Slovak University of Technology
Faculty of Electrical Engineering and Information Technology
Department of Applied Informatics and Computing Technology

Discrete Logarithm Problem in Degree Six Finite Fields

Pavol Zajac

Dissertation

submitted for the degree of *Philosophiae Doctor, PhD.*

Supervisor: prof. RNDr. Otokar Grošek, PhD.

Program: 9.1.9 Applied mathematics

Bratislava 2008

Declaration

This thesis is a presentation of my original research work, with due reference to the literature.

Bratislava, February 2, 2008 Pavol Zajac

There is nothing concealed that will not be disclosed, or hidden that will not be made known.

Luke 12:2

Acknowledgements

I would like to thank my thesis supervisor Professor Otokar Grošek for the guidance and help in the research work. My gratitude belongs also to all who contributed to this project with expert advice or software implementation.

Contents

List of Tables

List of Figures

Notation and abbreviations

$\mathbb{Z}$	The set of all integers.
$\mathbb{Z}_{>0}$	The set of all positive integers.
$\mathbb{Z}_n$	The set of congruence classes modulo n, $\mathbb{Z}/n\mathbb{Z}$.
$\mathbb{Q}$	The set of all rational numbers.
$\mathbb{F}_{p^n}$	A finite field with characteristic p and degree n.
$K = \mathbb{Q}(\alpha)$	A number field.
$\mathcal{O}_K$	The ring of integers of a number field K.
$f, f(x)$	A polynomial.
$g \in G$	A generator of a cyclic finite group G.
i, j, k, m, n	Integers.
p, q	Primes.
α, β	Roots of polynomials.
ξ	An algebraic number.
$\mathfrak{p}, \mathfrak{q}$	Prime ideals.
$\mathfrak{B}$	A set of prime ideals (a factor base).
(ξ)	The principal ideal of ξ, $\xi\mathcal{O}_K$.
$N(\xi), N(\mathfrak{p})$	The norm of ξ, $\mathfrak{p}$.
$Tr(\xi)$	The trace of ξ.
$D(f), D(K)$	Discriminant of polynomial f, number field K.
$L_x(\alpha, c)$	Subexponential complexity function, defined by equation (2.4).

2D,3D,4D	Two-, three-, four- dimensional space
CRT	Chinese Remainder Theorem
DL	Discrete Logarithm
DLP	Discrete Logarithm Problem
DSA	Digital Signature Algorithm [**81**]
ECM	Elliptic Curve factorization Method
IFP	Integer Factorization Problem
GIPS	Billion (Giga) Instructions Per Second
NFS	Number Field Sieve
MIPS	Million Instructions Per Second
RSA	Cryptosystem of Rivest, Shamir, Adleman [**90**]
XTR	Efficient and Compact Subgroup Trace Representation, see Chapter 4
XTR-DL	Discrete Logarithm in XTR group

CHAPTER 1

Introduction

Algorithms and protocols of asymmetric cryptography are mostly based on two (conjecturally) difficult mathematical problems. First of them is the problem of factoring large integers (used e.g. in RSA [**90**]). The second one is the problem of computing discrete logarithms in finite groups (used e.g. in DSA [**81**]). There are more of the difficult problems that can be used in cryptography, e.g. based on logarithmic signatures [**75**]. However, none of them has gained enough popularity to replace the existing algorithms.

The integer factorization problem (IFP) can be defined in general as follows: Given composite integer n find two integers $p, q > 1$, such that $n = pq$. In cryptography, we are interested only in difficult cases of factorization, i.e. where p and q are large primes. There are many algorithms that solve general IFP. They can be split in two groups: general-purpose algorithms that can factor any integer n; and special-purpose that can factor only special integers, e.g. smooth numbers.

When assessing the strength of a cryptosystem, our prime concern is the actual and asymptotic complexity of the fastest general-purpose algorithm. A special-purpose algorithm can usually factor an integer faster then general-purpose one, when certain conditions are met. On the other hand, the set of integers for which these conditions are met, even if infinite, is usually sparse. Thus, it is possible to avoid the use of such numbers when selecting cryptosystem parameters. When a new special-purpose algorithm is found, we can change the parameters of cryptosystems or protocols affected. On the other hand, a new general-purpose factoring algorithm with lower asymptotic complexity can lead even to the total break of all cryptosystems based on the hardness of the problem.

Similar remarks apply to the discrete logarithm problem (DLP). DLP can be defined as follows: Given a prime p, and two numbers a, b such that $a \equiv b^x \pmod{p}$, where x is an integer, find x. Modular exponentiation is fast, so when we are able to find parameters such that DLP is difficult, we can build asymmetric cryptosystems based on DLP. It is more difficult to divide algorithms for solving DLP to general-purpose and special-purpose ones, as DLP definition can be extended to any finite cyclic group and thus also the level of generality.

The IFP and DLP are clearly quite different. Thus it can be surprising that the Number Field Sieve is nowadays the fastest general purpose algorithm to solve both IFP and DLP. Clearly, the details of the algorithm differ depending on the area of application, but the general principle and asymptotic complexity is essentially the same. On the other hand, we can view the Number Field Sieve not as a single algorithm, but as a class of algorithms, or a generic method. As such, all descriptions of the NFS hide many of actual implementation details and leave a lot of space to apply different heuristics. It

is conjectured that these heuristics do not influence the asymptotic running time. The opposite is true for the actual running time. A careful tuning of parameters, and the utilization of implementation heuristics can save a lot of computational resources.

My first research experience with subexponential methods to solve IFP and DLP started more than 5 years ago, as a preparation for my master thesis. The topic of my master thesis switched to Elliptic Curve Cryptography, but the interest in NFS remained. Thus it was natural to consider it as a topic of my PhD study and research. In the time of the choice (2004), a lot of progress had been done on the field of DLP in $\mathbb{F}_p$, such as [**52**]. In that time however, it was not known whether NFS, or some method with similar complexity, can be practically applied to a DLP in field extensions $\mathbb{F}_{p^n}$, with medium sized characteristic p.

One of the practical motivations to solve the DLP in these fields is inspired by XTR based cryptosystems [**70, 63**]. XTR uses a subgroup of $\mathbb{F}_{p^6}^*$ with prime order q dividing $p^2 - p + 1$ (called an XTR group). Its elements are represented by their traces, and the efficient arithmetic is developed to allow a fast exponentiation. XTR-DL problem is to find the unknown exponent d from $Q = G^d$, where G, Q are known XTR traces.

The XTR-DL problem can be solved in XTR group by generic methods with asymptotic complexity $O(q^{1/2})$. If q is chosen as large as possible, i.e. p^2-p+1, then the complexity becomes $O(p)$. Clearly, the computation becomes infeasible very fast with growing p. On the other hand, XTR-DL can be transformed (in a polynomial time) to an instance of the DLP in the finite field $\mathbb{F}_{p^6}$. Our goal was to apply the NFS algorithm to solve this problem, and subsequently the original XTR-DL problem. Meanwhile, a new algorithm [**55**] was published, which presumably can solve the DLP in any finite field. However, we have not seen any practical results for fields of degree six.

Most of the problems in the application of NFS arise from the fact that fields of degree six must use NFS with two degree 6 polynomials (with joint degree 12). The size of the polynomial is too large for any p in a practical range for experiments. Optimal degree of the (single) NFS polynomial is

$$d = \left(3^{1/3} + o(1)\right) \left(\frac{\ln x}{\ln \ln x}\right)^{1/3},$$

where x is the field size. This means $d = 6$ for degree six fields with characteristic of more than 100-bits, which is outside of scope of our computational resources.

The research summarized in this thesis has accomplished at least these basic goals:

- a practical implementation of the NFS was created that can solve DLP in degree six finite fields, and subsequently the XTR-DL problem;
- the three and higher dimensional sieving was explored;
- many experiments were concluded to provide a basis for selecting optimal parameters and heuristics in the NFS implementation;
- areas explored in this thesis can have consequences in other areas of NFS applications;
- DLP in a field $\mathbb{F}_{p^6}$ with a 40-bit characteristic p (240-bit field size) was solved.

Unlike most of the pure mathematical research, this thesis place heavy emphasis on experiments. Unfortunately, the experimental area of Applied Mathematics is a somewhat unclear territory, and it is even rejected by some of the pure mathematicians. However, in topics covered by this thesis a theoretical approach is complicated by three basic problems. The first one is a very broad fundamental theory required, covering algebraic number theory, probability, algorithm complexity, etc. The second one is that the NFS algorithm (and related problems) is quite well-researched from the theoretical point of view, with gaps related to very difficult areas of mathematics like the Riemann hypothesis. The third one is that a lot of theoretical results are probabilistic and/or using some specific conditions, leading to sometimes unexpected effects in practice. Thus our approach is to use the known (or new/conjectured) theoretical results, and explore the applications in the experimental manner. Our experimental results can later generate further theoretical research, especially if some of the results are unexpected, or not completely covered by the theory. Another important aspect of the experimental research is that the results can be understood and further utilized by a broader audience of technical experts, especially in cryptography.

Structure of this thesis is as follows: Chapter 2 contains some of the specific mathematical preliminaries. These are mainly in forms of specific remarks connected to further work, as there is already an extensive amount of literature covering the theoretical side of the work. Chapter 3 provides a state of the art background on the topic of the discrete logarithm problem, basic algorithms for the solution of the DLP, and its applications in cryptography. Parts of this chapter were published in [**106**]. Chapter 4 provides a state of the art on XTR and its applications, and provides the background on transformation of XTR-DL problem to DLP in degree six finite field.

Chapter 5 gives the basic background on the Number Field Sieve. First part of the chapter provides the overview of the method. In the next part, a short mathematical background on the method is provided, along with an algorithm description from the practical point of view.

Chapter 6 contains a discussion of NFS complexity and its parametrization. Basic complexity estimates for applications of NFS to $\mathbb{F}_{p^6}$-DLP are elaborated. Various applicable NFS variants and different implementation options are summarized and discussed. Core of the discussion is devoted to the selection of a sieve polynomial in $\mathbb{F}_{p^6}$-DLP case.

