Research Article

Gary McGuire* and Oisín Robinson

Lattice Sieving in Three Dimensions for Discrete Log in Medium Characteristic

https://doi.org/10.1515/jmc-2020-0008
Received Feb 25, 2020; accepted Sep 01, 2020

Abstract: Lattice sieving in two dimensions has proven to be an indispensable practical aid in integer factorization and discrete log computations involving the number field sieve. The main contribution of this article is to show that a different method of lattice sieving in three dimensions will provide a significant speedup in medium characteristic. Our method is to use the successive minima and shortest vectors of the lattice instead of transition vectors to iterate through lattice points. We showcase the new method by a record computation in a 133-bit subgroup of $\mathbb{F}_{p^6}$, with p^6 having 423 bits. Our overall timing is nearly 3 times faster than the previous record of a 132-bit subgroup in a 422-bit field. The approach generalizes to dimensions 4 or more, overcoming one key obstruction to the implementation of the tower number field sieve.

Keywords: Number field sieve, discrete log

2020 Mathematics Subject Classification: 11Y05

1 Introduction

The most widely adopted public-key cryptography algorithms in current use are critically dependent on the (assumed) intractability of either the integer factorization problem (IFP), the finite field discrete logarithm problem (DLP) or the elliptic curve discrete logarithm problem (ECDLP). The most effective known attacks against IFP and DLP use the same basic algorithm, namely the Number Field Sieve (NFS). This algorithm has subexponential complexity in the input size. On the other hand, all known methods to attack the ECDLP in the general case have exponential complexity. However there are special instances of the ECDLP which can be attacked by effectively transferring the problem to a finite field, allowing the NFS to be used. For example such instances arise in the context of pairing-based cryptography, where certain elliptic curves can be used to realize 'Identity-Based Encryption' (IBE). There is a trade-off between the reduced security due to the size of the finite field on which the security is dependent, and increased efficiency of the pairing arithmetic. The optimal parameters have been the subject of intense scrutiny over the last few years, which have seen a succession of improvements in the NFS for the DLP in the medium characteristic case. This is directly relevant in the case of pairings, where the finite field on which the security of the protocol depends is typically a small degree extension of a prime field.

A key part of the NFS is lattice sieving. The main contribution of this article is to demonstrate that different methods of lattice enumeration can make a significant difference to the speed of lattice sieving.

This paper is organized as follows. In section 2, we give a very brief overview of the Number Field Sieve algorithm in the medium-characteristic case. A more detailed explanation can be found in [11]. One of the main bottlenecks of this algorithm is lattice sieving, which involves enumerating points in a (low-dimensional) lattice. We propose in section 3 a straightforward idea to significantly increase enumeration speed in dimension three and above. The idea is to change the angle of planes that are sieved through in order to reduce the

***Corresponding Author: Gary McGuire:** UCD School of Mathematics and Statistics, University College Dublin, Ireland
Oisın Robinson: UCD School of Mathematics and Statistics, University College Dublin, Ireland

number of planes. This idea has been used before for lattice enumeration in a sphere [17], however it has not been applied successfully to lattice sieving for the NFS. We show that the idea can work well by using integer linear programming to find an initial point for iteration in a plane within the sieve cuboid. In section 4 we propose a novel method to amortize memory communication overhead which applies regardless of dimension. In section 5 we give details of a new record discrete log computation in $\mathbb{F}_{p^6}$. The previous record due to Grémy et al [11] had p^6 with 422 bits, and this paper has p^6 with 423 bits. We deliberately chose a field size just one bit larger in order to attempt a direct comparison of methods and timings. In section 6 we present a record pairing break with the same prime p. Finally we conclude in section 7 and mention some possible future research ideas.

2 Number Field Sieve

We start with a very brief sketch of the NFS in the most naive form suitable for computing discrete logs in $\mathbb{F}_{p^n}$. Consider the following commutative diagram:
The polynomials $f_0(x)$ and $f_1(x)$ are irreducible in $\mathbb{Z}[x]$ of degree n, and they define the number fields $\mathbb{Q}(\alpha)$

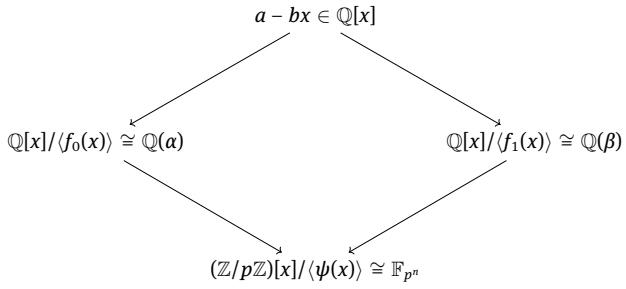

Figure 1: Commutative diagram of NFS for discrete log in $\mathbb{F}_{p^n}$.

and $\mathbb{Q}(\beta)$ respectively. We require that $f_0(x)$ and $f_1(x)$, when reduced modulo p, share a factor $\psi(x)$ of degree n which is irreducible over $\mathbb{F}_p$. This defines the finite field $\mathbb{F}_{p^n}$ as $(\mathbb{Z}/p\mathbb{Z})[x]/\langle\psi(x)\rangle$. Usually $\psi(x)$ is simply the reduction of $f_0(x)$ modulo p.

In two-dimensional sieving, for a bound B, we inspect many pairs of integers (a, b) with $0 < a \le B$ and $-B \le b \le B$ in the hope of finding many pairs such that

$$\mathrm{Res}(f_0, a - bx) \qquad \text{and} \qquad \mathrm{Res}(f_1, a - bx)$$

are both divisible only by primes up to a bound B_2. This B_2 is called the smoothness bound, or the large prime bound. The set of all polynomials $a - bx$ that we inspect is called the search space.

There is also a bound B_1, called the factor base bound, which is the largest prime used in sieving. The factor base consists of all primes up to this bound B_1. In brief, we seek two different representations of (the image of) $a - bx$ over the factor base. There are excellent detailed explanations of the NFS, see [3, 10, 11, 16]. We refer the reader to those papers for further details. Recently, new variations of NFS have been described where the norms (i.e. resultants) are even smaller in certain fields, see [18, 21].

