [18]. Two of them are widely deployed: one with a 298-bit prime [5] and the other with a 753-bit prime [5,9]. With our 87-bit prime we are still far from these concrete parameters, but our work shows that in order to evaluate the security of these curves the right threat to consider is TNFS.

More precisely, we consider the 521-bit finite field $\mathbb{F}_{p^n}$ where $n = 6$ and $p = \texttt{0x6fb96ccdf61c1ea3582e57}$ is a 87-bit prime. The extension degree n is composite with factors $\eta = 3$ and $\kappa = 2$. The prime p is "random" in the sense that it is the closest prime to the 87 first bits of RSA-1024, the 1024-bit integer coming from the RSA Factoring Challenge, such that $p^2 - p + 1$ is also prime. Moreover, we choose as target an element in $\mathbb{F}_{p^6}$ whose decimal digits are taken from π:

$$\begin{aligned}\texttt{target} =&(314159265358979323846264338327950288419716939937510 5\iota \\ &+ 820974944592307816406286208\iota^2) + x(899862803482534211706798214 \\ &+ 808651328230664709384460955058223172535940812848111\iota^2).\end{aligned}$$

This does not fully define the element since this depends on the representation taken for $\mathbb{F}_{p^6}$. For this, we will simply choose the one that follows from the polynomial selection below.

Because the computation of a discrete logarithm in a group can be reduced to its computation in one of its prime subgroups by Pohlig-Hellman's reduction. We want to consider a prime order subgroup for which the computation is the hardest. To do so, we will work modulo

$$\ell = p^2 - p + 1 = \texttt{30c252a90b588491be0a93f6fd11924531a80adb333b},$$

the 174-bit prime order of a subgroup of the multiplicative group of our target finite field. Indeed, any other factor of $p^6 - 1$, namely $p - 1, p + 1$ and $p^2 + p + 1$, correspond to subfields in which DLP would be much easier to solve. For example, if ℓ divides $p - 1$, the elements considered in DLP belong to $\mathbb{F}_p^*$ and solving DLP with NFS in this prime field of 87 bits can be done in nearly negligible time. On the other hand, the value $p^2 - p + 1$ does not correspond to any subfields of $\mathbb{F}_{p^6}$.

6.2 Polynomial selection

Three polynomials with specific characteristics must be chosen for TNFS. The following polynomials were provided to us by Guillevic after careful consideration of which polynomial selection method would provide the best polynomials. As explained previously, in order to choose between the Conjugation method

[3] A cycle of curves is a pair of pairing-friendly elliptic curves $\mathcal{E}_1$, $\mathcal{E}_2$ such that $\mathcal{E}_1$ is defined over a finite prime field $\mathbb{F}_{p_1}$ with prime order p_2, and $\mathcal{E}_2$ is defined over the finite field $\mathbb{F}_{p_2}$ with order p_1.

or JLSV1, it suffices to compare the size of the norms N_1 and N_2 for many $(a(\iota), b(\iota))$-pairs. Indeed, the parameters of f_2 outputted by Conjugation are the same as the parameters of f_1 and f_2 outputted by JLSV1 and thus it suffices to compare the norms N_1, N_2 for the Conjugation method. Averaging over 10^5 $(a(\iota), b(\iota))$-pairs clearly showed that N_1 was much smaller than N_2, and thus the Conjugation method was finally considered to choose the polynomials. Code to reproduce these results can be found at https://gitlab.inria.fr/tnfs-alpha/alpha.git.

The polynomial h is of degree $\eta = 3$, monic and irreducible modulo p. In our computation, we use

$$h(\iota) = \iota^3 - \iota + 1.$$

This polynomial has the following property: $1/\zeta_2(K_h) = 0.9009$. We could have chosen a polynomial h of degree $\eta = 2$. This would have resulted in a sieving space of dimension 4 instead of 6. However, for a first large-scale experiment, the goal was to explore the efficiency of sieving in higher dimension, thus the choice of a degree 3 polynomial. We leave for future work the comparison of a TNFS computation with a polynomial h of degree 2.

The polynomials f_1, f_2 are selected thanks to the Conjugation method [2]. We recall that this method looks for polynomials of degree κ and 2κ. We get

$$f_1 = x^4 + 1,$$

and

$$f_2 = 11672244015875x^2 + 1532885840586x + 11672244015875.$$

These polynomials are irreducible over $\mathbb{Z}[\iota][x]$ and the polynomial f_2 corresponds to the irreducible common factor of these two polynomials, as it is the case for the Conjugation method.

6.3 Relation collection

Many parameters have to be balanced in practice in Algorithm 1 to optimise the relation collection step. The computational cost of sieving greatly varies depending on the balance between these parameters. In particular, one needs to fix a smoothness bound B, a range of special-q's to consider and a sieving space, which in our context reduces to choosing the radius R of the 6-dimensional sphere we sieve with.

Getting enough relations In order for the linear algebra step of the algorithm to succeed, one must collect enough relations to construct the linear system of equations. The smoothness bound B gives the number of relations that are needed for the linear algebra step to succeed, *i.e.*, the required number of relations N_{tot} corresponds to $2\pi(B)$, where $\pi(x)$ is the prime counting function. Indeed, the factor basis regroups prime ideals of $\mathcal{O}_1$ and $\mathcal{O}_2$ of norm less than B and of degree 1 and thus the size of the factor basis is twice the number of prime ideals of norm less than B. By Chebotarev density theorem, the latter is given

by the counting function π. A good approximation for the counting function is the logarithmic integral function.

Interestingly enough, two opposite considerations can be made. In theory, we would want slightly more relations in order to obtain a full-rank matrix. In practice, we require slightly less relations since some ideals will not be considered in the factorizations of the norms.

Now that we know an approximation of how many relations we need, one must balance the range of special-q's, the smoothness bound B and the radius R of the sieving space in order to optimize the total cost of the sieving step. Let $[q_{\min}, q_{\max}]$ denote the range of special-q to use for sieving.

In order to choose this interval in an optimal way, one must first estimate how many relations are generated on average from a single special-q of a given size. To do so, one can use the Dickman ρ function to estimate the probability of smoothness for the norms N_1 and N_2. This allows us to get an estimate of the number of relations expected from a special-q as a function of B. Using this estimation, one can then balance the parameters (q_{min}, R, B) in order to find the optimal combination. We defined the total cost (of relation collection) in this context to be the number of special-q required times the computation time for a single special-q. At this early stage, since we are looking at estimations, the time of sieving for a single special-q is taken to be R^6. The volume of a 6-sphere is given by $\frac{\pi^3}{\Gamma(4)} \cdot R^6$ and we ignore the constants at this stage. These parameters influence the total cost of sieving in the following way.