Chapter 7 covers topics connected with sieving. Background information on sieving is provided, along with NFS specific implementation remarks. Our older [**113**], and newer [**115**] sieve algorithm is presented, along with implementation recommendations and basic results. Chapter contains remarks on some of the sieve-specific topics, like the logarithmic estimates of norms, the effective line sieving and the use of the large primes. A new algorithm using large primes is proposed, but it is not elaborated in more details, as its usefulness for our immediate goal was limited.

Chapter 8 contains detailed experimental results. Special notice is given to an influence of a sieve polynomial on the smoothness density. Some of the results were published in the article [**116**]. Section 8.2 contains discussion and experimental results concerning the choice of the optimal sieve region. It provides supportive arguments for the use of 3D sieve in case of using NFS to solve DLP in $\mathbb{F}_{p^6}$. Section 8.3 examines the

behavior of a randomized version of the sieve, and the impact of the tolerance parameter on the number of equations obtained from the sieve and the speed of the sieve. Recommendations for optimal sieve parameters based on the experimental results are provided. Section 8.4 contains the final experimental results (partly presented in [**114**]). The largest computational experiment conducted was the solution of the DLP in $\mathbb{F}_{p^6}$ with 40-bit characteristic p. The total time to find an XTR-DL in the corresponding XTR group, is much faster than if Pollard's rho algorithm (or similar group-size based methods) was used. Sieving results are summarized and extrapolated to provide initial estimates for larger fields.

Selected parts of this work were previously published (or accepted for publication) in [**106, 113, 115, 116, 114**]. Our research was partially supported by grants VEGA 1/3115/06 and ESF SORO/JPD3-038/2005, and by the National Security Authority of Slovak Republic. Expert suggestions and comments of the thesis supervisor Professor Otokar Grošek, and of Professor Ladislav Satko contributed to the theoretical aspects of the project. Realization part was eased by the help of Mgr. Marek Sýs, Ing. Vladislav Novák, and Ing. Matúš Jókay. Author would also like to thank Professor Igor Semaev for important comments and suggestions leading to significant improvements in the quality of this work.

CHAPTER 2

Preliminaries

The understanding of the Number Field Sieve algorithm requires mathematical preliminaries from many parts of mathematics, especially from Number Theory (both Algebraic and Analytic). Effective implementation of the NFS requires also a particular insight into computer science related topics, such as cache friendly algorithm implementation, distributed computing, etc. In this chapter we focus on some basic mathematical preliminaries related to NFS.

Throughout the work, we use standard notions of groups, fields and algebraic numbers. We refer the reader to Chapter 2 of [**77**] for a short summary of the group and field related topics. For a finite field reference, we recommend the book [**73**], and for algebraic number theory, especially from the algorithmic side, we recommend the book [**23**].

The Number Field Sieve from algebraic and implementation point of view is described in Chapter 5. In Section 2.1, we focus on some practical aspects of the required algebraic number theory. Although the algebraic number theory is needed for a correct implementation of the NFS algorithm, most of the results in this work can be understood with only a basic knowledge. In Section 2.2 we further summarize preliminaries from analytic number theory, concerning the smooth numbers. They are used in estimating the asymptotic complexity of NFS, and in the reasoning behind some NFS implementation options.

2.1. Remarks to algebraic numbers and ideals

Let K denote an algebraic number field $\mathbb{Q}(\alpha)$, with α a root of a monic irreducible polynomial f of degree d. Ring of algebraic integers of K will be denoted $\mathcal{O}_K$. Algebraic integers in general do not have unique factorization. Thus ideals of $\mathcal{O}_K$ are used instead. More specifically, we usually work in module $\mathbb{Z}[\alpha]$ only. Points $(a_0, a_1, \ldots, a_{d-1}), a_i \in \mathbb{Z}$ either represent an algebraic integer $\xi = \sum_{i=0}^{d} a_i \alpha^i$, or the principal ideal $(\xi) = \xi\mathcal{O}_K$. This ideal is then factored to prime ideals. In practice however, only the norm $N(\xi)$ is factored. If $p_i | N(\xi)$, the corresponding ideal $\mathfrak{p}_i$ is determined by a common root of $f(x)$ and $a(x) = \sum_{i=0}^{d} a_i x^i$ modulo p_i. If index $[\mathcal{O}_K : \mathbb{Z}[\alpha]] > 1$, a special care must be taken for ideals over primes dividing the index. To simplify the implementation, we can avoid such fields altogether.

For a sake of simplicity, let us consider only unramified prime ideals. Classically, ideals are processed in the two-element representation. If ideal $\mathfrak{p}$ corresponds to a monic irreducible factor $r(x) | f(x) \mod p$, then its two element representation is $(p, r(\alpha))$. Degree of the ideal is $\deg r$, and its norm is $p^{\deg r}$. Ideal $\mathfrak{p}$ can also be seen as a submodule P of $\mathbb{Z}[\alpha]$ with known basis. All elements of P have norm divisible by the

norm of the ideal $\mathfrak{p}$. Let $r(x) = x^s + \sum_{i=1}^{s-1} r_i x^i$. Then the basis of P is (basis vectors in rows)

$$\mathcal{L}(P) = \begin{pmatrix} p & 0 & \cdots & 0 & 0 & \cdots & 0 \\ 0 & p & \cdots & 0 & 0 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\ r_0 & r_1 & \cdots & 1 & 0 & \cdots & 0 \\ 0 & r_0 & \cdots & r_{s-1} & 1 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & r_{d-s} & r_{d-s+1} & \cdots & 1 \end{pmatrix}$$

There are other useful basis representations, such as Hermite Normal Form (HNF, see section 2.4.2 of [**23**]) or LLL-reduced basis [**66**]. Using a suitable basis of ideal, we can quickly identify all elements of a bounded subspace of $\mathbb{Z}[\alpha]$ belonging to a given ideal.

Module representation of an ideal is also used in the lattice sieving [**86**]. One ideal $\mathfrak{q}$ with large norm q is fixed, and then instead of sieving module $\mathbb{Z}[\alpha]$, we sieve module $\mathfrak{q}$ with smaller ideals. Afterwards we choose different $\mathfrak{q}$. Pollard in [**86**] points out that by optimal application of this method it is possible to find 83 % of NFS equations for only 8.6 % of work. However, this estimate was done, if the sieve space was two dimensional. As we show further, to solve XTR-DL problem we require three dimensional sieve. The lattice sieve adaptation to three dimensions have specific problems, such as lattice reduction required for computing bases in different $\mathfrak{q}$-modules. Still the lattice sieve is useful for computing individual discrete logarithms with descent method (see Section 5.2.4). Further notes on sieving are summarized in Chapter 7.

2.2. Smooth numbers and smoothness probability

The Number Field Sieve algorithm is based on smooth numbers. For some well-chosen bound B we are able to find particular numbers (norms of algebraic integers) with prime divisors not exceeding B. We call these integers B-smooth. Their number is larger than the number of primes below B. Moreover, it is fast to factor integers, when we know (a-priori) that they are smooth. Using the sieve, we can also identify smooth integers from some (large) set of integers. The effectiveness of the NFS then depends on the smoothness probability among random or specifically constructed integers.

2.2.1. Smooth integers. A positive integer $n \in \mathbb{N}$ can be written uniquely as a product of prime powers $n = \prod_{i=1}^{c} p_i^{e_i}$, $e_i > 0$, $p_i > p_j$ if $i < j$, p_i prime. Thus for any n we can define function:

$$\mathrm{pf} : \mathbb{N} \times \mathbb{N} \to \mathbb{N}, \quad \mathrm{pf}(n,i) = \begin{cases} p_i, & \text{if } i \le c; \\ 1, & \text{otherwise.} \end{cases}, \tag{2.1}$$

where c is the number of prime factors of n. Function pf returns i-th prime factor ordered from the largest prime factor.

Another useful function is a valuation function $\mathrm{val} : \mathbb{N} \times \mathbb{N} \to \mathbb{N}$,

$$\mathrm{val}(n,p) = \max\{e \in \mathbb{N}_0 | n \equiv 0 \pmod{p^e}\}. \tag{2.2}$$

DEFINITION 2.2.1. Let $n \in \mathbb{Z}$ and let $B > 0$. If $\mathrm{pf}(|n|, 1) \leq B$, then n is called a B-smooth number.

DEFINITION 2.2.2. Let $n \in \mathbb{Z}$ and let $0 < B < B_1$. If $\mathrm{pf}(|n|, 1) \leq B_1$, and $\mathrm{pf}(|n|, 2) \leq B$, then n is called a (B_1, B)-semismooth number.

Smooth and semismooth numbers play an important role in the NFS algorithm. It is easy to factor B-smooth integers, when B is small, and equivalently to compute valuations $\mathrm{val}(n, p)$ for $p < B$. In many IFP/DLP solving algorithms, we must find more B-smooth integers than the number of primes less than B. After factoring and/or computing valuations, it is then possible to write a system of equations leading to a solution of original IFP/DLP.

DEFINITION 2.2.3. Let $n \in \mathbb{Z}$ and let $B > 0$. Let there exist such k, that $\mathrm{pf}(|n|, i) > B$, for each $i < k$, and $\mathrm{pf}(|n|, i) \leq B$, for each $i \geq k$. Then $\prod_{i=k}^{c} \mathrm{pf}(|n|, i)^{e_i}$ is called a B-smooth part of n. Primes $\mathrm{pf}(|n|, i)$ for $i = 1, \ldots, k-1$, are called large (prime) factors of n.

2.2.2. Smoothness probability. The number of smooth numbers and their distribution is important in the estimates of the NFS complexity. The number of y-smooth numbers in the interval $[1, x]$ is given by the function $\Psi(x, y)$. There are many analytic and numeric results concerning computation of function Ψ. Good overview and a practical method for computing tight bounds for Ψ is presented by Bernstein in [**11**]. An important result for the NFS complexity is the asymptotic behavior of Ψ. Dickman in [**30**] observed that

$$\lim_{y \to \infty} \frac{\Psi(y^u, y)}{y^u} = \rho(u), \text{ for } u > 0. \tag{2.3}$$

Here ρ is a continues function satisfying $\rho(u) = 1$ for $0 < u \leq 1$ and $u\rho(u) = \int_{u-1}^{u} \rho(t)dt$ for $u > 1$. A first approximation is $\rho(u) \approx u^{-u}$. A more general results for semismooth integers can be found in [**9**].