2.1 Lattice Sieving

The original lattice sieve (due to J.M. Pollard [20]) has developed into the 'special-q' lattice sieve which we now outline. Let q be a rational prime, let r be an integer with $f_0(r) \equiv 0 \mod q$, and let $\mathfrak{q} = \langle q, \theta - r\rangle$ be an ideal of $K = \mathbb{Q}(\theta) \cong \mathbb{Q}[x]/\langle f_0\rangle$ lying over q. We choose $\mathfrak{q}$ to be smaller than our smoothness bound. We fix a

factor base, which will consist of all prime ideals in the ring of integers of K whose norm is smaller than our pre-determined factor base bound. We look for (integral) ideals of K that are divisible by $\mathfrak{q}$, and we do this by looking for ideals whose norm is divisible by q. We also would like the norm to be divisible by many other small primes p. We fix q and iterate over all p in the factor base using a sieve.

In 3-dimensional sieving, the pairs (a, b) corresponding to $a - bx$ (in section 2) become triples (a, b, c) corresponding to $a + bx + cx^2$. A sieving ideal will have the form $\langle a + bx + cx^2 \rangle$. Relation collection examines a subset $\mathcal{S}$ of the whole set of polynomials $A(x)$ of degree 2. The subset $\mathcal{S}$ is called the *search space* and is made of the polynomials $A(x)$ of bounded coefficients.

We perform sieving in three dimensions as follows. We use a fixed-size sieve region $H = [0, B[\times[-B, B[\times[-B, B[$ where each lattice point will correspond to a norm which is always divisible by q and hopefully divisible by many p. Define lattices Λ_q and Λ_{pq} by

$$L_q = \begin{bmatrix} q & -r & 0 \\ 0 & 1 & -r \\ 0 & 0 & 1 \end{bmatrix}, \qquad L_{pq} = \begin{bmatrix} pq & -t & 0 \\ 0 & 1 & -t \\ 0 & 0 & 1 \end{bmatrix}$$

where $f_0(r) \equiv 0 \mod q$ and $f_0(t) \equiv 0 \mod pq$, and the columns are a basis. Compute an LLL-reduced basis for both Λ_q and Λ_{pq} to get matrices L'_q and L'_{pq}. Then let

$$L' = (L'_q)^{-1} \cdot L'_{pq}$$

which is an integer matrix by construction. Let Λ' be the lattice with basis L'. We mark all (i, j, k) in $H \cap \Lambda'$. As a result, for a sieve location (i, j, k) that has been marked, if we let

$$\begin{bmatrix} a \\ b \\ c \end{bmatrix} = L'_q \cdot \begin{bmatrix} i \\ j \\ k \end{bmatrix}$$

then we know that the norm of $\langle a + b\theta + c\theta^2 \rangle$ is divisible by both q and p.

We compute and reduce L_q once per special-$\mathfrak{q}$, and compute L_{pq} etc for each p. We compute (a, b, c) only for (i, j, k) that have been marked for many p (above a pre-determined threshold).

Our new results have two aspects. First, in section 3 we improve the speed of enumeration of points in dimensions higher than two. Second, in section 4 we give a new way of avoiding cache locality issues by the use of a histogram of lattice point hits. This applies regardless of dimension.

3 Faster enumeration

In a lattice Λ of rank n recall that the i-th successive minimum is defined by

$$\lambda_i(\Lambda) = \inf\{r \in \mathbb{R} : \dim(\mathrm{span}(\Lambda \cap B_r)) \geq i\}$$

where $B_r = \{x \in \mathbb{R}^n : ||x|| \leq r\}$. In particular, $\lambda_1(\Lambda)$ is the length of a shortest nonzero vector in Λ. A basis $v_1, \ldots, v_n$ for Λ is said to be a Minkowski-reduced basis if, for $k = 1, 2, \ldots, n$, v_k is the shortest lattice element that can be extended to a basis with $v_1, \ldots, v_{k-1}$.

We assume we are sieving in three dimensions. We fix a bound B and let

$$H = [0, B[\times[-B, B[\times[-B, B[$$

be the sieving region. Let Λ' be the lattice defined in section 2. The problem is to list the elements of $\Lambda' \cap H$ in an efficient way. In previous work this is done by going through the planes parallel to the xy-plane, and enumerating the lattice points in each of these planes. We propose a different method which uses fewer planes.

Let v_1, v_2, v_3 be vectors having lengths $\lambda_1(\Lambda'), \lambda_2(\Lambda'), \lambda_3(\Lambda')$, the first three successive minima of Λ'. These three vectors are guaranteed to exist and we can either find all three, or an acceptably close approximation (see Remark 1). The origin together with v_1 and v_2 define a plane which we call P. Let

$$c_{max} = \max\{c \in \mathbb{N} : H \cap (P - c \cdot v_3) \neq \emptyset\}.$$

We refer to the plane $G = P - c_{max} \cdot v_3$ as the 'ground plane'.

Our approach is very simple: to enumerate all lattice points in H, we enumerate all points in the ground plane G that lie in H, and then all points in the translates $G + kv_3$ for $k = 1, 2, 3, \ldots$ that lie in H, until we reach the last translate intersecting H.

Remark 1. Finding v_1, v_2, v_3 is done with the LLL algorithm. In practice, in very small dimension such as three, this is sufficient to find a Minkowski-reduced basis, or a close approximation which is good enough for our purposes.

Remark 2. To easily enumerate points in a plane $G + kv_3$, we first locate one point p_0 that is contained within the plane and the sieving region H. For this, we use integer linear programming (described in this context below). Once we have located p_0, we proceed to enumerate points in this plane by adding and subtracting multiples of v_1 and v_2 from p_0, until by doing so we are no longer within H. This is done inductively, by first enumerating all $p_0 + c_1 v_1$ where c_1 runs over all integers such that $p_0 + c_1 v_1$ is in H. Then we add v_2 and enumerate all $p_0 + v_2 + c_1 v_1$ where c_1 runs over all integers such that $p_0 + v_2 + c_1 v_1$ is in H. Then we add v_2 again, and repeat. This may not be the optimal method of enumerating points in $G + kv_3$, however it worked well in our computations and is sufficient for our purposes. Pseudo-code for this can be seen in Algorithm 1.

This inductive procedure will extend to higher dimensions, as long as the integer linear programming problem required to find the corresponding feasible points is tractable. We expect this to be the case certainly up to dimension six, and perhaps further.

Remark 3. If the lattice is very skewed, it is possible that the last valid sieving point in the plane is $p_k = p_{k-1} + v_1 + c \cdot v_2$, where $c \geq 2$ (and p_{k-1} is the previous point). It would be preferable to be able to reach all points by unit additions of v_2 so for practical purposes, we do this and ignore the cases where such 'outlier' points are missed.