Impact of R: When choosing a large radius R, one expects to find more relations per special-q since the sieving space is larger. However, increasing R also increases the cost of enumeration for a given special-q. Overall for a fixed q_{min} and B, we observed in our experiments that when R increases, the total cost increases too. Indeed, when R increases the number of special-q needed decreases (which is both good and expected). However, this is counter-balanced by the increase in the cost of enumeration.

Impact of B: Similarly, increasing the smoothness bound B naturally increases the probability of being smooth. This results in more expected relations per special-q and thus a smaller interval would be required. On the other hand, the total number of relations required is equal to $2\pi(B)$ relations and thus the higher the value of B, the more relations are needed. This can be observed in our experiments where for increasing B, the total cost decreases and then starts increasing again.

Impact of q_{min}: The larger the special-q, the less relation it is expected to produce and thus we will require a larger interval. But if $q_{\min}$ is too small, one must take into account the effect of the special-q duplicates.

The behaviour of the parameters are summarized in Table 4 where $\nearrow$ means the parameters is being increased and $\searrow$ decreased.

parameters	# relations required	# special-qs	total cost
q_{min} ↗	–	↗	↗
R ↗	–	↘	↗
B ↗	↗	↘ then ↗	↘ then ↗

Table 4: Influence of increasing the size of parameters

In our computation, we chose the parameters

$$q_{min} = 5,000,113 \approx 2^{22.2}, \quad q_{\max} = 26,087,683 \approx 2^{24.6}, \quad B = 2^{27}, \quad R = 21,$$

$$p_{\max} = 10^7, \quad \mathcal{T}_{\text{sieve}} = 60, \quad p_{\text{batch}} = 2^{27}, \quad \mathcal{T}_{\text{batch}} = 0.$$

This results in a total of 1,280,000 special-q's subsets of polynomials to sieve on. Further optimizing these parameters with more extensive experiments is left for future work.

Sieving and enumeration. For each special-q, the first step in relation collection is to run a sieving algorithm to collect promising relations. This is done using Algorithm 4 which enumerates all vectors in $\mathcal{L}_{\mathfrak{Q},\mathfrak{p}} \cap S_6(21)$, for each prime ideal up to $p_{\max}$. In our implementation, we used this algorithm only on the f_2-side. On the f_1-side, the norms are much smaller, and the cost of enumerating is too high compared to the information it gives about the probability of being smooth.

In our code, enumeration is done with the program `enum.c` which takes as input a radius R, a cofactor bound $\mathcal{T}_{\text{siev}}$ and pre-computed files containing roots of the polynomials $h, f_1, f_2 \pmod p$ for primes $p \leq p_{\max}$ and a range of special-q with the associated reduced bases $M_{\mathfrak{Q}}$. It outputs the list of sieve-survivors. Let us fix a special-q $\mathfrak{Q}$. The enumeration algorithm must be run for each prime ideal $\mathfrak{p}$ in K_2 up to $p_{\max} = 10,000,000$ and promising candidates, *i.e.,* promising $(a(\iota), b(\iota))$-pairs, correspond to vectors for which the norms have a small cofactor.

In order to keep track of hits, we use a sieving table indexed by the vectors $\underline{\phi}$ of the coefficients of $a(\iota), b(\iota)$ converted to an unsigned 64-bit integer value. Whenever $\underline{\phi}$ is in $\mathcal{L}_{\mathfrak{Q},\mathfrak{p}} \cap S_6(21)$ for a given $\mathfrak{p}$, we add an approximation of $\log p$ to the sieving table at the corresponding index, where p is the prime below $\mathfrak{p}$. We will then only look at promising candidates which are the $(a(\iota), b(\iota))$-pairs for which $\sum \log p > 50$.

For each of these promising candidates, we compute approximations of the corresponding norms, meaning using floating numbers to accelerate the computation, and finally output the $(a(\iota), b(\iota))$-pairs for which the cofactor $\mathcal{C}_{\text{sieve}}(N_2(a(\iota) - b(\iota)\alpha_2)) \leq \mathcal{T}_{\text{sieve}} = 60$. Note that we also divide by the special-q. These sieve-survivors obtained at the end of enumeration as the output of `enum.c` are represented as

$$\begin{array}{l}\text{(rel1) } a_1(\iota)\ b_1(\iota)\ N_{1,f_2}\ N_{1,f_1}\\ \text{(rel2) } a_2(\iota)\ b_2(\iota)\ N_{2,f_2}\ N_{2,f_1}\\ \quad\ldots\end{array}$$

where $a_i(\iota), b_i(\iota)$ are encoded into 64-bit integer values. The formatting is chosen to match the pre-existing format in CADO-NFS in order to directly branch ourselves into CADO-NFS's code for the batch and ECM algorithms.

All in all, we collect approximately 76,401 million sieve-survivors. Note that these survivors are dealt with on-the-fly: in order to avoid storing all of them, they are removed just after the batch algorithm. Recall that at this stage of the algorithm, we also remove the K_h-unit duplicates and the ζ_2-duplicates.

Number of leaves and nodes in the enumeration trees. We analyze our enumeration algorithm by computing the expected number of leaves and nodes for a fixed special-q as the value of p increases. As seen in Figure 8, the output of our enumeration algorithm matches the expected values given by the formulae in Section 5. Both the amount of nodes and leaves decrease when p increases. The ratio r between the amount of leaves and nodes also decreases with p. We see that the estimation of the number of internal node is not precise. However, it gives a good idea of the general behavior of the algorithm. Furthermore r indeed remains high, which is a good indication that we do not spend too much computation for each divisibility information gathered by the process.

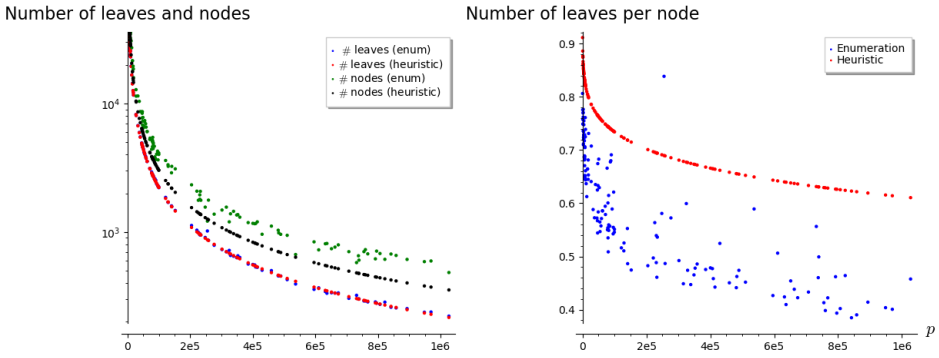

Fig. 8: Number of leaves and nodes (left) and number of leaves per node (right) as a function of p for a fixed 24-bit special-q. We see that as p increases, both the number of nodes visited by the enumeration algorithm and the number of leaves decreases, as expected. We compare the output of our code with the formulae given in Section 5 using the Gaussian heuristic.