The number $\Psi(x, y)/x$ denotes a probability that randomly selected integer from the interval $[1, x]$ (with uniform distribution) is y-smooth. We will call a probability that a randomly selected integer from the set M is y-smooth a smoothness probability (with respect to y and M).

The smoothness probability depends on the smoothness bound and the size of the numbers involved. In the NFS context a family of functions is used:

$$L_x(\alpha, c) = \exp\left(c(\ln x)^{\alpha}(\ln \ln x)^{(1-\alpha)}\right), \tag{2.4}$$

where $\alpha, c \in \mathbb{R}$, $0 < \alpha \leq 1$, $c > 0$. It follows from [**21, 29**] that a random positive integer $\leq L_x(\alpha_1, c_1)$ is $L_x(\alpha_2, c_2)$-smooth with probability

$$L_x\left(\alpha_1 - \alpha_2, -\frac{c_1}{c_2}(\alpha_1 - \alpha_2) + o(1)\right) \tag{2.5}$$

for $x \to \infty$. Asymptotic complexity of the NFS is based on this estimate [**61**].

2.2.3. Smooth algebraic integers. Let α be a root of polynomial $f \in \mathbb{Z}[x]$ irreducible over $\mathbb{Z}$. Then $K = \mathbb{Q}(\alpha)$ is an algebraic number field. The ring of integers of K will be denoted $\mathcal{O}_K$.

Let $\nu \in \mathcal{O}_K$. In general, ν cannot be uniquely factored in $\mathcal{O}_K$, which complicate the notion of smooth algebraic integer. Instead of factoring algebraic numbers directly, we switch to ideals of $\mathcal{O}_K$.

An algebraic number $\nu \in \mathcal{O}_K$ defines the principal ideal $(\nu) = \nu\mathcal{O}_K$. As an ideal of $\mathcal{O}_K$, it can be factored uniquely (up to permutation) into a product of prime ideals

$$(\nu) = \prod_{i=1}^{c} \mathfrak{p}_i^{e_i}, \quad \text{with } e_i > 0, N(\mathfrak{p}_i) \geq N(\mathfrak{p}_j) \text{ for } i < j.$$

We can define functions pf and val for algebraic numbers/pricipal ideals and prime ideals in similar manner like in Equations (2.1) and (2.2). The order of prime ideals is not unique, since there are more prime ideals with the same norm, i.e. $N(\mathfrak{p}_i) = N(\mathfrak{p}_j)$ for some i, j. Thus we must choose some arbitrary order of prime ideals in the definition of function pf.

Definition 2.2.4. We say that an algebraic integer $\nu \in K$ is B-smooth, if the corresponding ideal decomposition of principal ideal (ν) has $N(\mathfrak{p}_1) \leq B$.

In algebraic number fields, the absolute value of the norm of ν is the same as the norm of principal ideal (ν). We denote it just by $N(\nu) \in \mathbb{Z}_{>0}$. Thus we can find unique factorization $N(\nu) = \prod_{i=1}^{c} p_i^{e_i}$, with factors in descending order from the largest factor p_1.

If ν is B-smooth, then clearly its norm is also a B-smooth integer. The converse is not true in general, due to higher degree prime ideals. E.g. if a prime ideal $\mathfrak{p}$ of degree 2 lies over prime $p > \sqrt{B}$, then its norm $N(\mathfrak{p}) = p^2 > B$.

In the NFS algorithm it is customary to use only degree 1 ideals, and thus the smoothness of algebraic number ν is equivalent to smoothness of its (absolute value of) norm $N(\nu)$. From the practical point of view, the smoothness of $N(\nu)$ is easier to evaluate than smoothness in sense of Definition 2.2.4. Thus in practical situations we consider all algebraic numbers with smooth norm to be smooth.

In algebraic number fields we can also define the notion of smoothness probability.

Definition 2.2.5. Let M be a set of algebraic integers, $M \subset K$. The probability that randomly chosen $\nu \in M$ is B-smooth is called a smoothness probability with respect to M, and B respectively.

Smoothness probability can be estimated by using either Equation (2.3) or (2.5), after we have suitable upper bound on norms of algebraic integers from M. More details on smoothness probability in number fields of degree 6 is provided in Chapter 6.

CHAPTER 3

Discrete logarithm problem and its applications in cryptography

3.1. Definition of the discrete logarithm

A classical definition of the discrete logarithm problem (DLP), and the notion of discrete logarithm (DL) originate in number theory. DLP can be simply stated as follows: Given three integers a, b, n find the (least non-negative) integer x, such that $a = b^x \pmod{n}$. If such an integer exists, we call it the discrete logarithm of a w.r.t. base b modulo n, denoted by $\log_{b,n} a$.

DLP was later generalized in group theory. Every finite cyclic group with n elements is isomorphic to the additive group $\mathbb{Z}_n$. Clearly, DLP can be transformed to a problem of finding isomorphism between $\mathbb{Z}_n^*$ and $\mathbb{Z}_{\varphi(n)}$.

Definition 3.1.1. Let $(G, \cdot)$ be an arbitrary finite cyclic group of order n. Let α be the generator of G. The *generalized discrete logarithm problem* (GDLP) is the following: Given an element $\beta \in G$, find the unique integer x, $0 \leq x < n$, such that $\alpha^x = \beta$. The integer x is called the discrete logarithm of β to the base α, denoted by $x = \log_\alpha \beta$.

In this case GDLP is equivalent to finding an isomorphism between any finite cyclic group G and $\mathbb{Z}_{\operatorname{ord} G}$. Thereafter, if not specified, we always assume the generalized discrete logarithm problem, and shortly write DLP. A subgroup S of a finite cyclic group G is also a cyclic group. Thus we can find discrete logarithms to the base given by a generator of S. It is well known that the properties of the function $\log_\alpha(x)$ in a group of order n are similar to the properties of the logarithmic function for real-numbers:

$$\log_\alpha xy = (\log_\alpha x) + (\log_\alpha y) \pmod{n}, \tag{3.1}$$

$$\log_\alpha x^z = z(\log_\alpha x) \pmod{n}, \tag{3.2}$$

where $x, y \in \langle\alpha\rangle$ and z is an arbitrary integer. If γ is an arbitrary generator of G, we can write

$$\log_\gamma \alpha(\log_\alpha \beta) = \log_\gamma \beta \pmod{n}. \tag{3.3}$$

If we are able to compute both discrete logarithms of α, β in base γ, we can compute $\log_\alpha \beta$ as a solution of congruence (3.3). If β belongs to a subgroup $S \subset G$ generated by α, the logarithm can always be computed. We do not even need to know the value of γ, as long as both logarithms are in the same base. In this case, γ is omitted from the logarithm notation. Thus $\log \alpha$ simply means a discrete logarithm of α to some fixed, but not specified base. Also, even if not explicitly specified, discrete logarithms always represent modulo n classes.

In the case of a finite field $\mathbb{F}_{p^n}$, the multiplicative group is cyclic. Thus we can compute a discrete logarithm of any non-zero field element w.r.t. some generator g of $\mathbb{F}^*_{p^n}$. Definition 3.1.1 also allows to take some additive cyclic subgroup of $\mathbb{F}_{p^n}$, and to compute discrete logarithms in this group. E.g. for group $(\mathbb{F}_p, +)$ this leads to solve the equation $xa = b \mod p$, with unknown x. This problem is however easy to solve.[1] Thus in the sequel, the notion discrete logarithm in a finite field refers always to discrete logarithms in the multiplicative group of a finite field.

3.2. Overview of the methods to find discrete logarithms

Methods to solve the discrete logarithm problem can be divided into two main classes. Group-independent methods work for any finite cyclic group. Group-dependent methods can only be applied to certain types of groups.

The complexity of group-independent methods is determined by the factorization of the group order. If the factorization of the group order n is known, it is possible to compute discrete logarithms in subgroups of prime order. Using Chinese Remainder Theorem (CRT) we finally compute discrete logarithm modulo n. This is the main idea of the so-called Pohlig-Hellman method, presented in section 3.2.3.

Even if the order of the group is unknown, it is still possible to compute discrete logarithms (and group order). In this case we should be able to estimate bounds of the interval in which we expect the actual logarithms. Complexity of all known group-independent methods is exponential (in number of bits of n, resp. the maximal of $p_i|n$).

An example of group-dependent methods is the application of Extended Euclid algorithm to compute the solution of $xa = b \mod p$, and thus effectively solve DLP in $(\mathbb{F}_p, +)$. This algorithm has polynomial complexity (in number of bits of p). However, no polynomial algorithm for DLP solution is known to exist in multiplicative subgroup of a finite field. Still, the DLP can be solved with subexponential complexity in a full multiplicative group of any finite field. The complexity of these methods depend on the size of the whole field, even if we try to solve discrete logarithms in its (relatively) small subgroup. Different complexity estimates, when comparing a problem from the subgroup point of view, and from the field point, can be used for a better selection of parameters for DL-based cryptosystem [**96**]. This is also one of the key ingredients used by XTR, as shown in Chapter 4.

It is important to know possible methods of solving the DLP, and their complexity estimates, to assess the security of DLP-based cryptosystems. The security of system can be improved by increasing the size of the keys, resulting in a larger group size q and/or field size p^n. On the other hand, the actual implementation is then more expensive. Given the complexity estimate, we can trade off key-sizes and still the system cannot be broken by solving the DLP (according to the current knowledge).

[1] This is not true for additive groups in general, e.g. it is difficult to solve DLP in a group of points of some elliptic curves.

Increasing the key size of the system is only reasonable, if there is no polynomial time algorithm for solving the underlying DLP. The question of existence of such algorithm in general, and in multiplicative groups of $\mathbb{F}_{p^n}$ is still open, although it is widely believed that no such algorithm exists on conventional computers. On the other hand, Shor [**103**] has published a polynomial time algorithm for solving the DLP on the quantum computer. From the practical point of view, this algorithm is unusable for breaking the currently used DLP-based cryptosystems, as we are unable to construct quantum computer with quantum registers large enough to perform the actual computation.