We have quantified the percentage of missed points, based on test data from the record computation and comparison with an exhaustive search. We consistently found about 1.8% of points were missed per special-q. Since there are usually up to hundreds of thousands of primes on a sieving side, and multiple planes per prime, it would only take a small number of missed points per prime to account for this percentage. This likely happens at planes intersecting only corners of the sieve region.

Remark 4. In two dimensions, the sieving method of Franke and Kleinjung [7] is very efficient. Our approach works in the 2d case also, using the first two successive minima of the 2d lattice, but it will not quite compete with the method in [7] in terms of speed of enumeration because we must do a little extra work when dealing with boundaries. This shows that dealing with the boundaries of the sieving region is not trivial.

We summarize the above description in Algorithm 1.

Remark 5. The observer may notice that in Algorithm 1 as presented, there is no guarantee upon adding (v_1, v_2, v_3) to (x, y, z) to move to the next row, that (x, y, z) is still within the sieving plane and boundary. We have written the algorithm like this to get across the main idea, but in the code every time we switch to a new row, we solve a mini integer linear programming task to get a multiple of (u_1, u_2, u_3) to subtract from and pin (x, y, z) to the start of the row while keeping it within the boundary. Solving a one-dimensional integer linear programming problem can be done in-line, and does not lead to much extra code complexity.

Algorithm 1 3*D* Lattice Sieve

Input:

Boundary $[0, B[\times[-B, B[\times[-B, B[$, special-q prime q, polynomial f, memory array M.

Output:

List of prime divisibility events contained in M

procedure

$r \leftarrow$ root of $f \bmod q$

$$L \leftarrow \begin{bmatrix} n & -r & 0 \\ 0 & 1 & -r \\ 0 & 0 & 1 \end{bmatrix}$$

Reduce L in-place via L^2 algorithm (i.e. efficient LLL)

$m \leftarrow 0$

for every factor base prime p **do**

for every root s of $f \bmod p$ **do**

$n \leftarrow p \cdot q$

$t \leftarrow q \cdot \left(s \cdot \left(q^{-1} \bmod p\right) \bmod p\right) + p \cdot \left(r \cdot \left(p^{-1} \bmod q\right) \bmod q\right)$ // Chinese remainder theorem

If $t >= n$, $t \leftarrow t - n$

$$L_2 \leftarrow \begin{bmatrix} n & t & 0 \\ 0 & 1 & t \\ 0 & 0 & 1 \end{bmatrix}$$

Reduce L_2 in-place via L^2 algorithm

$L_3 \leftarrow q \cdot L^{-1} \cdot L_2$

$L_3 \leftarrow L_3/q$ (all entries)

Reduce L_3 in-place via L^2 algorithm

$u = (u_1, u_2, u_3) \leftarrow$ first column of L_3

$v = (v_1, v_2, v_3) \leftarrow$ second column of L_3

$w = (w_1, w_2, w_3) \leftarrow$ third column of L_3

$(n_x, n_y, n_z) \leftarrow (u_2v_3 - u_3v_2, u_3v_1 - u_1v_3, u_1v_2 - u_2v_1)$ // Compute normal

$(x, y, z) \leftarrow (w_1, w_2, w_3)$

Repeatedly subtract (w_1, w_2, w_3) from (x, y, z) until we are at the ground plane

while Plane defined by (n_x, n_y, n_z) and (x, y, z) intersects box $[0, B[\times[-B, B[\times[-B, B[$ **do**

$(ws_1, ws_2, ws_3) \leftarrow (x, y, z)$ // Remember starting w vector

$(a, b) \leftarrow (0, 0)$

Compute (a, b) using algorithm 2 to place (x, y, z) within boundary

$(x, y, z) \leftarrow (x, y, z) + a \cdot (u_1, u_2, u_3) + b \cdot (v_1, v_2, v_3)$

$(s_1, s_2, s_3) \leftarrow (x, y, z)$ // Remember starting vector

while $(x, y, z) \in [0, B[\times[-B, B[\times[-B, B[$ **do**

Subtract (u_1, u_2, u_3) from (x, y, z) until we are at the 'start of the row'

for every vector in current row **do**

$id \leftarrow x + ((y + B) \ll log_2(B)) + ((z + B) \ll log_2(B^2))$

$M[m{+}{+}] \leftarrow \{id, log_2(p)\}$

$(x, y, z) \leftarrow (x, y, z) + (u_1, u_2, u_3)$

end for

$(x, y, z) \leftarrow (x, y, z) + (v_1, v_2, v_3)$

end while

$(x, y, z) \leftarrow (s_1, s_2, s_3) - (v_1, v_2, v_3)$

Repeat the above while loop in the opposite direction in the plane

That is, by subtracting (v_1, v_2, v_3) from (x, y, z) to move to the next row.

We can still progress within each row by pinning to start, then adding (u_1, u_2, u_3).

When finished in both directions, move to next plane:

$(x, y, z) \leftarrow (ws_1, ws_2, ws_3) + (w_1, w_2, w_3)$

end while

end for

end for

end procedure

3.1 Previous Lattice Enumeration Methods

Lattice enumeration is widely used in algorithms to solve certain lattice problems, such as the Closest Vector Problem. However, sieving in a cuboid introduces many complications that do not occur when sieving in a ball.

Our enumeration here is similar to Fincke-Pohst-Kannan's algorithm (FPK) for lattice enumeration [6, 17]. However, it is significantly different in that we sieve in a box, as opposed to a sphere. Further, we do not compute a norm for every point to test if it is within the boundary - use of a box allows us to separate many points which may be treated in fast loops with no individual boundary checking. In practice this makes a huge difference. Note that L. Grémy's space sieve is 120 times faster than the FPK algorithm (see [10]). We outperform the space sieve by over 2.5×. Note also that sieving in a rectangular region is fundamental to Franke and Kleinjung's 2d lattice sieve algorithm, and its success depends on the shape of this region.