Balancing sieving, batch and ECM. The sieve-survivors outputted by the enumeration algorithm are now the inputs to the batch algorithm implemented

in CADO-NFS [1]. Running batch and ECM is done sequentially in CADO-NFS. Here we choose not to run ECM as the computation is efficient enough for the record to finish in reasonable time. Further optimizing the parameters including the ECM algorithm is left for future work. We therefore select the batch-survivors with $p_{\text{batch}} = B = 2^{27}$ and $\mathcal{T}_{\text{batch}} = 0$. Indeed, we want the batch-survivors to correspond to our relations which is equivalent to saying that all the norms have no cofactor left, *i.e.,* they completely factor into primes up to the smoothness bound $p_{\text{batch}} = B$. Finally, `finishbatch` will output the final relations encoded as follows

$$a(\iota),\, b(\iota)\colon \sqcup, \sqcup, \sqcup, \cdots : \sqcup, \sqcup, \sqcup, \cdots$$

where $\sqcup$ represents the prime factors of N_2 (before the :) and N_1 (after the :) expressed in hexadecimal.
Removing all the possible duplicates further reduces the final amount of relations. This is described in Table 5.

The relation collection was run with an early version of our code, and took the equivalent of 25,300 core-hours. After the optimization given in Algorithm 4, it was going 10 % faster, and would have taken only 23,300 core-hours. On our sample computations, the relations were strictly identical to the one computed with the previous code, so we did not run again the whole computation.

	Sieve-survivors	Batch-survivors (removing K_h/ζ_2-dup.)	Removing special-q duplicates
# survivors	76 401M	18.69M	13.63M
% kept	0.013%	0.02 %	73 %

Table 5: Number of survivors after each step of the relation collection algorithm. The percentage is given with respect to the previous step. The percentage of sieve-survivors is taken with respect to $\text{vol}(S_6(21)) \times \#\text{special-}q\text{'s}$.

6.4 From a set of relations to a matrix

The goal is now to transform the factorization of the norms into a factorization of ideals in order to construct the matrix. Each relation will correspond to a row of this matrix, each column corresponds to an ideal, and the coefficients in the row indicate the power of this ideal that occurs in the ideal factorizations on each side, for this relation.

This step is easy in theory, and efficient algorithms are given, for instance, in Cohen's textbook [10]. Furthermore, these algorithms are implemented in computer algebra software like Pari or Magma. A version that works for non-tower extensions fields is also present in CADO-NFS.

With the objectives of keeping our toolchain simple for this computation, of avoiding external dependencies with the associated format conversions that are prone to error, and of having an easy integration with CADO-NFS, we decided

to sacrifice some relations and write a simple and robust tool for this step. This could and should be improved in a later phase of the development of our TNFS code. Nevertheless, we described it briefly, as an illustration of the simplifications that can be done in an early stage of experimentation with variants of NFS. Currently, our code takes only into account prime ideals for which we can apply Proposition 1, and more precisely those that can be put in the form:

$$\mathfrak{p} = \langle p, \phi_h(\iota), x - \rho(\iota) \rangle, \tag{3}$$

where p is a prime, ϕ_h is a monic irreducible factor of h modulo p, and $\rho(\iota)$ is a root of f_1 (resp. f_2) in $\mathcal{O}(\iota)/\langle p, \phi_h(\iota) \rangle$. This corresponds more or less to excluding the so-called "bad ideals" in CADO-NFS terminology, for which there is no nice mathematical definition. Furthermore, our code does not take properly into account multiplicities.

In concrete terms, we proceed as follows:

1. Build a database of all ideals of the form 3, for each side, up to the bound $B = 2^{27}$. This is actually a file similar to the `renumber` file in the classical CADO-NFS setting, that contains one ideal per line, with the side, p, $\phi_h(\iota)$ and $\rho(\iota)$.
2. For each relation, for each prime p that arises in the factorization of a norm, for each prime ideal $\mathfrak{p}$ above p in the database, check whether $\mathfrak{p}$ divides the relation.
3. If the sum of the degrees of the dividing ideals is equal to the multiplicity of p in the factorization of the norm, we deduce that they all have multiplicity one. If there is only one dividing ideal, we deduce its multiplicity from the multiplicity of p in the norm. Otherwise, we discard the relation.

Finally, we remark that due to the simple form of the ideals we consider, testing divisibility is easy. Let $(a(\iota), b(\iota))$ be a relation, and $\mathfrak{p} = \langle p, \phi_h(\iota), x - \rho(\iota) \rangle$ be an ideal. The question if whether $\mathfrak{p}$ divides the principal ideal of the element $a(\iota) - b(\iota)\alpha_1$ (resp. $a(\iota) - b(\iota)\alpha_2$, if we are considering the other side). From Proposition 1, this translates into checking the following equality:

$$a(\iota) \equiv \rho(\iota) b(\iota) \pmod{p}, \pmod{\phi_h(\iota)}.$$

Our code does not generically handle projective ideals, that are ideals above primes that divide the leading coefficient of f_1 and f_2. However, we made an ad-hoc treatment for f_2 and $p = 5$, since rejecting relations involving this prime would be too costly. As in the classical NFS, this is handled by considering the root 0 of $\tilde{f_2}$ where $\tilde{f_2}$ is the polynomial obtained by reversing the coefficients of f_2 and exchanging the roles of a and b.

In the end, we discard relations that involve ideals that are above p dividing the discriminant of the fields, and those for which there are non-trivial multiplicities. Fortunately, the amount of relations we throw away in this step only corresponds to a loss of 2% in our computation. The number of batch-survivors given in Table 5 is 18.69M, and removing the relations for which we are not able to obtain the ideal factorization we finally have 18.25M batch-survivors.

6.5 Linear algebra

Before starting the linear algebra step, the matrix must undergo a few modifications in order to speed up the resolution of the system. This is done by a step called filtering. More precisely, the aim of filtering is to reduce the size of the matrix of relations without modifying its kernel.

Dealing with special-q duplicates. As mentioned in Section 3.4, only ζ_2-duplicates and K_h-unit duplicates can be dealt with prior to constructing the matrix. To remove special-q duplicates, we compare the ideal factorizations of each relation and remove identical lines in the file containing all our relations. Before eliminating special-q duplicates we had 18.25M batch-survivors. Removing these duplicates decreased the amount of survivors to 13.63M, a loss of 27%.

Filtering. The matrix of relations is now ready to be sent to CADO-NFS's filter. We have 15.21M ideals in the factor basis, and thus the input matrix to CADO-NFS's filter is a matrix of size 13.63M $\times$ 15.21M. Note that not all the ideals intervene in the relations. The goal of filtering is to reduce the size of the matrix and make it square. Filtering in CADO-NFS uses two steps: purge and merge.