3.2.1. Shanks algorithm. Shanks [**102**] (see also [**77**], section 3.6.2) proposed an algorithm for solving the DLP that is based on time-memory trade-off in exhaustive search. It is also called *baby-step giant-step algorithm*. Its idea comes from the following observation. When $\beta = \alpha^x$ and the order of α is n, then we can write $x = im + j$, where $m = \lceil \sqrt{n} \rceil$, $0 \leq i, j < m$. Then $\alpha^x = \alpha^{im}\alpha^j$, giving us $\beta(\alpha^{-m})^i = \alpha^j$.

INPUT : $\alpha \in G$ having order n, $\beta \in G$
OUTPUT: $x = \log_\alpha \beta$

---

(1) LET $m \leftarrow \lceil \sqrt{n} \rceil$.
(2) Create table of $T = \langle (j, \alpha^j) \rangle$ FOR $j = 0, 1, \ldots, m-1$ (baby-step).
(3) LET $\gamma \leftarrow \beta$.
(4) FOR $i = 0 \ldots m-1$:
    (a) IF (j, γ) is in table T RETURN $x = im + j$.
    (b) ELSE $\gamma \leftarrow \gamma \cdot \alpha^{-m}$ (giant-step).

This algorithm has both time complexity and memory demands bounded by $O(\sqrt{n})$. The trade-off between time and memory can be further refined by another choice of m in step (1). By another choice of m we can also speed up the algorithm, if we know the interval in which to search for the x. In this case, it is also possible to compute discrete logarithms, even if do not know the exact size of the group.

3.2.2. Collision methods. Pollard's ρ- and λ-methods represent another class of exponential algorithms for solving the GDLP [**87**]. Algorithms of this class are based on finding "collisions" in one or more pseudorandom sequences. They are non-deterministic in nature, but yield good practical results.

Pollard's ρ-method uses one pseudo-randomly iterated sequence in the underlying finite group. As the number of points is finite, the sequence becomes cyclic in some step. At this point, we can compute the discrete logarithm. The name of the method comes from the visualization of this process, which looks like the Greek letter ρ.

Next we give more details of this method. We are searching for $\log_\alpha \beta$ where $\alpha \in G$ is an element of order n. We form a sequence $y_1, y_2, \ldots$ of triples $y_i = (x_i, a_i, b_i)$ by iteratively applying function $f : \langle \alpha \rangle \times Z_n \times Z_n \to \langle \alpha \rangle \times Z_n \times Z_n$,

$$f(x, a, b) = (\alpha^k \beta^l x^m, ma + k, mb + l)$$

where $k, l, m \in Z_n$ are given by x belonging to a particular partition S_j of G (all partitions should have roughly equal size and it should be easy to verify to which partition x belongs to). Function f satisfies the property

$$x_i = \alpha^{a_i}\beta^{b_i} \Rightarrow x_{i+1} = \alpha^{a_{i+1}}\beta^{b_{i+1}}.$$

We know the single point $(1, 0, 0)$ which satisfies this property (if we knew another one we could compute DL directly). This is a starting point of the sequence. When we find the cycle, we can find y_s, y_t, such that their first coordinates are the same (i.e $x_s = x_t$). Then

$$\alpha^{a_s}\beta^{b_s} = x_s = x_t = \alpha^{a_t}\beta^{b_t}.$$

From this $\alpha^{a_s - a_t} = \beta^{b_s - b_t}$. Now if $\gcd(b_s - b_t, n) = 1$ (which has a high probability), we can compute $log_\alpha \beta = (b_s - b_t)^{-1}(a_s - a_t) \mod n$.

A computer implementation of the ρ-method is very simple. It is usually implemented using Floyd's cycle finding algorithm [**36**]. It does not require any substantial memory overhead, since we only need to store two triples y_i and y_{2i}. An alternative faster cycle detection is due Brent [**17**]. Time complexity estimates of the ρ-method are bounded by $O(\sqrt{n})$.

Pollard's λ-method uses collisions of two pseudo-random sequences (sometimes called kangaroos). One of them is a reference sequence ("tame kangaroo"), starting from known point $t_0 = \alpha^s$ and second one ("wild kangaroo") starts from a point $w_0 = \beta = \alpha^x$, where x is the unknown logarithm.

The sequences are iterated using the pseudorandom function a. This gives us two sequences $t_{i+1} = t_i \cdot \alpha^{a(t_i)}$ and $w_{i+1} = w_i \cdot \alpha^{a(w_i)}$ respectively. There is a high probability (due to birthday paradox) that these sequences will collide, i.e. $t_i = w_{i'}$. The visualization of the sequences looks like the Greek letter λ.

In the point of collision we can compute x from the equation

$$s + \delta_{t_i} = x + \delta_{w_{i'}} \pmod n,$$

where δ is remembered distance of the sequence $\delta_{t_{i+1}} = \delta_{t_i} + a(t_i)$. To be able to detect the collision we must store the points of the sequences. To save the memory, we store only some distinguished points (easy to check due to some property) and do not check the whole sequences.

This method can be modified for finding discrete logarithms in a given interval, i.e. $x \in [a, b]$, where a and b are known, even if the order n is unknown. λ-method can be run with time complexity $2\sqrt{b-a} + O(\log n)$, if we know that $x \in [a, b]$. The required memory storage is small, and the computing can be implemented in more parallel computers with almost linear speedup.

3.2.3. Pohlig–Hellman algorithm. Algorithm proposed by Pohlig and Hellman [**85**] can lower the complexity bound of general DLP solving methods, when the factorization of the order of α is known. When order $n = {p_1}^{e_1}{p_2}^{e_2} \cdots {p_n}^{e_n}$, then we can compute single $x_i = \log_\alpha \beta \mod p_i^{e_i}$.

We find x_i's using the following algorithm:

INPUT : G, n, α, β, prime p, exponent e
OUTPUT: $\log_\alpha \beta \mod p^e$

---

(1) $j \leftarrow 0$.
(2) $\beta_j \leftarrow \beta$
(3) WHILE $j \leq e-1$
    (a) $\delta \leftarrow \beta_j^{n/p^{j+1}}$
    (b) FIND k such that $\delta = \alpha^{kn/p}$ (DLP in subgroup)
    (c) $a_j \leftarrow k$
    (d) $\beta_{j+1} \leftarrow \beta_j \alpha^{-a_j p^j}$.
    (e) $j \leftarrow j+1$
(4) RETURN $x_i = \sum_j^{e_i-1} a_j p^j$.

After computing x_i's we can combine the partial results with Chinese remainder theorem to $x = \log_\alpha \beta$. Computation of x_i's is a less complex problem, as we solve the DLP in subgroups with a smaller order p_i.

Overall complexity of the algorithm is given by the complexity of DLP in the subgroup of order q, where q is the largest prime factor of n. We can use an estimate $O(\sqrt{q})$ as a good approximation of real complexity. This algorithm has important practical consequences. In a cryptographic application we require that it is hard to solve the DLP. Hence the order n of the cyclic group must have a large prime factor.

3.2.4. Index calculus methods. The former algorithms did not impose any specific restriction on the group G in which we tried to solve the DLP. On the other side, we can apply index calculus (IC) methods only for solving the DLP in full multiplicative group of finite fields $\mathbb{F}_{p^n}$. We can transform some of the generalized DLP in other groups to DLP in multiplicative group of finite fields, but usually we get fields too large for IC methods to be effective.

Index calculus methods are sometimes called factor base methods, by the most significant step in the algorithms. These methods are similar to that used for factorization (QS, NFS). In general the algorithms based on IC methods have subexponential complexity, with constants dependent on the field and heuristics used.

The main idea of IC methods is as follows: Choose an appropriate factor base $S = \{p_1, \ldots, p_k\}$, such that most of the group elements of G can be factored over S. IC based algorithms first find discrete logarithms of elements in S, and these can be then used to compute discrete logarithms in G.

3.2.4.1. *IC methods in* $\mathbb{F}_p$. IC methods work for arbitrary field, but we describe only how it works in a prime field $\mathbb{F}_p$. Let g be a generator of $\mathbb{F}_p^*$ and let $t \in \mathbb{F}_p^*$. We are looking for integer l, such that $g^l \equiv t \pmod p$, or in other words we are looking for $l = \log_g t$.

We can write the general index-calculus method in $\mathbb{F}_p^*$ in the following steps:

INPUT : p prime, g generator of Z_p^*, $t \in Z_p^*$
OUTPUT: $l = \log_g t$

---

(1) *Choose the smoothness bound* B and create the factor base $S = \{p_i | p_i \leq B, p_i \text{ is prime}\}$.
(2) *Search for relations.*
Choose random number r, $1 \leq r < p-1$. If we can factor $g^r \mod p$ over S, then we can write an equation

$$r \equiv r_1 \log_g p_1 + \cdots + r_k \log_g p_k \pmod{p-1},$$

where $\log_g p_1, \cdots, \log_g p_k$ are unknowns. Store equations in the growing system of equations repeating the step until enough equations are found for system to be solvable.
(3) *Linear algebra.*
Solve the system of equations from previous step and compute unknowns $\log_g p_1, \ldots, \log_g p_k$.
(4) *Find the discrete logarithm.*
Choose random integers R, $1 \leq R < p-1$, until the number $g^R t \mod p$ can be factored over S. Using equation

$$log_g t \equiv -R + \tau_1 \log_g p_1 + \cdots + \tau_k \log_g p_k \pmod{p-1}$$

and values computed in previous step we can find the actual $\log_g t$.

If the parameters are properly chosen (according to asymptotic estimates based on smoothness probability), then the most time consuming step is the search for relations. On the other hand, this task can be easily distributed to many computers, unlike the linear algebra step. Complexity of these two steps can be to a small degree balanced by a careful choice of parameters, especially the smoothness bound B. Once we have found the discrete logarithms of the factor base, only the last step is required to find any other discrete logarithm in $\mathbb{F}_p^*$.

The above algorithm is only a scheme that requires many implementation details, like the choice of a factor base, factoring technique used, how to search for equations fast or how to solve linear algebra.

The complexity of the algorithms is bound with the choice of factor base. Asymptotically optimal factor base size is given as $L_p(1/2, c + o(1))$, where c is a constant, and L is defined by equation (2.4). Constant c is mainly influenced by the factorization methods used in the second step. If we use fast smoothness testing, overall complexity can be written as $L_p\left(1/2, \sqrt{2}\right)$. If we use trial division, complexity will rise up to $L_p(1/2, 2)$.