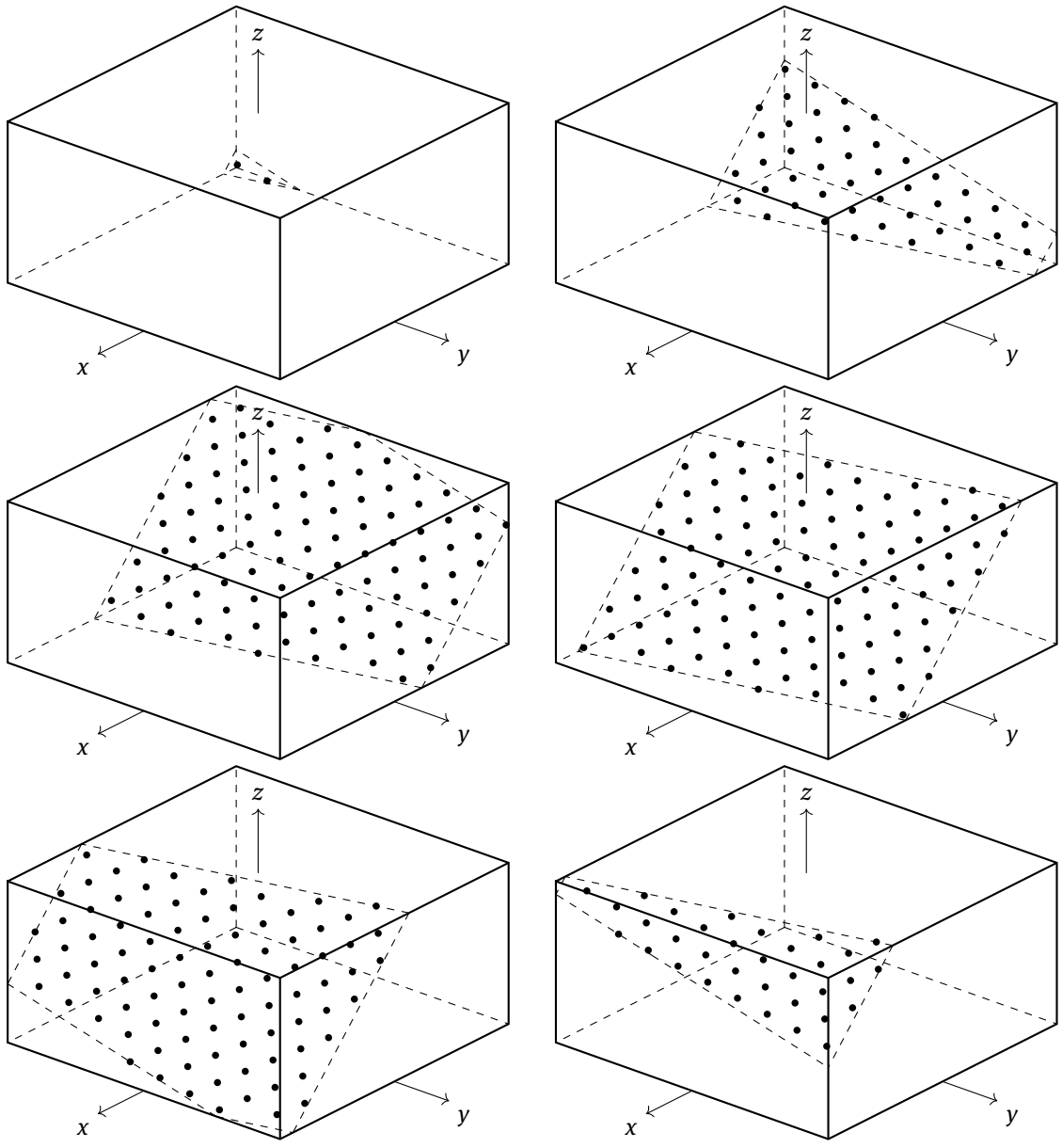

Figure 2: Six dense sublattices cover every point in the sieve region.

3.2 Example

The lattice used in Figure 2 is the following:

$$\begin{bmatrix} 10 & 18 & 35 \\ -12 & 18 & 13 \\ -7 & -22 & 18 \end{bmatrix}$$

The sieve region is $[-100, 100] \times [-100, 100] \times [0, 100]$. In this example, using our method, 6 planes cover every valid sieving point. With traditional plane sieving, using planes that are parallel to the base of the sieving cuboid, 101 planes are needed, each with at most four lattice points.

3.3 Integer Linear Programming

Given a plane defined by u, v and a point R, with R not necessarily contained in the sieving region defined by $H = [0, B[\times[-B, B[\times[-B, B[$, the task is to find a point $p_0 = (x_0, y_0, z_0)$ that is provably contained in the intersection of the plane and H, if such a point exists. We look for $a, b \in \mathbb{Z}$ such that $p_0 = R + a \cdot u + b \cdot v$ and $p_0 \in H$.

This can be formulated as an integer linear programming problem, where the aim is to either minimize x, subject to

$$A \cdot x \leq C$$

where $x = (a, b) \in \mathbb{Z}^2$ and $A \in \mathbb{Z}^{6\times 2}$, $C \in \mathbb{Z}^6$, depend on u, v, B, or find any feasible point, if one exists. This problem is well studied, and though it is NP-hard in general, can be solved easily in small dimensions. It is computationally trivial in dimension 3, for example, which we use in this article.

We shall give more details now. Our approach is based on 'Fourier-Motzkin' elimination. Given $R = (x, y, z), u = (u_1, u_2, u_3), v = (v_1, v_2, v_3)$ we seek a, b such that $R + a \cdot u + b \cdot v$ is feasible. We set up the following system of inequalities based on the boundary $[0, B[\times[-B, B[\times[-B, B[$:

$$\begin{bmatrix} u_1 & v_1 \\ u_2 & v_2 \\ u_3 & v_3 \\ -u_1 & -v_1 \\ -u_2 & -v_2 \\ -u_3 & -v_3 \end{bmatrix} \begin{bmatrix} a \\ b \end{bmatrix} \leq \begin{bmatrix} B - x - 1 \\ B - y - 1 \\ B - z - 1 \\ x \\ B + y \\ B + z \end{bmatrix}.$$

Our approach is to eliminate a, deduce b and then compute a. We first transform the system so that the entries of the first column of A are normalized (i.e. we multiply across by the least common multiple of u_1, u_2, u_3 and divide across respectively by $|u_1|, |u_2|, |u_3|$. Thus each entry in column 1 of A is $\pm k$ for some $k \in \mathbb{Z}$). We look for all pairs of rows of A such that a cancels. We achieve this with a nested loop in (i, j) and where we disregard a pair of rows if both a and b cancel (this happens e.g. if $|i - j| = 3$). For a given pair of rows where we have eliminated the term in a, it is then easy to compute

$$b = \left\lfloor \frac{C_i + C_j}{V_i + V_j} \right\rfloor$$

where C_i are the constants in the RHS vector, and V_i are the coefficients of the vector v in A, as long as $V_i + V_j \neq 0$ (otherwise we skip this (i, j)). When we consider all pairs, we keep the smallest value of b found. We then compute the smallest corresponding value of a by back-substitution in all 6 rows. This is summarized in algorithm 2.

Remark 6. It is possible that no feasible point exists or is found, but we can detect this by checking if p_0 is within bounds.