The purge step consists in removing columns that only contain zero coefficients. Indeed only 87% of the ideals of the factor basis appear in relations. The rest leading to zero-columns are deleted. Besides, the purge step removes columns (and corresponding lines) that contain a unique element. These columns correspond to prime ideals, called *singletons*, that occur only once in all the relations. We start with 13.63M lines in the matrix. After removing the singletons, we are left with only 5.21M lines. Hence, purge reduces the number of lines in our matrix by approximately 62%. Thus even if the purge step and more generally filtering is not present in the complexity analysis of TNFS, it is of significant importance in practice for the feasibility of the linear algebra step.

The next step of filtering is merge, which corresponds to a structured Gaussian elimination. It aims at further reducing the matrix size by performing linear combinations of the rows of the input matrix. In our computation, the merge takes as input a square matrix of dimension 5.21M and, after Gaussian elimination up to a density of 100 coefficients per line, its size is decreased to 1.73M. If we eliminate up to 150 coefficients per line, the size is decreased to 1.51M.

Finally, after filtering, we have 1.51M relations and a $(1.51\text{M} + 7) \times 1.51\text{M}$ dimension matrix. The entire filtering step removes 89% of the relations (or 92% if we count before the removal of the duplicates). The 7 extra columns come from the Schirokauer maps, as we see now.

Schirokauer maps. In order to easily use CADO-NFS implementation of Schirokauer maps, we propose a rather simple trick: represent the tower of number fields as an absolute (non-tower) extension field. More precisely, we follow the general theory of virtual logarithms [23, 37] (see also Section 4.2.3 of [11]) and recall that a Schirokauer map is any surjective morphism from $K_i^*/(K_i^*)^\ell \to (\mathbb{Z}/\ell\mathbb{Z})^{r_i}$ where K_i is a number field and r_i its unit rank. In the

classical NFS setup, K_i is simply an extension of $\mathbb{Q}$, whereas in the Tower setup, K_i is an extension of $\mathbb{Q}(\iota)$.

It is then possible to define a Schirokauer map in TNFS by first defining an isomorphism from the intermediate fields $K_i = \mathbb{Q}(\iota, \alpha_i)$ to a number field K_{F_i} of degree $\deg h \times \deg f_i$ and then using a classical Schirokauer map $\Lambda_{\text{classical}}$ from the latter to $(\mathbb{Z}/\ell\mathbb{Z})^{r_i}$. In other words, we define the map Λ_i as

$$\Lambda_i : K_i^*/(K_i)^{*\ell} \xrightarrow{\simeq} K_{F_i}^*/(K_{F_i})^{*\ell} \to (\mathbb{Z}/\ell\mathbb{Z})^{r_i}.$$

A polynomial F_i and the corresponding isomorphism

$$\Phi_i : \ K_i \to K_{F_i}$$

are easy to find, and can be represented by the images of each base elements. Thereafter, the map from K_i to K_{F_i} is seen as a linear map and applying it to an element is essentially free.

There are as many Schirokauer maps as the rank of units in K_i. For our computation, this means 2 on the f_1-side and 5 on the f_2-side. Computing Schirokauer maps, *i.e.,* filling seven columns, takes 40 core-hours.

Remark 8. In general, and also true in our computation, the values $\Phi_i(1), \Phi_i(\iota), \cdots$ have denominators. As Schirokauer maps, as implemented in CADO-NFS, only require integers, we directly multiply the coefficients by the least common multiplier of these denominators. This denominator-clearing can be made completely transparent by choosing Schirokauer maps Λ_i that evaluate to zero over $\mathbb{Q}$. This is the `legacy` mode in CADO-NFS's implementation of Schirokauer maps.

Solving the system. Solving the linear system with 1.51M rows and columns, including the 7 dense columns of Schirokauer maps is done with the block-Wiedemann algorithm, as implemented in CADO-NFS. We used the default behavior which is to run 7 sequences, each one taking one of the heavy columns as input (see [25]). Therefore, the large number of Schirokauer maps does not induce an increase in the running time of this step.

All the sequences can be run in parallel, for a total cost of 1,210 core hours. The reconstruction of the linear generator and the final sum-up leading to the kernel vector must be added up, and the overall cost of linear algebra is 1,403 core hours. We emphasize that the most expensive steps of the linear algebra part are the computation of the sequences and the solution step where the sparse matrix-vector multiplication is the most costly operation.

Un-filtering. The kernel vector gives the virtual logarithms of 1.51M ideals belonging to the factor basis. Using the relations that were deleted during the purge and merge process, many more can be deduced. This can be seen as reverting the filtering, where each time a relation is re-added, we check if it involves a (unique) ideal for which the virtual logarithm is not yet known. If so, we can deduce it. After this process, we know the virtual logarithms of 12M ideals, corresponding to 79.4% of the factor bases elements.

6.6 Descent step and discrete logarithm of the target

Now that we know the virtual logarithms of (a large proportion of) the factor basis elements, we are ready for the descent step. We choose as generator the element $g = x + \iota$. This element g lifted in the field defined by f_1 is a unit (of infinite order). This allows us to easily compute its virtual logarithm as it is found using the virtual logarithms outputted by the linear algebra step and an additional Schirokauer map computation.

Remark 9. Note that the virtual logarithm of the generator is computed in an arbitrary basis, say γ, the same basis for which we have the virtual logarithms of the elements of the factor basis outputted by the linear algbebra step. In order to get the discrete logarithm of our target element in base g, we first compute the discrete logarithm of the target in base γ. Because we know the discrete logarithm of g in base γ we can finally obtain the discrete logarithm of our target in base g.

As mentioned in Section 2.2, we use Guillevic's algorithm [19] to optimize the initial splitting step. The descent starts by a smoothing step which required 45 core hours to generate 64M candidates and 10 core hours to identify an element $s \in \mathbb{F}_{p^6}^*$ such that its lift to K_1 has a 35-bit smooth norm. The factors of s greater than 27-bit for which we do not have the virtual logarithms yet are descended in a single special-q step. This descent is done with the same strategy as for the relation collection, namely enumeration and batch, but using a larger radius $R = 33$. Because of the small amount of factors concerned, the time is negligible. The descent step takes 55 core hours. The overall time in core hours of the computation is reported in Table 6.

Relation Collection	Linear algebra	Schirokauer maps	Descent	Overall time
23,300	1,403	40	55	24,798

Table 6: Overall time of our record computation in core hours.

We finally find the discrete logarithm of our target element:

$$\log(\texttt{target}) = 76272808168753222977667479701383785303538529763154 98.$$

To confirm the validity of our computation, we verify that $g^{\log(\texttt{target})} = \texttt{target}$ is true, modulo ℓ-th powers, as we computed the discrete logarithm only modulo ℓ.

7 Towards an efficient use of automorphisms in TNFS

7.1 Automorphisms present in our 521 example

In our 521-bit example, the f_1 and f_2 polynomials are obtained by the Conjugation method, so that both are palindromic polynomials. Their coefficients are the

same if one reads them in reverse order. As a consequence, there is an explicit automorphism of order 2 of the form $\sigma : x \mapsto 1/x$ in the fields K_1 and K_2.