3.2.4.2. *Evolution of IC methods.* Index calculus methods in general number fields are based on Adleman's algorithm [**1**]. A specialized application for fields of characteristic 2 is described by Coppersmith in [**25**], later improved and implemented by Gordon and McCurley [**47**]. This method is also called a polynomial sieve, generalized version

is due to Gao and Howell [**41**]. Using this algorithm, Thomé [**107**] was able to find discrete logarithms in the field $\mathbb{F}_{2^{607}}$.

Randomized index calculus method presented above is based on the original ideas of Coppersmith, Odlyzko and Schroeppel [**27**]. These methods were superseded in $\mathbb{F}_p$ by the application of the Number Field Sieve method [**46**]. NFS decreases the asymptotic complexity to $L_p(1/3, c)$ by using algebraic numbers and sieve methods instead of randomized checking. A record solution was presented by Weber [**109, 110**]. Overview of classical index calculus methods and NFS is also provided in [**95**].

A long time, there was an $L_x(1/3, c)$ algorithm to solve DLP only for fields of large prime characteristic $\mathbb{F}_p$ $(x = p)$, and for fields of very small characteristic, mostly $\mathbb{F}_{2^n}$ $(x = 2^n)$. Constant c under asymptotic settings for general prime field is $c = (64/9)^{1/3} \doteq 1.9229$ for an original Schirokauer's algorithm [**92**]. This bound was later improved by Matyukhin [**76**] to $c = \left(\frac{96+26\sqrt{13}}{27}\right)^{1/3} \doteq 1.902$. An algorithm for special primes p with much lower constant $c = (32/9)^{1/3} \doteq 1.526$ was presented by Semaev [**98**].

A variant of NFS for fields $\mathbb{F}_p$ that was presented by Joux and Lercier [**52**] is more efficient from the implementation point of view. They have also provided a method to replace Coppersmith's algorithm for fields of small characteristic in [**54**], repeating and improving the previous record of [**107**]. Together with results of [**55**], these algorithms provide $L_x(1/3, c)$ algorithm for all finite fields. A variant of [**52**] with multiple polynomials was presented by Commeine and Semaev [**24**], with smaller constant c. Unfortunately, the field sizes with a significant impact of different c's are out of a practical range to compare the methods.

Our work targeting XTR-DL problems require solving discrete logarithms in field $\mathbb{F}_{p^6}$. In this case, algorithm of [**55**] can be applied. Its asymptotical behavior is known, but for the fields, where this algorithm is feasible, some assumptions in complexity estimates are not yet valid (namely the degree of polynomial used). As we show in this work, after solving some specific issues, it is possible to implement this algorithm, and solve the corresponding XTR-discrete logarithms.

3.3. Cryptographic applications of the DLP

In 1976 Whitfield Diffie and Martin Hellman published important article *New directions in cryptography* [**31**]. It was in the time of rapid growth of computing and communication technology. New communication possibilities required stronger cryptography to ensure privacy. This could be accomplished by many new *symmetric* ciphers, e.g. DES that was just in the standardization phase [**80**]. Symmetric cryptography requires that both participants share one secret key K. There has to be a secure way to exchange this key. Contemporary ways, e.g. travel in person or exchange by trusted couriers, were too costly and impractical for general use.

Diffie and Hellman proposed a solution – Diffie-Hellman key agreement scheme – that was based on a concept of *one-way functions*. A function f is a one-way if, for any argument x in the domain of f, it is easy to compute the corresponding value $f(x)$, yet, for almost all y in the range of f, it is *computationally infeasible* to find x such that

$y = f(x)$ for any given y. As we have seen, one of the conjectured one-way functions is exactly the modular exponentiation, under the condition that the corresponding discrete logarithm problem is infeasible.

The first published discrete logarithm based key agreement scheme was Diffie–Hellman key agreement scheme [**31**]. Algorithms and protocols extending the basic scheme are used in many applications, e.g. in SSH authentication. If the exchange is performed using arithmetic of field $\mathbb{F}_p$, both parties must exchange at least $2 \log p$ bits in every session. We suppose that public parameters are precomputed and fixed for many sessions. Thus increasing the size of the parameter p, while increasing security, increases also the communication overhead (as well as complexity of multiplications).

The hardness of computing discrete logarithms can be used also in public-key cryptography. ElGamal's cryptosystem [**33**] is an example of a discrete logarithm based public key encryption system. More important is variant of ElGamal's cryptosystem used for digital signatures, the *Digital Signature Algorithm* (DSA). It is now standardized under the name *Digital Signature Standard* (DSS, FIPS186-2, [**81**]). Parameters of DSA are prescribed by the standards, with the bit-size of the field characteristic p at least 1024 bits. Due to the Pohlig-Hellman attack (see section 3.2.3), we also require that order of the multiplicative group $\mathbb{F}_p^*$ has a large prime factor q. A subgroup of this order is used for generating the signature. The complexity of the signature generation depends on both field parameter p and the size of the used cyclic group q. Bit-size of p determines the speed of a single multiplication or squaring operation. Number of squaring operations in modular exponentiation is given by the bit-size of q. Length of the signature is $2 \log q$. Better estimates on complexity and recommended key sizes can be found in [**83**].

3.4. Motivation for XTR

Cryptographic algorithms based on the intractability of the DLP usually work in a multiplicative group $\mathbb{F}_p^*$, or its suitable subgroup. It is possible to replace this group with another cyclic group, where DLP is a difficult problem. It should be at least as difficult to compute, as in a standardized DSA algorithm. A multiplicative group of $\mathbb{F}_{2^n}$ is often used due to a faster field arithmetic. Nowadays, it is recommended to use the Elliptic Curve Cryptography (ECC) [**82**] with arithmetic operations defined in a group of points of an elliptic curve. However, not all elliptic curves are suitable for cryptography, and the parameter selection is quite tedious. ECC is also one of the reasons to study the discrete logarithm problem in (higher degree) finite fields, as for some curves it is possible to transform an Elliptic Curve DLP to a DLP in a finite field [**78**].

If we consider DSA algorithm, the signature is compressed to two coordinates, with total of $2 \log q$ bits. While field security is affected by the subexponential methods (we have to significantly increase p), group security grows exponentially with the size of q. Thus we want to have a group representation and arithmetic dominated by q instead of p, if possible. As shown in Chapter 4, there is indeed a practical solution, called Efficient and Compact Subgroup Representation (XTR). Field $\mathbb{F}_p$ is replaced by extension field $\mathbb{F}_{p^6}$, but the elements are represented as traces in $\mathbb{F}_{p^2}$. Arithmetic works directly with

this traces in efficient manner. This allows a reduction to a 1/3 of original bit sizes, and even better performance gain (XTR is estimated to be 8-times faster). As we will show further, the experimental evidence indicate that XTR is more secure (in terms of cost function) than classical DL-based system (over $\mathbb{F}_p$, or $\mathbb{F}_{2^n}$) of equivalent field size. A similar comparison works against RSA.

CHAPTER 4

XTR overview

Diffie-Hellman key exchange, DSA, ElGamal encryption, and many cryptographic protocols are based on the exponentiation in the full multiplicative subgroup of a finite field. This is, however, not an optimal solution. As pointed in Section 3.2 the complexity of the generic solution of the discrete logarithm problem for a cyclic group is given by $O(q^{1/2})$, where q is the largest prime factor of the group size. On the other hand, complexity of the DLP in the finite field is also given by $O(L_x(1/3, c))$, where $x = p^d$ is the field size. While DLP in groups with 160-bits q are already well outside the reach of the whole current computing power, DLP in fields with 1024-bit size are believed to be solvable within next 10 years. There is a large discrepancy between secure 160-bits of group size, and almost insecure 1024-bits of field size. Whole multiplicative group with bit-size same as the field size is very inefficient from the performance-to-security point of view.

Schnorr has suggested in [**96**] to use a subgroup of relatively small prime order (160-bits compared to 1024-bits) instead of the full multiplicative group. This leads to speedup in the modular exponentiation, as the exponent is at most $\lceil \log_2 q \rceil$ bits long. Another saving is achieved in the size of the signature and in the verification step. Even if we use only a small subgroup of the finite field, arithmetic operations still work on the whole N-bit representations of field elements. To also compress field element representation and the complexity of arithmetic operations, we can use XTR or another similar systems.

In this chapter we present a basic overview of the XTR. This name is derived from the acronym ECSTR — Efficient and Compact Subgroup Trace Representation. In short, XTR is a system enabling us to achieve security of the field $\mathbb{F}_{p^6}$ by using elements and arithmetics of $\mathbb{F}_{p^2}$. XTR was published by Lenstra and Verheul in [**70**]. It is an evolution of the older system from [**18**], the first system that allowed $1:3$ reduction of discrete logarithm based cryptosystems. Other systems are known with similar properties, like Lucas-based cryptosystems [**16**], cubic field system GH [**45**], or CEILIDH [**91**].

In fact, Rubin and Silverberg have generalized the idea into a concept of Torus-Based Cryptography [**91**]. Torus-based and similar cryptosystems work in field extensions $\mathbb{F}_{p^n}$ using arithmetic on elements of algebraic torus $T_n(\mathbb{F}_p)$ of dimension $\varphi(n)$. These systems have $n \log p$ bits of security when exchanging $\varphi(n) \log p$ bits of information. Thus they are more efficient than classical Diffie-Hellman by a factor of $n/\varphi(n)$.

We must note that XTR is not based on algebraic tori directly, although it can be derived from it [**91**]. XTR has more efficient arithmetic than CEILIDH, but it is not

possible to efficiently implement the multiplication operation (only exponentiation). Furthermore, it is not possible to generalize XTR to higher fields[1].

Torus-based approach leads further to new index-calculus based method to solve DLP directly on algebraic tori [**48**]. This method, however, does not apply to XTR as defined originally over $\mathbb{F}_p$. Thus, the security of XTR is based directly on difficulty of solving discrete logarithm in the field $\mathbb{F}_{p^6}$.

Using [**70, 63**] we now show, how XTR parameters are generated, how modular exponentiation is computed using XTR, what is the security of XTR-based cryptosystem, and how to convert the XTR parameters back to $\mathbb{F}_{p^6}$.