Algorithm 2 Integer Linear Programming

Input:

Boundary $[0, B[\times[-B, B[\times[-B, B[$, plane P defined by $u = (u_1, u_2, u_3)$, $v = (v_1, v_2, v_3)$, $R = (x, y, z)$ such that $R \in P$, but R not necessarily in boundary.

Output:

(a, b) such that $p_0 = R + a \cdot u + b \cdot v$ contained in both P and boundary.

procedure

$U \leftarrow \{u_1, u_2, u_3, -u_1, -u_2, -u_3\}^T$
$V \leftarrow \{v_1, v_2, v_3, -v_1, -v_2, -v_3\}^T$
$C \leftarrow \{B - x - 1, B - y - 1, B - z - 1, x, B + y, B + z\}^T$
$L \leftarrow |\mathrm{LCM}(u_1, u_2, u_3)|$
Normalize system - multiply all entries by L, divide U_i, V_i, C_i by original $|U_i|$
$a \leftarrow B, b \leftarrow B$
for all pairs of rows (i, j) **do**
  if $U_i = 0$ and $V_i > 0$ **then**
    $b_{TRIAL} = \lfloor C_i / V_i \rfloor$
    if $b_{TRIAL} < b$ then $b \leftarrow b_{TRIAL}$
  else
    if $U_i < 0$ and $U_j > 0$ and $|i - j| \neq 3$ **then**
      $D \leftarrow V_i + V_j$
      if $D > 0$ **then**
        $b_{TRIAL} = \lfloor (C_i + C_j) / D \rfloor$
        if $b_{TRIAL} < b$ then $b \leftarrow b_{TRIAL}$
      end if
    end if
  end if
end for
for the best value of b, compute a by back substitution.
Return (a, b)

end procedure

4 Dealing with Cache Locality Issues

Representing a 3d lattice in memory can be done in different ways. One problem with arranging points first by adjacent lines, then adjacent planes etc is that the storage/retrieval of points tends to result in random memory access patterns, which severely impacts performance. This is a fundamental concern in large-scale computation. Computer manufacturers address this by providing various levels of 'cache', i.e. a limited quantity of high-speed memory, too costly for main memory, which is used as a temporary store of frequently-accessed or burst-access data. Traditionally, lattice-sieving has always had to make use of cache to minimize the cost of the random memory access patterns that occur in practice. The usual way this is handled is via 'bucket sorting' [1].

Our idea to improve cache locality without using bucket sorting is simple: list and sort. We encode the x, y, z coordinates of each lattice point as a 32-bit integer. A sieve 'hit' is an ordered pair (integer, $\log p$) where p is a rational prime that divides the norm of the ideal corresponding to the lattice point/integer. Here $2 \le p \le B_1$ where B_1 is the factor base bound. We store all sieve hits in a list of increasing size. Because the list increases strictly linearly, this is ideally suited to fast memory access and is compatible with all levels of cache. By itself, this is not an advantage because we have merely collected a long list of randomly ordered pairs. However, we can sort this list by sorting the integers in increasing order, i.e., we sort these pairs by the integers only. We

can then recover lattice points with a large smooth part via a linear scan of the sorted list, because if an integer appears K times as the first coordinate of a pair then it will appear with K different values of p. The more times it appears, the smoother the lattice point is.

Remark 7. Our list/sort approach must be considered carefully in terms of memory because we must store $\log p$ and full co-ordinates for every sieve event. While in practice sorting the list of all sieve events is still fast, the memory required to store the full list is considerable (it certainly does not fit in cache all at once, but cache is still essential and is used implicitly in the sorting routine, which is cache-friendly). This meant we had to ensure our sieving parameters could be set to fully utilize the compute capacity of our sieving nodes. See section 5 for further details.

Remark 8. For our record computation (see next section) we used a sieving region H of size $2^9 \times 2^{10} \times 2^{10}$. A traditional lattice sieve would thus take 2^{29} bytes, or exactly 512MB to store the sieve (assuming one byte to store $\log p$ per sieve location). We need to store 5 bytes per sieve hit, one byte for $\log p$ plus 4 bytes for the (x, y, z) coordinates, but note that we do not have to represent every possible sieve location, only the hits. In practice, for the parameters we used to set the record, each instance of the sieve program records roughly 500 million sieve events, and so requires about 5×500 million bytes = 2500MB of RAM in total per special-q. This is approximately a fivefold increase over 512MB for a traditional lattice sieve, but it was not a problem for us as our production nodes had the memory to cope.

Remark 9. We avoid sieving the smallest primes, as the corresponding lattices are very dense and greatly increase the memory required for the list + sort approach. This would not really be an issue if we used bucket sorting, but either way these smallest primes can probably be left out, because they tend to appear in almost every relation. The less dense lattices (not for small primes) should be sieved, and our lattice enumeration method will be very fast here.

Remark 10. Grémy et al's [11] sieve does not include bucket sort, and neither do we. It could be a possible improvement to replace our list/sort with enumerate/bucket sieve (and we could retain our fast enumeration idea as bucket sorting relates only to memory access).

We compare sieving statistics in Table 1 between our implementation and that of [11]. Note that all of these times give total special-q time excluding the cofactorization time. In [11], sieving and memory access are intertwined and so we compare this to our combined sieving/listing/sorting time. Our siever is over 4.5× faster. Sieving speed in our case is not the bottleneck. The extra sieving speed is gained at the expense of increased memory requirements.

Table 1: Sieve performance comparison (times in seconds)

Authors	factor base bound	H	q_{min}	q_{max}	$\#\{q\}$	av. time	min time	max time
GGMT	2^{21}	$2^{10} \times 2^{10} \times 2^8$	16000000	16001000	7	143.93	142.17	145.28
GGMT	2^{21}	$2^{10} \times 2^{10} \times 2^8$	86500000	86501000	14	142.07	140.53	143.82
GGMT	2^{21}	$2^{10} \times 2^{10} \times 2^8$	262000000	262001000	9	142.40	140.95	144.34
GGMT	2^{22}	$2^{10} \times 2^{10} \times 2^8$	16000000	16001000	7	169.74	166.12	171.82
GGMT	2^{22}	$2^{10} \times 2^{10} \times 2^8$	86500000	86501000	14	167.53	166.01	173.55
GGMT	2^{22}	$2^{10} \times 2^{10} \times 2^8$	262000000	262001000	9	167.50	165.02	172.17
this work	2^{24}	$2^9 \times 2^{10} \times 2^{10}$	16000000	16001000	7	35.47	34.94	36.95
this work	2^{24}	$2^9 \times 2^{10} \times 2^{10}$	86500000	86501000	14	35.80	35.39	37.16
this work	2^{24}	$2^9 \times 2^{10} \times 2^{10}$	262000000	262001000	9	36.37	35.71	37.54

5 Record computation in F_{p^6}

We implemented the 3d case of our lattice sieving idea in C and used it to set a new record in solving the discrete log in the multiplicative subgroup $(\mathbb{F}_{p^6})^\times$. Previous records were set by Zajac [24], Hayasaka et al (HAKT) [15], and Grémy et al (GGMT) [11]. All computations were done on the main compute nodes of the Kay cluster at ICHEC, the Irish Center for High-End Computing. Each node consists of dual 20-core Intel Xeon Gold 6148 (Skylake) processors @ 2.4 GHz, with 192Gb RAM per node. All timings have been normalized to a nominal 2.0GHz clock speed.