From Section 3, a degree 1 ideal $\mathfrak{p}$ of norm p in $\mathcal{O}_1$ of interest in the relation collection has the form $\mathfrak{p} = \langle p, \iota - r_\iota, x - r_x \rangle$, where r_ι is a root of h modulo p and r_x is a root of f_1 modulo p. Its conjugate $\mathfrak{p}^\sigma$ has the same form, where r_x is replaced by $1/r_x$ modulo p, which is also a root of f_1, due to its palindromic form. This generalizes to any ideal of the form given by Equality (3).

Let $\phi(x, \iota) = a(\iota) - b(\iota)x$ be a polynomial that gives a relation, so that, when mapped to K_1, its ideal-factorization contains only prime ideals below the smoothness bound. Taking the conjugate $\phi^\sigma(x, \iota) = b(\iota) - a(\iota)x$, its mapping to K_1 also has an ideal-factorization containing only prime ideals below the smoothness bound, since each ideal will be replaced by its conjugate. This is the same on both sides, and therefore, each relation gives another relation for free.

This property is well-known and when the automorphism is of order 2, it leads to a factor-2 saving in the timing of the relation collection phase, for instance by using only one special-q in each σ-orbit.

In order to also save on the linear algebra computation time, one needs to be able to replace each pair of columns representing a pair of conjugate ideals $(\mathfrak{p}, \mathfrak{p}^\sigma)$ of the factor basis, by a single column. It is tempting to hope that the virtual logarithms of two conjugate ideals take opposite values modulo ℓ. Then the two columns could be replaced by the difference between them. The matrix size would be reduced by a factor of 2, leading to a factor-4 saving.

Unfortunately, this is not true in general, and it is actually wrong in our 521-bit computation. Thus, a naive use of automorphisms would save a factor 2 in the relation collection, but would not help the linear algebra phase. In the rest of this section, we propose a construction to compute one virtual logarithm from the virtual logarithm of the conjugate ideal, and thus improve the linear algebra too.

7.2 Galois-compatible Schirokauer maps (GCSM)

We start by giving a precise definition of the kind of automorphisms that can be useful in the TNFS context.

Definition 4. *Let us fix a TNFS-diagram as in Figure 2. A TNFS-automorphism of order r is a set of 3 automorphisms of order r:*

- *an automorphism of K_1 over $\mathbb{Q}$;*
- *an automorphism of K_2 over $\mathbb{Q}$;*
- *an automorphism of $\mathbb{F}_{p^n}$ over $\mathbb{F}_p$;*

such that all of them can be expressed as $z \mapsto \tau(z)$, where τ is the same rational fraction, in the coordinate systems of the diagram. We abuse notations and write $\sigma : z \mapsto \tau(z)$ for all these automorphisms.

In all generality, the element z is a bivariate expression in x and ι. Since τ represent field automorphisms, it is enough to write its image on x and y

separately. Note that when we wrote $\sigma : x \mapsto 1/x$ in the previous subsection, we left implicit the fact that it should be extended to $\sigma : \iota \mapsto \iota$, leaving the second variable unchanged.

Definition 5 (Galois-compatible Schirokauer map). *Let us fix a TNFS-diagram as in Figure 2 and let σ be a TNFS-automorphism. Let ℓ be the prime dividing $\Phi_n(p)$ modulo which we want to compute discrete logarithms. A Galois-compatible Schirokauer map (GCSM) of K_i (for a side $i = 1$ or 2) is a Schirokauer map $\Lambda : K_i^*/(K_i^*)^\ell \to (\mathbb{Z}/\ell\mathbb{Z})^{r_i}$ where r_i is the unit rank of K_i, such that $\Lambda(z) = 0$ for all the elements z that are fixed by σ.*

This definition is sufficient to obtain a relation between a virtual logarithm of an ideal and the virtual logarithm of its conjugate, as we now explain. This fixes what is missing in our 521-bit example.

Theorem 1. *In a TNFS-diagram context, let σ be a TNFS-automorphism of order 2. Let ℓ be the prime dividing $\Phi_n(p)$ modulo which we want to compute discrete logarithms, and let Λ be a GCSM of K_i for this ℓ. Then the associated virtual logarithms* vlog *are such that for all prime ideals $\mathfrak{p}$ of $\mathcal{O}_i$,*

$$\operatorname{vlog} \mathfrak{p} \equiv -\operatorname{vlog} \mathfrak{p}^\sigma \mod \ell.$$

Proof. Let cl_i be the class number of K_i. For any prime ideal $\mathfrak{p}$, we can consider $\gamma_\mathfrak{p} \in K_f$ a generator of the principal ideal $\mathfrak{p}^{\mathrm{cl}_i}$. The element $\gamma_\mathfrak{p}$ is defined up to a unit, and, by surjectivity of Λ, we can choose it so that $\Lambda(\gamma_\mathfrak{p}) = 0$. Then, by definition, the virtual logarithm $\operatorname{vlog} \mathfrak{p}$ is equal to the (classical) discrete logarithm of $\bar{\gamma_\mathfrak{p}}$, the image of $\gamma_\mathfrak{p}$ in the target finite field $\mathbb{F}_{p^n}$.

Let us now evaluate $\operatorname{vlog} \mathfrak{p}^\sigma$. First we have $(\mathfrak{p}^\sigma)^{\mathrm{cl}_i} = (\gamma_\mathfrak{p}^\sigma)$. If we can show that $\Lambda(\gamma_\mathfrak{p}^\sigma) = 0$ then $\gamma_\mathfrak{p}^\sigma$ will be a generator of $(\mathfrak{p}^\sigma)^{\mathrm{cl}_i}$ that is valid to define $\operatorname{vlog} \mathfrak{p}^\sigma$ as equal to $\log(\overline{\gamma_\mathfrak{p}^\sigma})$ modulo ℓ.

By the morphism property of the Schirokauer maps, $\Lambda(\gamma_\mathfrak{p}\ \gamma_\mathfrak{p}^\sigma) = \Lambda(\gamma_\mathfrak{p}) + \Lambda(\gamma_\mathfrak{p}^\sigma)$. In this equation, we already know that $\Lambda(\gamma_\mathfrak{p})$ is zero. In addition, since $\gamma_\mathfrak{p}\ \gamma_\mathfrak{p}^\sigma$ belongs to the subfield of K_i of all the elements fixed by the automorphism, by our hypothesis on Λ, we have that $\Lambda(\gamma_\mathfrak{p}\ \gamma_\mathfrak{p}^\sigma) = 0$. Therefore, the last term of the equation is also zero, namely $\Lambda(\gamma_\mathfrak{p}^\sigma) = 0$.