4.1. XTR parameters

The finite field $\mathbb{F}_{p^6}$ can be generated as a field extension in three ways:

(1) As a degree 6 extension of $\mathbb{F}_p$;
(2) As a degree 3 extension of $\mathbb{F}_{p^2}$, with $\mathbb{F}_{p^2}$ being degree 2 extension of $\mathbb{F}_p$;
(3) As a degree 2 extension of $\mathbb{F}_{p^3}$, with $\mathbb{F}_{p^3}$ being degree 3 extension of $\mathbb{F}_p$.

Representation (2) is used by the XTR system, although the real field elements are not represented at all. On the other hand, representation (1) is more useful for discrete logarithm computation. Although we have many possible ways of representing the arithmetic in the finite field $\mathbb{F}_{p^6}$, it is still only a single finite field (using field isomorphism). It is possible to change the representation in polynomial time.

The full multiplicative group of a finite field $\mathbb{F}_{p^6}$ has $p^6 - 1$ elements. We can factor the group size at least into

$$p^6 - 1 = (p-1)(p+1)(p^2+p+1)(p^2-p+1).$$

The subgroup of size $p-1$ is the multiplicative group of the subfield $\mathbb{F}_p$, the subgroup of size $p^2 - 1 = (p-1)(p+1)$ is the multiplicative group of the subfield $\mathbb{F}_{p^2}$, and the subgroup of size $p^3 - 1 = (p-1)(p^2+p+1)$ is the multiplicative group of the subfield $\mathbb{F}_{p^3}$. Subgroup G with order $p^2 - p + 1$ is thus the proper subgroup of $\mathbb{F}^*_{p^6}$, i.e. has no elements from the subfields except neutral element.

Let $q | p^2 - p + 1$, q is prime. Then there exists $g \in G$ generating subgroup $\langle g \rangle \subseteq G$ of order q. Group G is called *XTR supergroup*, and group $\langle g \rangle$ is called *XTR (sub)group*. If $q = p^2 - p + 1$, then these two groups are identical.

Group G was chosen, because it is possible to represent its elements and arithmetic very effectively using traces of elements of G in $\mathbb{F}_{p^2}$. The reason to choose $q < p^2 - p + 1$ is again in efficiency. If a typical field size is around 1024-bits, the characteristic p has about 170 bits. Then $p^2 - p + 1$ has around 340 bits, and we require only 160-bits q, which is much smaller.

In order to be able to use the fast XTR arithmetic described further in Section 4.2, the prime p should be $2 \mod 3$. In this case $p^2 - p + 1 = 0 \mod 3$, thus we always have both XTR subgroup and supergroup. In this case the system is vulnerable to a so

[1] XTR in $\mathbb{F}_{p^{30}}$ would allow us to represent elements of $\mathbb{F}_{p^{30}}$ using only 8 $\mathbb{F}_p$ elements.

called subgroup attack. The attacker forces (or uses fault attack to obtain) signature of an element of XTR supergroup G, which is not an element of XTR subgroup, but of some another subgroup of G with low order s. Then he knows the secret parameter modulo s. Direct attack can be mitigated by group membership verification. However, verification of XTR subgroup membership is considerably slower than verification of XTR supergroup membership.

To render these attacks ineffective, we can either choose q equal to $(p^2 - p + 1)/3$ (with impact to signature size, or using small exponents), or we can use p such that $(p^2 - p + 1)/3$ has only 2 large factors of the similar size (one of them is q). Algorithms for parameter selection used for cryptographic applications are described in detail in the Section 3 of [**63**].

From the cryptanalytic point of view, we are interested in parameters that are weak, and can be used in DLP computation experiments. The DLP in finite fields $\mathbb{F}_p$ with p having 160 decimal digits (ca. 530 bits) have already been computed [**56**], using method of [**52**]. This would give us an expectation that DLP in $\mathbb{F}_{p^6}$, with p having 88 bits is also solvable, even if we chose secure q near to $p^2 - p + 1$. For research purposes, given p, we are interested in cases where q is as high as possible, i.e. $q = (p^2 - p + 1)/3$, and focus on solving the DLP in corresponding field using the Number Field Sieve.

4.2. Trace representation

Let $p \equiv 2 \pmod 3$. Then $x^2 + x + 1$ is irreducible over $\mathbb{F}_p$, and its roots α and α^p form an optimal normal basis for $\mathbb{F}_{p^2}$ over $\mathbb{F}_p$. From $\alpha^i = \alpha^{i \bmod 3}$ it follows that

$$\mathbb{F}_{p^2} = \{x_1\alpha + x_2\alpha^2 : \alpha^2 + \alpha + 1 = 0, x_1, x_2 \in \mathbb{F}_p\}.$$

This representation leads to effective multiplication in $\mathbb{F}_{p^2}$, and to free computation of Frobenius automorphism $x^p = x_1^p\alpha^p + x_2^p\alpha^{2p} = x_2\alpha + x_1\alpha^2$.

Trace is linear map from extension field E into its subfield F. It is computed as a sum of conjugates over F. The conjugates over $\mathbb{F}_{p^2}$ of $h \in \mathbb{F}_{p^6}$ are h, h^{p^2}, and h^{p^4}. Thus

$$Tr_{\mathbb{F}_{p^6}/\mathbb{F}_{p^2}}(h) = h + h^{p^2} + h^{p^4}.$$

Further on, we will not use the subscript in trace notation, and the trace will always be over $\mathbb{F}_{p^2}$.

XTR supergroup G has order $p^2 - p + 1$, thus we can compute conjugates of $g \in G$ as $g^{p^2} = g^{p-1}$, and $g^{p^4} = g^{-p}$, so that $Tr(g) = g + g^{p-1} + g^{-p}$. Norm of element g (in $\mathbb{F}_{p^2}$) is a product of conjugates, thus for every $g \in G$, $N(g) = gg^{p-1}g^{-p} = 1$. Finally, characteristic polynomial of g over $\mathbb{F}_{p^2}$ can be computed as

$$(x - g)(x - g^{p-1})(x - g^{-p}) = x^3 - Tr(g)x^2 + Tr(g)^p x - 1 \in \mathbb{F}_{p^2}[X].$$

This polynomial is fully determined by $Tr(g)$. The same holds for any power of g: for any integer n the conjugates of g^n are the roots of

$$x^3 - Tr(g^n)x^2 + Tr(g^n)^p x - 1 \in \mathbb{F}_{p^2}[X],$$

and the latter polynomial is fully determined by $Tr(g^n)$.

Basic XTR principle is that given $Tr(g)$ we can effectively compute $Tr(g^n)$. In cryptographic protocols $g, g^n \in \mathbb{F}_{p^6}$ can then be replaced by $Tr(g), Tr(g^n) \in \mathbb{F}_{p^2}$. Representation is then reduced to one third of the original. Using Algoritm 2.3.7 of [**70**] it is indeed possible to compute $Tr(g^n)$ given $Tr(g)$, and much faster than computing g^n from g. It is thus possible to use XTR in any cryptographic algorithms using group exponentiation by choice of suitable p, q, and $Tr(g)$.

In an XTR system, the parameters p, q, and $Tr(g)$ are public, precomputed, and (usually) shared among many users. In static system, user Alice generates randomly her secret key $d, 0 < d < q$, and publishes her public key $Tr(g^d)$. In Diffie-Hellman key exchange, Alice and Bob generate random a, b privately, and publicly exchange $Tr(g^a)$, and $Tr(g^b)$ respectively.

From the cryptanalytic point of view, we are interested in the problem opposite to XTR exponentiation, XTR-discrete logarithm (XTR-DL) problem:

DEFINITION 4.2.1 (XTR-DL). Let p, q, and $Tr(g)$ be XTR system parameters, i.e. p, q are prime, G is a subgroup of order p^2-p+1 of $\mathbb{F}_{p^6}$, and $g \in G$ is a generator of subgroup of G of order q.

XTR-discrete logarithm problem is to compute $d \in \mathbb{Z}_q$ for any given $Tr(g^d)$.

To solve the XTR-DL problem, it is useful to revert the trace representation back to $\mathbb{F}_{p^6}$, and to solve the corresponding DLP in this field (we suppose that XTR group security is high enough). It is not known, whether it is possible to solve XTR-DL with some index-calculus method acting directly on trace representations.

4.3. Representation conversions

Let XTR-DL instance be given in notation of Definition 4.2.1. Let us suppose that we are able to compute discrete logarithms in the whole multiplicative group of $\mathbb{F}_{p^6}$. Let x be a generator of $\mathbb{F}^*_{p^6}$. We will use notation $a = \log y$ to correspond to the least positive integer a, such that $x^a = y$. The order of XTR group is q, thus it suffices to compute

$$d = \frac{\log g^d}{\log g} \mod q. \tag{4.1}$$

We do not need to know the generator x, nor the exact value of the discrete logarithms in $\mathbb{F}_{p^6}$, but only their $\mod q$ values. However, we must convert traces $Tr(g)$, $Tr(g^d)$ back to corresponding elements g, g^d, in the chosen $\mathbb{F}_{p^6}$ representation.

Let $\mathbb{F}_{p^6}$ be represented as a vector space with the base $T = \{1, \theta, \theta^2, \ldots, \theta^5\}$. Here θ is a root of the degree 6 polynomial $f(x)$ irreducible over $\mathbb{F}_p$. Elements g, g^d are the roots of the polynomial

$$F(c, x) = x^3 - cx^2 + c^p x - 1, \tag{4.2}$$

where $c = Tr(g)$, or $c = Tr(g^d)$ respectively. The polynomial (4.2) has all its roots in $\mathbb{F}_{p^6}$. We can compute them using linear polynomials (algorithms in [**73**], pages 103–107). Other possibility is to use algorithm from Section 4.3 of [**73**].