With $\phi = (1+\sqrt{5})/2$, we chose the prime $p = \lfloor 10^{21} \cdot \phi \rfloor + 29$. Our target field is $\mathbb{F}_{p^6}$, where p^6 has 423 bits. This is comparable to the field size of the previous record at 422 bits [11]. One consequence of our choice is to allow a fair comparison of the total effort required to solve discrete logs in a field of this order of magnitude.

5.1 Polynomial selection

We implemented the Joux-Lercier-Smart-Vercauteren (JLSV_1) algorithm and ranking polynomials by their 3d Murphy E-score, after about 100 core hours found the following polynomial pair from the cyclic family of degree six described in [9]:

$$f_0 = x^6 - 40226000394x^5 - 100565001000x^4 - 20x^3 + 100565000985x^2 + 40226000400x + 1$$
$$f_1 = 80447172120x^6 + 104483881186x^5 - 945497878835x^4 - 1608943442400x^3 - 261209702965x^2 + 378199151534x + 80447172120$$

We computed the 3d alpha score for these and found $\alpha(f_0) = -3.6$ and $\alpha(f_1) = -12.6$.

5.2 Relation collection

Our implementation was written as a standalone executable, independent of CADO-NFS, producing relations in the format that CADO-NFS can use. We carry out cofactorization using Pollard's $p-1$ algorithm and two rounds of Edwards elliptic curve factorization. The cofactorisation implementation uses a standard approach and is not an improvement on CADO-NFS's cofactorization rig. Cofactorization typically takes between 120-130 seconds, this is the bottleneck.

Our program is extremely fast to sieve. This allowed us not only to use a larger factor base, but also to search for relations that are 'twice as difficult' to find, i.e. to use a smoothness bound of 2^{28} as opposed to the 2^{29} used in the 422-bit record. We were able to use a factor base bound of 2^{24} with no major loss of speed. Typically sieving takes less than 40 seconds per special-q, including sorting.

The time per special-q was roughly constant across the entire range, at between 150-170 seconds. The total time per special-q is made up of sieving plus cofactorization. There were about 500 million sieve hits per special-q. As we explained in Remark 8 each sieve hit takes exactly 5 bytes of memory, so we need about 5×500 million bytes = 2500MB of RAM per special-q. We were able to utilize all 40 cores on our sieving nodes, where each node has 192Gb of memory. We needed about 2500MB $\times 40$ = 100GB in total.

We were able to use a larger sieve region due to the speed of lattice enumeration. We used a sieving region of size $2^9 \times 2^{10} \times 2^{10}$, compared to the region $2^{10} \times 2^{10} \times 2^8$ used in the previous record.

We sieved most special-qs on the f_0 side with norm between 16M and 263M. Our program sieves only one ideal in each Galois orbit. We apply the Galois automorphism as a post-processing step. We found 7,152,855 unique relations and then applied the Galois automorphism (which is trivial in core-hours) and after removing duplicates we were left with 34,115,391 unique relations. The total sieving effort was 69,120 core hours.

Table 2: Key statistics of record computations in $\mathbb{F}_{p^6}$. All timings in core hours. Recall that $\mathcal{S}$ is the search space, q is the norm of $\mathfrak{q}$, H_0, H_1, H_2 are the bit lengths of sieving dimensions, and q_{max} is the largest special-$\mathfrak{q}$ norm we encounter (which will be just below the smoothness bound).

Authors	GGMT	This work
Field size (bits)	422	423
α-values	−2.4,−14.3	−3.6,−12.6
Murphy-E	$2^{-20.51259}$	$2^{-20.45961}$
Sieving region H	$2^{10} \times 2^{10} \times 2^{8}$	$2^{9} \times 2^{10} \times 2^{10}$
Factor base bounds	$2^{21}, 2^{21}$	$2^{24}, 2^{24}$
Smoothness bounds	$2^{29}, 2^{29}$	$2^{28}, 2^{28}$
$\#\mathcal{S} = q_{max}2^{H_0+H_1+H_2}$	2^{55}	2^{57}
Special-$\mathfrak{q}$ side	0	0
Range of $\mathfrak{q}$	$]2^{21}, 2^{27.9}[$	$]2^{23.9}, 2^{30}[$
Galois action	6	6
#unique relations	71,850,465	34,115,391
#required relations	$\approx 56M$	29,246,136
purged	18,335,401	7,598,223
filtered	5,218,599	2,754,009
Total sieving time	201,600	69,120

5.3 Construction of matrix

We modified CADO-NFS [23] to produce a matrix arising from degree-2 sieving ideals for the linear algebra step.

5.4 Linear algebra

We used the Block Wiedemann implementation in CADO-NFS (we compiled commit `d6962f667d3c...` with MPI enabled), with parameters $n = 10$ and $m = 20$. Due to time constraints, we needed to minimize wall clock time so we chose to run the computation on 4 nodes, to reduce the iteration time for the Krylov sequences. Also, to avoid complications, we did not run the 10 Krylov sequences in parallel. The net result was that we spent 11,760 core hours on the Krylov step, which is suboptimal (but got us the result in time). It took 24 core hours (on one core) to compute the linear generator and 672 core hours for the solution step. This gave $2,754,009$ of the factor base ideal virtual logarithms. We ran the log reconstruction to give a final total of $25,215,976$ known virtual logarithms out of a possible total of $29,246,136$ factor base ideals.

We note that a similar-sized matrix was solved in [11], which used 1,920 core hours for the Krylov step. However, due to our choice of smoothness bound, set to 2^{28}, our linear algebra effort to set the new record was considerably less than that of the previous record of 422 bits, which involved a Krylov step taking 23,390 core hours for a smoothness bound set to 2^{29}.