Hence $\operatorname{vlog} \mathfrak{p}^\sigma = \log(\overline{\gamma_\mathfrak{p}^\sigma}) = \log(\bar{\gamma_\mathfrak{p}}^\sigma) = -\log(\bar{\gamma_\mathfrak{p}}) = -\operatorname{vlog} \mathfrak{p} \mod \ell$.

In the conditions of this theorem, we can use the strategy of merging pairs of columns in the matrix. This leads to the following result.

Corollary 1. *In the TNFS algorithm, if the diagram is such that there exists a TNFS-automorphism of order 2, together with Galois-compatible Schirokauer maps on both sides, then a factor 4 can be saved in the complexity of the linear algebra step.*

7.3 Constructions of GCSM

We place ourselves in the setting of Definition 5 and consider the field K_1. Let L_1 be the subfield of K_1 of elements fixed by the automorphism σ. Since σ has order 2, then L_1 is of index 2 in K_1. We make this extension explicit, and write $K_1 = L_1(w)$, so that any element z of K_1 can be uniquely written $z = z_0 + z_1 w$, with z_0 and z_1 in L_1.

The traditional construction of Schirokauer maps is based on the map $z \mapsto z' = (z^\varepsilon - 1)/\ell \mod \ell$, where ε is the LCM of all the orders of the multiplicative groups of the residue fields of K_1 modulo ℓ. We follow this strategy, and subsequently write z' as $z'_0 + z'_1 w$. The coefficients of z'_1 in its polynomial representation are then candidates for being Galois-compatible Schirokauer maps. Indeed, if z is in the subfield L_1, then z' is also in L_1, and therefore z'_1 is equal to 0. Therefore, we get the expected property.

The number of coefficients in the representation of z'_1 is the degree of L_1, that is $\frac{1}{2}(\deg h)(\deg f_1)$. If this number is not smaller than the unit rank r_1 of K_1, then we can pick r_1 such coefficients, and they form a GCSM. We will discuss later (see Section 7.4) a condition under which this is the case.

The case of the 521-bit example. In the 521-bit computation, we used $h(\iota) = \iota^3 - \iota + 1$, $f_1(x) = x^4 + 1$, and $f_2(x)$ a palindromic polynomial of degree 2. The TNFS-automorphism is given by $\sigma : x \mapsto 1/x$. For K_2, the subfield L_2 is $\mathbb{Q}(\iota)$. The expression of z' as $z'_0 + z'_1 w$ is then the natural expression coming from the TNFS diagram. The element z'_1 has 3 coefficients and the unit rank is 2, so there are enough choices to define a GCSM on the K_2 side.

On the K_1 side, this is slightly more complicated. The subfield L_1 fixed by σ is the quadratic extension of $\mathbb{Q}(\iota)$ defined by $\xi = x + 1/x$. One can directly check that $\xi^2 = 2$, so that $L_1 = \mathbb{Q}(\iota)(\sqrt{2})$. In the construction, the element z' has the form

$$z' = c_0 + c_1 x + c_2 x^2 + c_3 x^3,$$

where the c_i's are elements of $\mathbb{Z}[\iota]/\ell\mathbb{Z}[\iota]$. In order to control what happens over L_1, we apply a basis change and use $\langle 1, \xi, x, x\xi \rangle$ instead of $\langle 1, x, x^2, x^3 \rangle$, as a basis of K_1 over $\mathbb{Q}(\iota)$. We obtain the following rewriting of z':

$$z' = (c_0 - c_2) + (-c_3)\xi + (c_1 + c_3)x + (c_2)x\xi.$$

In order to define Λ that is zero on L_1, it suffices to pick coefficients of $(c_1 + c_3)$ and c_2. This gives $2 \deg h = 6$ coefficients among which to choose, which is enough, since the unit rank is 5.

We discovered this GCSM strategy too late to apply it during our record computation. However, we tested it on a small example with the same polynomials h and f_1. We checked again that with the traditional choice of Schirokauer maps, the property $\operatorname{vlog} \mathfrak{p} \equiv -\operatorname{vlog} \mathfrak{p}^\sigma \mod \ell$ did not hold. However, our GCSM construction allowed to enforce it, as predicted by the theory. Since this adds just an easy linear transform, the additional cost of the GCSM computation is completely negligible.

An example for $\mathbb{F}_{p^{12}}$ with $\deg h = 4$. The approach taken in the previous example is to have the order 2 automorphism σ come from f_1 and f_2 being palindromic. In the case of an $\mathbb{F}_{p^{12}}$ TNFS-computation with $\deg h = 4$, this would not work. Indeed, in the TNFS-diagram, the field $\mathbb{Q}(\iota)$ is mapped to $\mathbb{F}_{p^4}$ (because h is irreducible modulo p). Hence, the common irreducible factor φ of f_1 and f_2 modulo p has to be of degree 3. As an automorphism of K_1 of order 2, if σ fixes $\mathbb{Q}(\iota)$, then it will be mapped to an automorphism in $\mathbb{F}_{p^{12}}$ fixing the subfield $\mathbb{F}_{p^4}$. However, since the extension $\mathbb{F}_{p^{12}}/\mathbb{F}_{p^4}$ is cyclic of degree 3, it cannot have an automorphism of order 2. As a consequence, in this context, any TNFS-automorphism of order 2 must come from h and not from f_1 and f_2.

We propose to study the case where the construction is based on the following polynomial defining $\mathbb{Q}(\iota)$: $h(\iota) = \iota^4 - \iota^3 + \iota^2 - \iota + 1$, which has $\sigma : \iota \mapsto 1/\iota$ as order-2 automorphism. We leave the polynomials f_1 and f_2 undefined, since their shape do not affect the discussion. We only assume that their coefficients are in $\mathbb{Z}$ which is possible because $\deg \varphi = 3$ is coprime to $\deg h = 4$.

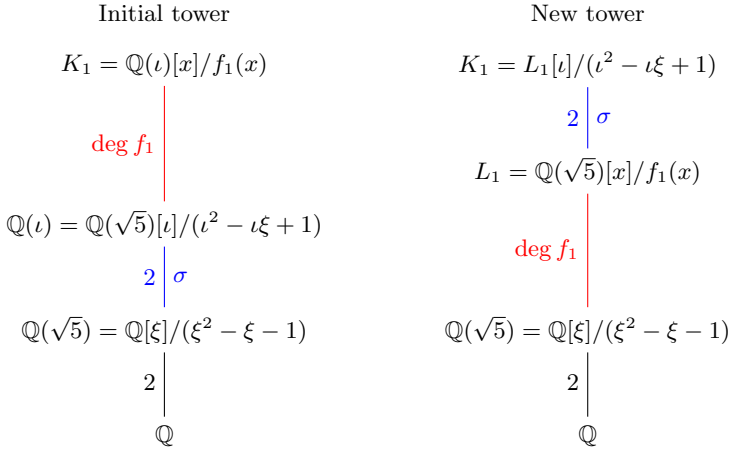

Fig. 9: Change of field representation in order to define a Galois-compatible Schirokauer map.