Having fixed a representation of an element $\alpha \in \mathbb{F}_{p^2} \subset \mathbb{F}_{p^6}$, we can find 3 roots of polynomial $F(Tr(g), x)$, and 3 roots of polynomial $F(Tr(g^d), x)$. These roots are conjugates of g, and g^d over $\mathbb{F}_{p^2}$. As such, we can write them in the form

$$g^{p^{2i}}, \quad g^{dp^{2j}}, \quad i, j = 0, 1, 2.$$

Let a be any root of $F(Tr(g), x)$, and b any root of $F(Tr(g^d), x)$. Then clearly

$$d \in \{d_i = \frac{\log b}{\log a} p^{2i} \mod q | i = 0, 1, 2\}. \tag{4.3}$$

The exact value of d can thus be determined by computing two discrete logarithms in $\mathbb{F}_{p^6}$ modulo q, and at most 5 exponentiations in XTR group. All operations in transforming the problem from XTR-DL to DL problem in $\mathbb{F}_{p^6}$ are fast. We are thus only interested in an effective algorithm for computing discrete logarithms modulo q in $\mathbb{F}_{p^6}$, given by arbitrary field representation. The asymptotically fastest algorithm is the Number Field Sieve. However, to our best knowledge, until now it was not known how it can be practically implemented in extension fields of the degree 6. In this thesis, we present a practical implementation of the said algorithm, a comparison with other NFS applications, and address various specific issues related to the algorithm.

CHAPTER 5

The Number Field Sieve

The Number Field Sieve (NFS) is currently the fastest method (asymptotically) to solve the discrete logarithm and integer factorization problems. The complexity of factoring and DL methods is customarily given in terms of function $L_x(\alpha, c)$[1] defined by equation (2.4). This function can be seen as an interpolation between exponential complexity ($\alpha = 1$) and polynomial complexity ($\alpha = 0$). As such, problems in this complexity class are called subexponential.

The NFS algorithm was first introduced by Pollard in 1988 [**86**], in the context of factoring special integers of the form $x^3 + k$. It is interesting that the method was inspired by ideas of Coppersmith, Odlyzko and Schroeppel [**27**], originally presented in the context of discrete logarithms. Two main versions of the NFS for factoring are known: a version for factoring numbers of special form called Special Number Field Sieve (SNFS) [**67**]; and the method for general integers called General Number Field Sieve (GNFS) [**20**]. The main distinction is in the polynomial selection phase, leading to a different constant in complexity estimates. SNFS factors special integer N with complexity $L_N(1/3, c)$, where $c = (32/9)^{1/3} \approx 1.5262\ldots$. GNFS factors any integer N with $c = (64/9)^{1/3} \approx 1.9229\ldots$. Improvements of Coppersmith [**26**] have lowered this constant to $c \approx 1.9018\ldots$ It was shown by Schirokauer [**94**] that it is possible to lower the constant c further for integers N of low weight. The difference between GNFS and SNFS is clearly visible in practical implementations. While the current GNFS record for factoring RSA numbers [**111**] is the factoring of 663-bit integer, there was a recent record of factoring 1017-bit integer using SNFS [**6**]. Lenstra [**64**] estimates that factoring 768-bit RSA moduli is approximately 500-times more difficult (in practice) than factoring 1024-bit special number (and 1024-bit RSA 1000-times harder than 768-bit RSA).

Adaptation of the NFS to compute discrete logarithms was first presented by Gordon [**46**]. This adaptation was meant for computing discrete logarithms in field $\mathbb{F}_p$, with large prime characteristic p. Gordon's method had asymptotic complexity given by $L_p(1/3, c)$, $c \approx 2.0800\ldots$ Constant c was lowered by Schirokauer [**92**] to $c = (64/9)^{1/3} \approx 1.9229\ldots$ The method was later modified and implemented by Joux and Lercier [**52**]. They present an inovation called virtual logarithms, later more precisely described by Schirokauer [**93**]. The conjectured asymptotic complexity of this algorithm is again $L_p(1/3, (64/9)^{1/3})$. While the previous algorithms could only compute a single discrete logarithm per computation, the method with virtual logarithms have a pre-computation stage, and the individual logarithm computation stage. The improved version of the method is presented by Commeine and Semaev [**24**]. Precomputation

[1] More precisely, in asymptotic estimates $L_x(\alpha, c + o(1))$ is used, with the $o(1)$ term for $x \to \infty$.

has complexity $L_p(1/3, c)$, $c \approx 1.9018\ldots$, and individual logarithms can later be found with $L_p(1/3, 3^{1/3})$ cost.

Different methods apply when trying to solve the discrete logarithms in extension fields $\mathbb{F}_{p^n}$, with $n > 1$. If the characteristic is fixed (usually small, the most common being $p = 2$) and n tends to infinity, the best available algorithm is the function field sieve [**2, 5, 51**]. Its complexity is also $L_{p^n}(1/3)$.

However, until recently, the best available approach for the intermediate case $\mathbb{F}_{p^n}$, with medium to large p, had complexity $L_{p^n}(1/2)$ [**3, 4**]. This is also the case of XTR-DL problem, which reduces to a DL problem in the field $\mathbb{F}_{p^6}$. Different methods have been proposed for these fields. Granger and Vercauteren use torus-based approach [**48**]. A practical realization of this algorithm led to a record computation in a field with 556-bit field size [**53**]. Joux and Lercier [**54**] proposed a family of algorithms which are based on the function field sieve, with complexity $L_{p^n}(1/3)$. These results however do not apply to XTR over prime fields (as described in Chapter 4), but influence the complexity of XTR defined over extension fields, i.e. in fields $\mathbb{F}_{p^{6n}}$ [**74**].

Joux et. al. at CRYPTO 2006 [**55**] presented possible ways of modifying the NFS to solve DLP in all remaining finite fields. The variants of NFS depend on the relative size of p and n. For p large compared to $L_{p^n}(2/3)$, a new polynomial selection algorithm is used. For p small compared to $L_{p^n}(2/3)$, a sieve must be performed over elements of degree higher than one. In the case of fields $\mathbb{F}_{p^6}$, we have $p < L_{p^6}(2/3)$ for $p < 2^{383}$. Thus we are interested in the variant with higher degree elements. Results from a numerical experiment in $\mathbb{F}_{p^3}$ with 40-digit p (394-bit field size) are presented in [**55**]. Only a standard sieve was used. Up to date, we are aware of no (public) XTR-DL records or even published XTR-DL computations using this variant of NFS algorithm, or a public implementation of higher dimensional sieving algorithm.

5.1. The NFS algorithm overview

The Number Field Sieve is in fact not a single algorithm, but a family of related algorithmic methods. There are many variants and optional parameters that influence the practical implementation of the NFS. In general, NFS (for both DLP in integer factoring) is characterized by these basic steps:

(1) **Parameter selection.**
(2) **Sieving.**
(3) **Linear algebra.**
(4) **Postprocessing.**

Another distinct characteristic of the NFS algorithm is that it operates with algebraic numbers. The main goal of the NFS is to collect enough relations among algebraic integers and/or their images under certain homomorphisms.

5.1.1. Parameter selection. The performance of the NFS can be influenced by various parametric choices. The most important are the choice of a sieve polynomial(s),

a smoothness bound and a sieve region shape. Other parameters are concerned with a sieving method, the use of the large primes, factoring techniques, etc.

The polynomial selection depends on the problem we are trying to solve. SNFS polynomial selection uses a special form of the number N we want to factor. GNFS and classical NFS for DLP in $\mathbb{F}_p$ try to find a polynomial with small coefficients and a small root modulo N or p respectively. The method of [**52**] suggests to use two polynomials with small coefficients and common root in $\mathbb{F}_p$. The method of [**55**] just extends the previous method to find two polynomials with a common root in $\mathbb{F}_{p^n}$. Various polynomial selection methods are further explored in Section 6.3.

Selection of other parameters is given by asymptotically optimal estimate. However, this estimate contains $o(1)$ term in the exponent. The practically optimal parameter set is usually determined experimentally (e.g. using preliminary sieving, or numeric methods to estimate the smoothness density).

5.1.2. Sieving. The goal of the sieving step is to identify smooth elements of the sieve region. A naïve method would be to trial divide all elements of the sieve region. This takes $O(B/\ln B)$ steps per every sieve region element. By employing the sieve, we can reduce this to $O(\ln \ln B)$ steps per sieve region element [**62**].

This phase of the algorithm is (under asymptotically optimal parametric setting) the most time consuming phase of the algorithm. However, it can be easily distributed to many computers running in parallel. It is also not very resource-intensive. The sieving algorithm is common for many NFS variants (including SNFS/GNFS for factoring, and NFS for DLP). The factoring and DLP variants differ in further processing of (semi-)smooth elements. In the XTR case, we have to implement a higher dimensional sieve, as presented in Chapter 7. Its concept is similar to a classical sieve, but there are some specific details concerning higher dimensional elements, e.g. the possible polynomial factorization.

5.1.3. Linear algebra. The sieving phase is used to collect a large set of linear equations. The set of equations obtained is very sparse. Such a large sparse system can be solved by using combinations of Structured Gaussian elimination [**84, 52**] and the conjugate gradient [**49**], Lanczos [**60**] (see also [**27, 84**] for implementation details) and Wiedemann methods [**112**]. Overview of this methods and their implementation is provided in [**59**]. We have used custom developed software (adaptation of [**105**]) to solve linear system, and as such it was not a primary object of our research. However, to finalize our largest sieving experiment, we had to further optimize the Lanczos solver, and adapt a specific version of Structured Gaussian Eliminations. Our experiments are described in more details in Section 8.4.4.

The linear algebra step is very resource intensive (takes a lot of memory), and is very difficult to run in parallel. It is thus a bottleneck of the whole NFS method. Sometimes a suboptimal set of parameters (a smaller factor base and a larger sieve region) is used to reduce the size of the linear system. This leads to a longer sieving time, but the sieving time cost can be transferred to many computers running in parallel. The extend

of this trade-off is severely limited by the decreasing smoothness probability in extended sieve regions.

5.1.4. Postprocessing. Postprocessing is a very specific phase of the NFS computation. In the factoring applications, the square root of certain algebraic numbers have to be found. Once it is computed, we have a chance to find factorization of N. If not successful, we can try another solution of the linear system.

In the case of discrete logarithm computation, original methods [**46**] used a special relation involving the unknown logarithm. Method [**52**] bring a new notion of virtual logarithms [**93**], which are simply the direct solutions of the linear system from linear algebra phase. The specific logarithm is computed in the post-processing phase. We construct a specific equation involving the unknown logarithm on one side, and a semi-smooth part on the other side. Virtual logarithms in the smooth part are known. Any additional virtual logarithms of large prime factors in the semi-smooth part can be computed by constructing more special equations, until we reach an equation with only a single large factor. This, so called descent method, is explained later.