5.5 Individual logarithm

Take the element $g = x + 2 \in \mathbb{F}_{p^6} = \mathbb{F}_p[x]/\langle f_0(x)\rangle$. Let

$$\ell = 958986809065895548825976460009393482920 9,$$

a large prime factor of $p^2 - p + 1$. Let $h = \left(p^6 - 1\right)/\ell$. Note that g is not a generator of the entire multiplicative subgroup of $\mathbb{F}_{p^6}$, but we do have that g^h is a generator of the subgroup of size ℓ. It is easy to compute $\mathrm{vlog}(g)$ since $N_0(g) = -3^3$.

We have vlog(g) = 895106961716290895353618327493761398526 5. We chose the target

$$t = 3141592653589793238 46x^5 + 2643383279502884197 16x^4 + 9399375105820974944 59x^3 + 2307816406286208998 62x^2 + 8034825342117067982 14x + 8086513282306647093 84$$

We implemented the initial splitting algorithm of A. Guillevic [13] in SAGE, and after a few core hours found that

$$g^{74265}t = uvw(-129592286880919x^2 - 103570474976165x - 5550010113050)$$

where $u \in \mathbb{F}_{p^2}, v \in \mathbb{F}_{p^3}, w \in \mathbb{F}_p$, so that their logarithm modulo ℓ is zero. The norm of the latter term is $-11 \cdot 37 \cdot 71 \cdot 97 \cdot 197 \cdot 821 \cdot 24682829 \cdot 33769709 \cdot 83609989 \cdot 13978298429383 \cdot 21662603713879 \cdot 74293619085767 \cdot 141762919001833 \cdot 381566853770521$. We had 5 special-q to descend, the largest having 49 bits. We used our 3d lattice sieve implementation to descend from these ideals of unknown log to factor base elements with known logarithms. This was a somewhat manual process and took about 8 hours.

Table 3: Comparison with other record computations in $\mathbb{F}_{p^6}$. All timings in core hours.

year	size of p^n	authors	algorithm	rel. col.	lin. alg.	total
2008	240	Zajac	NFS-HD	580	322	912
2015	240	HAKT	NFS-HD	527	-	-
2017	240	GGMT	NFS-HD	22	5	27
2017	300	GGMT	NFS-HD	164	39	203
2017	389	GGMT	NFS-HD	18,960	2,400	21,360
2017	422	GGMT	NFS-HD	201,600	26,880	228,480
2019	423	this work	NFS-HD	69,120	12,480	81,600

We obtained vlog(t) = 261962363706411635934642846706828724587 0, so that

$$\log_g(t) \equiv \text{vlog}(t)/\text{vlog}(g) \equiv 74358267505170152697182304026455579478 80 \bmod \ell.$$

6 Pairing break

Let p be the same prime as in the previous section. Define $\mathbb{F}_{p^2} = \mathbb{F}_p[i]/\langle i^2 + 2\rangle$. The curve $E/\mathbb{F}_{p^2} : y^2 = x^3 + b$, $b = i + 7$ is supersingular of trace p, hence of order $p^2 - p + 1$. Define $\mathbb{F}_{p^6} = \mathbb{F}_{p^2}[j]/\langle j^3 - b\rangle$. The embedding field of the curve E is $\mathbb{F}_{p^6}$. We take

$$G_0 = (5, 7515683283144806887 40i + 7516425540833156884 93)$$

and we check that $G_1 = [273]G_0$ is a generator of $E(\mathbb{F}_{p^2})[\ell]$. The distortion map $\phi : (x, y) \mapsto \left(x^p / \left(jb^{(p-2)/3}\right), y^p / \left(b^{(p-1)/2}\right)\right)$ gives a generator $G_2 = \phi(G_1)$ of the second dimension of the ℓ-torsion. We take the point

$$P_0 = (3141592653589793238 46i + 2643383279502884197 17,$$

$$1560320966141767888 064i + 3680673645359915583 80)$$

from the decimals of π, and $P_1 = [273]P_0 \in E(\mathbb{F}_{p^2})[\ell]$ is our challenge. We aim to compute the discrete logarithm of P_1 to base G_1. To do so, we transfer G_1 and P_1 to $\mathbb{F}_{p^6}$, and obtain $g = e_{Tate}\left(G_1, \phi(G_1)\right)$ and $t = e_{Tate}\left(P_1, \phi(G_1)\right)$, or

$$t = 709659446396722452 19x^5 + 760855550263311226560x^4 + 459517758627469463106x^3$$

$$+ 1075867962756498791880x^2 + 966415406496231787507x + 759380554536416832249,$$
$$g = 1445115464416256318145x^5 + 608219705720308630653x^4 + 132821383116103132604 9x^3$$
$$+ 104723931403852502861x^2 + 11182647223335284620 11x + 551285267384030855316$$

The initial splitting gave a 50-bit smooth generator

$$g^{289236} = uvw\left(-207659249318101x^2 - 32084626907475x + 36052674649889\right)$$

where $u \in \mathbb{F}_{p^2}, v \in \mathbb{F}_{p^3}, w \in \mathbb{F}_p$, so that their logarithm modulo ℓ is zero. The norm of the latter term is $11 \cdot 71 \cdot 79 \cdot 1453 \cdot 433123 \cdot 85478849 \cdot 34588617703 \cdot 40197196124443 \cdot 76694584420127 \cdot 370667620290007 \cdot 419573910884273 \cdot 823157513981483$. We had 6 special-q to descend. We also got a 49-bit smooth challenge of norm $23 \cdot 29^2 \cdot 41 \cdot 563 \cdot 2917 \cdot 1245103 \cdot 12006859 \cdot 107347203833 \cdot 506649149393 \cdot 39018481981309 \cdot 138780153403907 \cdot 174514280440993 \cdot 302260510161053$:

$$g^{91260}t = uvw\left(-59788863574984x^2 + 62066870577408x + 88384197770333\right)$$

We obtained $\mathrm{vlog}(g) = 759915148291253529528162192565836419591 3$ and
$\mathrm{vlog}(t) = 464222502376057311215259088735581932536 4$, so that
$\log_g(t) \equiv \mathrm{vlog}(t)/\mathrm{vlog}(g) \equiv 432595385604973025733233544349711543176 3 \bmod \ell$.