Let us make the field extension with L_1 explicit. For this, we consider the field $\mathbb{Q}(\iota)^+$, the subfield of $\mathbb{Q}(\iota)$ fixed by σ. Let $\xi = y + y^\sigma$. We have $\xi = -y^3 + y^2 + 1$, and its minimal polynomial is $h^+ = \xi^2 - \xi - 1$. One can also check that $\mathbb{Q}(\iota) = \mathbb{Q}(\iota)^+[\iota]/(\iota^2 - \iota\xi + 1)$, and that $\mathbb{Q}(\iota)^+$ is isomorphic to $\mathbb{Q}(\sqrt{5})$. The subfield L_1 of K_1 fixed by σ is then

$$L_1 = \mathbb{Q}(\sqrt{5})[x]/f(x).$$

In order to compute a GCSM on the K_1 side, we can then apply to z' the linear transformation corresponding to the field isomorphism that maps the original tower of extensions defining K_1 to the one that makes L_1 visible, as in Figure 9.

We can then easily take the coordinates that are always equal to 0 for elements that belong to L_1. The same can be done for K_2. The only remaining question is whether there are enough coordinates compared to the unit rank.

7.4 Condition on the number of real roots

We have seen that the Conjugation method used for our 521-bit computation allows the construction of GCSM. However, in general, the unit rank might be too large. We now make this obstruction more explicit.

Let f be one of the polynomial f_1 or f_2. Our construction produces a set of $\frac{1}{2}(\deg h)(\deg f)$ values from which to pick, in order to build a Galois-compatible Schirokauer map. From Dirichlet's unit theorem, this will be enough only if the number of real embeddings in K_f is sufficiently low.

More precisely, if r_{1f} is the number of real roots of f and r_{1h} is the number of real roots of h, then the number of real embeddings of K_f is $r_{1f}r_{1h}$. The unit rank of K_f is then

$$r = \frac{1}{2}\Big((\deg h)(\deg f) + r_{1f}r_{1h}\Big) - 1,$$

and this will not exceed the number of available coefficients only when the following condition is satisfied:

$$\big(\text{number of real roots of } f\big) \times \big(\text{number of real roots of } h\big) \leq 2.$$

Since this condition must be satisfied on both sides, it is much easier to reach if h is chosen to have zero or one real root (depending on the parity of its degree). In any case, this condition on the number of real roots is in contradiction with the common practice of favoring polynomials with many real roots in order to have smaller norms. Still, the benefit of using automorphisms should more than balance this penalty.

Concluding remarks

We recall the data from Table 1 in the introduction: the time for collecting relations in our 521-bit computation is only 23,300 core-hours, much less that the 69,120 core-hours of McGuire and Robinson for a 423-bit computation, also in a field of the form $\mathbb{F}_{p^6}$; and this was already much faster than Grémy's work.

This huge improvement is mostly due to the fact that our efficient sieving technique allows to work in large dimensions, and therefore enables the use of Tower NFS. While asymptotic complexities can hardly lead to definitive statements about the performance of TNFS for such "small" target finite fields, the norms that it produces are indeed quite small.

In an attempt to quantify this smallness, we compare them to the norms obtained for equivalent target sizes with the classical NFS algorithm for factoring and DLP in prime fields. For those, CADO-NFS can serve as a reference, since

the provided parameters are reasonably well optimized (we use 512-bit targets for the comparison instead of our 521-bit size, since 512-bit is a standard size for CADO-NFS). The result of this comparison is that, while in our computation we encountered norms with product of size around 250 bits, the equivalent for a 512-bit factorisation is around 280 bits, and for a 512-bit prime field DLP with Joux-Lercier polynomial selection, it is around 270 bits. We therefore notice that even if $\mathbb{F}_{p^6}$ is not a high degree extension, and even if a 521-bit size is still far from a secure cryptographic size, the "tower" effect is already pretty impactful.

Furthermore, we would like to emphasize that our experiment was merely a first demonstration. First, considering the total computation time needed: this record used 1400 times less core years than for current discrete logarithm computations in prime field. Indeed, we used roughly 2.8 core years to perform our computation while the last discrete logarithm record [7] in a 795-bit prime field took 4000 core years. There is also still much room for improvement in the tuning of the various parameters and the use of the explicit Galois action that is available with the Conjugation method. These will be necessary when working with Tower NFS on larger size finite fields.

Acknowledgements. We are indebted to Léo Ducas and Wessel van Woerden for insightful discussions about the lattice points enumeration aspect of this work. Many thanks to Aurore Guillevic, for numerous discussions, in particular about polynomial selection and the blockchain ecosystem. Experiments in this paper were carried out using the Grid'5000 testbed, supported by a scientific interest group hosted by Inria and including CNRS, RENATER and several Universities as well as other organizations (see `https://www.grid5000.fr`).

References

1. The CADO-NFS Development Team. CADO-NFS, An Implementation of the Number Field Sieve Algorithm. Found at https://gitlab.inria.fr/cado-nfs/cado-nfs, development version of January 2021
2. Barbulescu, R., Gaudry, P., Guillevic, A., Morain, F.: Improving NFS for the discrete logarithm problem in non-prime finite fields. In: Oswald, E., Fischlin, M. (eds.) EUROCRYPT 2015, Part I. LNCS, vol. 9056, pp. 129–155. Springer, Heidelberg (Apr 2015).
3. Barbulescu, R., Gaudry, P., Joux, A., Thomé, E.: A heuristic quasi-polynomial algorithm for discrete logarithm in finite fields of small characteristic. In: Nguyen, P.Q., Oswald, E. (eds.) EUROCRYPT 2014. LNCS, vol. 8441, pp. 1–16. Springer, Heidelberg (May 2014).
4. Barbulescu, R., Gaudry, P., Kleinjung, T.: The tower number field sieve. In: Iwata, T., Cheon, J.H. (eds.) ASIACRYPT 2015, Part II. LNCS, vol. 9453, pp. 31–55. Springer, Heidelberg (Nov / Dec 2015).
5. Ben-Sasson, E., Chiesa, A., Tromer, E., Virza, M.: Scalable zero knowledge via cycles of elliptic curves. In: Garay, J.A., Gennaro, R. (eds.) CRYPTO 2014, Part II. LNCS, vol. 8617, pp. 276–294. Springer, Heidelberg (Aug 2014).
6. Bernstein, D.J.: How to find smooth parts of integers (2004), `http://cr.yp.to/factorization/smoothparts-20040510.pdf`

7. Boudot, F., Gaudry, P., Guillevic, A., Heninger, N., Thomé, E., Zimmermann, P.: Comparing the difficulty of factorization and discrete logarithm: A 240-digit experiment. In: Micciancio, D., Ristenpart, T. (eds.) CRYPTO 2020, Part II. LNCS, vol. 12171, pp. 62–91. Springer, Heidelberg (Aug 2020).
8. Bouvier, C., Imbert, L.: Faster cofactorization with ECM using mixed representations. In: Kiayias, A., Kohlweiss, M., Wallden, P., Zikas, V. (eds.) PKC 2020, Part II. LNCS, vol. 12111, pp. 483–504. Springer, Heidelberg (May 2020).
9. CODA: MNT-6 curve with parameter 753 for Snark prover. Webpage at https://coinlist.co/build/coda/pages/MNT6753
10. Cohen, H.: Advanced Topics in Computational Number Theory. Graduate Texts in Mathematics, Springer New York (2012), `https://books.google.sc/books?id=OFjdBwAAQBAJ`
11. De Micheli, G.: Discrete Logarithm Cryptanalyses: Number Field Sieve and Lattice Tools for Side-Channel Attacks. Ph.D. thesis, Université de Lorraine (2021)
12. Fincke, U., Pohst, M.: Improved methods for calculating vectors of short length in a lattice, including a complexity analysis. Mathematics of Computation **44**, 463–471 (1985)
13. Franke, J., Kleinjung, T.: Continued Fractions and Lattice Sieving. Special-Purpose Hardware for Attacking Cryptographic Systems–SHARCS p. 40 (2005)
14. Gama, N., Nguyen, P.Q., Regev, O.: Lattice enumeration using extreme pruning. In: Gilbert, H. (ed.) EUROCRYPT 2010. LNCS, vol. 6110, pp. 257–278. Springer, Heidelberg (May / Jun 2010).
15. Granger, R., Kleinjung, T., Zumbrägel, J.: On the discrete logarithm problem in finite fields of fixed characteristic. Transactions of the American Mathematical Society **370**(5), 3129–3145 (2018)
16. Grémy, L.: Higher dimensional sieving for the number field sieve algorithms. In: ANTS 2018 - Thirteenth Algorithmic Number Theory Symposium. pp. 1–16 (Jul 2018)
17. Grémy, L., Guillevic, A., Morain, F., Thomé, E.: Computing discrete logarithms in $\mathbb{F}_{p^6}$. In: Adams, C., Camenisch, J. (eds.) SAC 2017. LNCS, vol. 10719, pp. 85–105. Springer, Heidelberg (Aug 2017).
18. Guillevic, A.: Pairing-friendly curves. Blogpost found at https://members.loria.fr/AGuillevic/pairing-friendly-curves
19. Guillevic, A.: Faster individual discrete logarithms in finite fields of composite extension degree. Mathematics of Computation **88**(317), 1273–1301 (Jan 2019).
20. Guillevic, A., Singh, S.: On the alpha value of polynomials in the Tower Number Field Sieve Algorithm. Mathematical Cryptology **1**(1) (Feb 2021)
21. Hanrot, G., Stehlé, D.: Improved analysis of kannan's shortest lattice vector algorithm. In: Menezes, A. (ed.) CRYPTO 2007. LNCS, vol. 4622, pp. 170–186. Springer, Heidelberg (Aug 2007).
22. Hayasaka, K., Aoki, K., Kobayashi, T., Takagi, T.: An experiment of number field sieve for discrete logarithm problem over GF(p^n). JSIAM Lett. **6**, 53–56 (2014).
23. Joux, A., Lercier, R.: Improvements to the General Number Field Sieve for discrete logarithms in prime fields. Mathematics of Computation pp. 953–967 (2003)
24. Joux, A., Lercier, R., Smart, N., Vercauteren, F.: The number field sieve in the medium prime case. In: Dwork, C. (ed.) CRYPTO 2006. LNCS, vol. 4117, pp. 326–344. Springer, Heidelberg (Aug 2006).
25. Joux, A., Pierrot, C.: Nearly Sparse Linear Algebra and application to Discrete Logarithms Computations. In: Contemporary Developments in Finite Fields and Applications (2016).

26. Kannan, R.: Improved Algorithms for Integer Programming and Related Lattice Problems. In: Proceedings of the Fifteenth Annual ACM Symposium on Theory of Computing. pp. 193–206. STOC'83, Association for Computing Machinery, New York, NY, USA (1983)
27. Kim, T., Barbulescu, R.: Extended tower number field sieve: A new complexity for the medium prime case. In: Robshaw, M., Katz, J. (eds.) CRYPTO 2016, Part I. LNCS, vol. 9814, pp. 543–571. Springer, Heidelberg (Aug 2016).
28. Kim, T., Jeong, J.: Extended tower number field sieve with application to finite fields of arbitrary composite extension degree. In: Fehr, S. (ed.) PKC 2017, Part I. LNCS, vol. 10174, pp. 388–408. Springer, Heidelberg (Mar 2017).
29. Kleinjung, T., Wesolowski, B.: Discrete logarithms in quasi-polynomial time in finite fields of fixed characteristic (2019), `https://eprint.iacr.org/2019/751`, cryptology ePrint Archive, Report 2019/751, to appear in Journal of the AMS
30. Lenstra, H.W.: Factoring Integers with Elliptic Curves. Annals of Mathematics **126**(3), 649–673 (1987)
31. McGuire, G., Robinson, O.: Lattice Sieving in Three Dimensions for Discrete Log in Medium Characteristic. Journal of Mathematical Cryptology **15**(1), 223 – 236 (01 Jan 2021).
32. Miyaji, A., Nakabayashi, M., Nonmembers, S.: New explicit conditions of elliptic curve traces for fr-reduction. IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences **84** (09 2001)
33. Nguyen, P.Q., Stehlé, D.: LLL on the average. In: Proceedings of the 7th International Conference on Algorithmic Number Theory. ANTS06, Springer-Verlag, Berlin, Heidelberg (2006)
34. Pollard, J.M.: The Lattice Sieve. In: Lenstra, A.K., Lenstra, H.W. (eds.) The development of the number field sieve. pp. 43–49. Springer Berlin Heidelberg, Berlin, Heidelberg (1993)
35. Sarkar, P., Singh, S.: New complexity trade-offs for the (multiple) number field sieve algorithm in non-prime fields. In: Fischlin, M., Coron, J.S. (eds.) EUROCRYPT 2016, Part I. LNCS, vol. 9665, pp. 429–458. Springer, Heidelberg (May 2016).
36. Schirokauer, O.: Using Number Fields to Compute Logarithms in Finite Fields. Mathematics of Computation **69**, 1267–1283 (2000)
37. Schirokauer, O.: Virtual logarithms. Journal of Algorithms **57**, 140–147 (2005)
38. Schnorr, C.P., Euchner, M.: Lattice Basis Reduction: Improved Practical Algorithms and Solving Subset Sum Problems. Mathematical Programming **66**(2) (1994)
39. Wiedemann, D.H.: Solving Sparse Linear Equations over Finite Fields. IEEE Transactions on Information Theory **32**(1), 54–62 (1986).