7 Conclusion

We have presented a new approach to lattice sieving in higher dimensions for the number field sieve, together with a novel approach to avoiding inefficient memory access patterns which applies regardless of dimension. In addition, we implemented the 3d case of our idea and used it to set a record in solving discrete log in $\mathbb{F}_{p^6}$, a typical target in cryptanalysis of pairing-based cryptography, in time a factor of more than $2.5\times$ better (in core hours) than the previous record, which was of a directly comparable size. It should be possible to improve the code further with more effort put into optimization. We have indicated that the sieving enumeration generalizes to higher dimensions as long as a certain integer linear programming problem is tractable. This has immediate implications for the possibility of implementation of the Tower Number Field Sieve and e.g. the Extended Tower Number Field Sieve, the latter of which is dependent on sieving in dimension at least four. The recent preprint [14] addresses one major prerequisite to the realization of TNFS and ExTNFS, concerning polynomial selection, while in the present work we give a strong indication that another obstruction, that of sieving efficiently in small dimensions of four and above, may be easier than first thought.

References

[1] Aoki, K., Ueda, H.: Sieving using bucket sort. In: Advances in cryptology—ASIACRYPT 2004, Lecture Notes in Comput. Sci., vol. 3329, pp. 92–102. Springer, Berlin (2004)

[2] Babai, L.: On Lovász' lattice reduction and the nearest lattice point problem. Combinatorica **6**(1), 1–13 (1986). 10.1007/BF02579403, https://doi.org/10.1007/BF02579403

[3] Barbulescu, R., Gaudry, P., Guillevic, A., Morain, F.: Improving NFS for the discrete logarithm problem in non-prime finite fields. In: Advances in cryptology—EUROCRYPT 2015. Part I, Lecture Notes in Comput. Sci., vol. 9056, pp. 129–155. Springer, Heidelberg (2015)

[4] Barbulescu, R., Gaudry, P., Joux, A., Thomé, E.: A heuristic quasi-polynomial algorithm for discrete logarithm in finite fields of small characteristic. In: Advances in cryptology—EUROCRYPT 2014, Lecture Notes in Comput. Sci., vol. 8441, pp. 1–16. Springer, Heidelberg (2014)

[5] Cohen, H.: A course in computational algebraic number theory, Graduate Texts in Mathematics, vol. 138. Springer-Verlag, Berlin (1993)

[6] Fincke, U., Pohst, M.: A new method of computing fundamental units in algebraic number fields. In: EUROCAL '85, Vol. 2 (Linz, 1985), Lecture Notes in Comput. Sci., vol. 204, pp. 470–478. Springer, Berlin (1985)

[7] Franke, J., Kleinjung, T.: Continued fractions and lattice sieving. In: Workshop record of SHARCS (2005) (2005), available at http://www.ruhr-uni-bochum.de/itsc/tanja/SHARCS/talks/FrankeKleinjung.pdf
[8] Gaudry, P., Grémy, L., Videau, M.: Collecting relations for the number field sieve in $\mathrm{GF}(p^6)$. LMS J. Comput. Math. **19**(suppl. A), 332–350 (2016)
[9] Gras, M.N.: Special units in real cyclic sextic fields. Math. Comp. **48**(177), 179–182 (1987)
[10] Gremy, L.: Sieve algorithms for the discrete logarithm in medium characteristic finite fields. In: Ph.D. thesis, Universite de Lorraine (2017), available at https://tel.archives-ouvertes.fr/tel-01647623
[11] Grémy, L., Guillevic, A., Morain, F., Thomé, E.: Computing discrete logarithms in $\mathbb{F}_{p^6}$. In: Selected areas in cryptography—SAC 2017, Lecture Notes in Comput. Sci., vol. 10719, pp. 85–105. Springer, Cham (2018)
[12] Guillevic, A.: Computing individual discrete logarithms faster in $\mathrm{GF}(p^n)$ with the NFS-DL algorithm. In: Advances in cryptology—ASIACRYPT 2015. Part I, Lecture Notes in Comput. Sci., vol. 9452, pp. 149–173. Springer, Heidelberg (2015)
[13] Guillevic, A.: Faster individual discrete logarithms in finite fields of composite extension degree. Math. Comp. **88**(317), 1273–1301 (2019)
[14] Guillevic, A., Singh, S.: On the Alpha Value of Polynomials in the Tower Number Field Sieve Algorithm (Aug 2019), https://hal.inria.fr/hal-02263098, working paper or preprint
[15] Hayasaka, K., Aoki, K., Kobayashi, T., Takagi, T.: An experiment of number field sieve for discrete logarithm problem over $gf(p^{12})$. JSIAM Letters **6** (Jan 2013)
[16] Joux, A., Lercier, R., Smart, N., Vercauteren, F.: The number field sieve in the medium prime case. In: Advances in cryptology—CRYPTO 2006, Lecture Notes in Comput. Sci., vol. 4117, pp. 326–344. Springer, Berlin (2006)
[17] Kannan, R.: Improved algorithms for integer programming and related lattice problems. In: Proceedings of the Fifteenth Annual ACM Symposium on Theory of Computing. p. 193–206. STOC '83, Association for Computing Machinery, New York, NY, USA (1983), https://doi.org/10.1145/800061.808749
[18] Kim, T., Barbulescu, R.: Extended tower number field sieve: a new complexity for the medium prime case. In: Advances in cryptology—CRYPTO 2016. Part I, Lecture Notes in Comput. Sci., vol. 9814, pp. 543–571. Springer, Berlin (2016)
[19] Kim, T., Jeong, J.: Extended tower number field sieve with application to finite fields of arbitrary composite extension degree. In: Public-key cryptography—PKC 2017. Part I, Lecture Notes in Comput. Sci., vol. 10174, pp. 388–408. Springer, Berlin (2017)
[20] Pollard, J.M.: The lattice sieve. In: The development of the number field sieve, Lecture Notes in Math., vol. 1554, pp. 43–49. Springer, Berlin (1993)
[21] Sarkar, P., Singh, S.: A general polynomial selection method and new asymptotic complexities for the tower number field sieve algorithm. In: Advances in cryptology—ASIACRYPT 2016. Part I, Lecture Notes in Comput. Sci., vol. 10031, pp. 37–62. Springer, Berlin (2016)
[22] Schirokauer, O.: Discrete logarithms and local units. Philos. Trans. Roy. Soc. London Ser. A **345**(1676), 409–423 (1993)
[23] The CADO-NFS development team: Cado-nfs, an implementation of the number field sieve algorithm (2019), available at http://cado-nfs.gforge.inria.fr/
[24] Zajac, P.: Discrete logarithm problem in degree six finite fields. In: Ph.D. thesis, Slovak University of Technology (2008), http://www.kaivt.elf.stuba.sk/kaivt/Vyskum/XTRDL