5.2. Using NFS to solve DLP in $\mathbb{F}_{p^n}$

5.2.1. Virtual logarithms. There are many variants of NFS, that can be used or adapted to compute a required discrete logarithm in $\mathbb{F}_{p^n}$. We are using the method of "virtual logarithms" [**52, 93**] based on Schirokauer's maps [**92**]. The main advantage of the method is that both the sieving phase and the linear algebra phase are independent of the concrete individual logarithms computed, thus can be executed only once for a given finite field. The final algebraic description of the method was presented in [**55**] as follows.

Let $\alpha_1, \alpha_2 \in \mathbb{C}$ be roots of two distinct monic polynomials $f_1, f_2 \in \mathbb{Z}[x]$ irreducible over $\mathbb{Z}$. Then $K_1 = \mathbb{Q}(\alpha_1)$, $K_2 = \mathbb{Q}(\alpha_2)$ are two algebraic number fields. Let t be a common root of f_1, f_2 in $\mathbb{F}_{p^n}$, which does not belong to any proper subfield of $\mathbb{F}_{p^n}$. This means that both f_1 and f_2 have a common irreducible factor f_0 of degree n over $\mathbb{F}_p$. Let $\mathcal{O}_K$ denote a ring of integers of the field K. Let $\phi_j : \mathcal{O}_{K_j} \to \mathbb{F}_{p^n}$, $\phi_j(\sum a_i \alpha_j^i) = \sum a_i t^i \mod p$, for $j = 1, 2$. These two mappings are ring homomorphisms with common image $\mathbb{F}_{p^n}$.

Let $\xi_1 = \sum a_i \alpha_1^i \in \mathcal{O}_{K_1}$ have a smooth norm $N(\xi_1) = \prod p_{1,j}^{e_j}$, with $p_{1,j} < B$. Then the corresponding principal ideal $(\xi_1) = \xi_1 \mathcal{O}_{K_1}$ can be uniquely factored as a product of prime ideals lying over primes $p_{1,j}$ (ξ_1 is a B-smooth algebraic integer). Let g be a generator of $\mathbb{F}_{p^n}^*$ and q be a (large) prime dividing $|\mathbb{F}_{p^n}^*| = p^n - 1$. Let the decomposition of the principal ideal generated by a smooth algebraic integer $\xi_1 \in K$ be

$$(\xi_1) = \prod \mathfrak{p}_{1,j}^{v_{1,j}}. \tag{5.1}$$

Let $h_1 = h(K_1)$ be the class number[2] of the field K, and let $\gcd(q, h_1) = 1$. Using decomposition (5.1), we can write

[2]We do not need to compute the class number in practice.

$$\xi_1^{h_1} = u_1 \prod \pi_{1,j}^{v_{1,j}}, \tag{5.2}$$

with u_1 a unit and $\pi_{1,j} \in \mathcal{O}_K$, $(\pi_{1,j}) = \mathfrak{p}_{1,j}^h$. We can apply mapping ϕ_1 to this equation and obtain equation in $\mathbb{F}_{p^n}$:

$$\phi_1(\xi_1)^{h_1} = \phi_1(u_1) \prod \phi_1(\pi_{1,j})^{v_{1,j}}, \tag{5.3}$$

and after taking logarithms in $\mathbb{F}_{p^n}^*$ we get a linear equation

$$h_1 \log_g \phi_1(\xi_1) = \log_g \phi_1(u_1) + \sum v_{1,j} \log_g \phi_1(\pi_{1,j}) \pmod{p^n - 1}. \tag{5.4}$$

Let us suppose further that $\xi_2 = \sum a_i \alpha_2^i \in \mathcal{O}_{K_2}$ is also a B-smooth algebraic integer. Using similar steps as above we can find an equation:

$$h_2 \log_g \phi_2(\xi_2) = \log_g \phi_2(u_2) + \sum v_{2,j} \log_g \phi_2(\pi_{2,j}) \pmod{p^n - 1}, \tag{5.5}$$

From the definition of ϕ_j we get

$$\phi_1(\xi_1) = \phi_2(\xi_2) = \sum a_i t^i. \tag{5.6}$$

Combining equations (5.4), (5.5), and (5.6), we get an equation

$$h_1^{-1} \log_g \phi_1(u_1) + \sum v_{1,j} h_1^{-1} \log_g \phi_1(\pi_{1,j}) = h_2^{-1} \log_g \phi_2(u_2) + \sum v_{2,j} h_2^{-1} \log_g \phi_2(\pi_{2,j}) \pmod{p^n - 1}. \tag{5.7}$$

We will denote by $x_{i,j}$ unknown numbers[3] $h_i^{-1} \log_g \phi_i(\pi_{i,j})$. Every $x_{i,j}$ corresponds to a single ideal $\mathfrak{p}_{i,j}$ with norm $p_{i,j} < B$, and is called a virtual logarithm of the ideal $\mathfrak{p}_{i,j}$.

Every pair of B-smooth integers (ξ_1, ξ_2) leads to a similar equation (5.7). The set of unknown virtual logarithms of ideals (with the norm below B) is always the same, and is limited to $O(B)$ elements. Unfortunately, units u_1, u_2 in every equation are generally different, as well as their virtual logarithms $h_i^{-1} \log_g \phi_i(u_i)$. We can overcome this problem by decomposing each unit into fundamental units. However, in this case we need to know both fundamental units, and the class number of the field. This is a difficult problem for most of the degree 6 number fields. Another possible solution is to use Schirokauer's logarithmic maps as described in section 5.2.2. In this case we are limited to a computation modulo a (large) prime factor of $(p^n - 1)$. In our case (XTR-DL computation), it is actually simply the group order q.

[3]As shown further in section 5.2.2, algebraic numbers $\pi_{i,j}$ are multiplied by a suitable well-defined unit, but still represent the same prime ideal.

In each case, we are able in computationally efficient way to replace terms $h_i^{-1} \log_g \phi_i(u_i)$ in equation (5.7) by a linear combination (modulo q) of a small number of "virtual" unknowns Λ_j:

$$h_i^{-1} \log_g \phi_i(u_i) = \sum_j \lambda_{i,j}(\xi_i)\Lambda_{i,j} \pmod q \tag{5.8}$$

Finally, we get an equation

$$\sum_j \lambda_{1,j}(\xi_1)\Lambda_{1,j} + \sum_j v_{1,j}x_{1,j} \equiv \sum_j \lambda_{2,j}(\xi_2)\Lambda_{2,j} + \sum_j v_{2,j}x_{2,j} \pmod q, \tag{5.9}$$

with unknown virtual logarithms of maps $\Lambda_{i,j}$, and unknown virtual logarithms of ideals $x_{i,j}$. We call equation (5.9) a smooth equation.

A set of all prime ideals in $\mathcal{O}_{K_i}$ lying over primes $p_{i,j} < B$ is called an algebraic factor base, denoted $\mathcal{B}_i$. Let $c_1 = |\mathcal{B}_1|$, and $c_2 = |\mathcal{B}_2|$. The number of prime ideals with the norm smaller than B can be bounded by $O(B)$. The number of unknown virtual logarithms of maps is bounded by the degree n of the number field. We have at most $c = c_1 + c_2 + 2n = O(B)$ different unknowns in the set of equations.

The most important part of the efficient NFS implementation is the fast method to identify pairs of smooth algebraic integers ξ_1, ξ_2. The most common technique is to employ some kind of a sieving algorithm (which gave the NFS its name). We dedicate Chapter 7 to a description of our sieving algorithm, as well as general remarks concerning the issues connected with the sieving.

We need to find $O(B)$ smooth pairs to construct more than c linearly independent smooth equations. Virtual logarithms are then non-trivial solutions of this linear system. By combining equations (5.5), and (5.8), and using computed virtual logarithms $\Lambda_{i,j}$, $x_{i,j}$ we can finally compute discrete logarithms of special elements of $\mathbb{F}_{p^n}^*$:

$$\log_g \phi_2(\xi_2) = \log_g \left(\sum a_i t^i\right) = \sum_j \lambda_{1,j}(\xi_1)\Lambda_{1,j} + \sum_j v_{1,j}x_{1,j} \pmod q, \tag{5.10}$$

To compute discrete logarithms of arbitrary elements of $\mathbb{F}_{p^n}^*$, we can use the so called descent technique described in Section 5.2.4.

5.2.2. Schirokauer maps. Let us suppose that unknown units u_1, u_2 in equation (5.7) are both q-th powers, i.e. $u_i \in (\mathcal{O}_{K_i}^*)^q$. Then clearly $\log_g \phi_i(u_i) = 0 \pmod q$. On the other hand, if units u_1, u_2 are not q-th powers, we would like to transform the equation (5.7) (in a computationally efficient way) to another smooth equation with only q-th powers of units.

To accomplish this, we can employ Schirokauer's logarithmic maps [**92, 93**]. Let us work in a single number field K defined by a root α of a polynomial f. For an element $\sum a_i \alpha^i$ we can compute these maps as follows:

(1) Let factorization of $f(x)$ over $\mathbb{F}_q$ (we suppose that q does not divide $D(f)$, i.e. q is not ramified in K) be:
$$f(x) \equiv \prod f_i(x) \pmod{q}.$$
(2) Compute $e = \operatorname{lcm}(q^{\deg f_i} - 1)$.
(3) Compute $l\left(\sum a_i x^i\right) = \left(\sum a_i x^i\right)^e - 1 \mod f(x) \mod q^2$.
(4) Let $l\left(\sum a_i x^i\right)/q = \sum \lambda_i x^i$.
(5) Output vector $\lambda = (\lambda_1, \lambda_2, \ldots, \lambda_n)$, $\lambda_i \in \mathbb{Z}_q$.

Using this algorithm we get more logarithmic maps than required: n instead of $r = r_1 + r_2 - 1$, where $n = r_1 + 2r_2$ is a degree of K, r_1 is the number of its real embeddings and $2r_2$ is the number of its imaginary embeddings. The difference is not significant when $n = 6$.

The maps are logarithmic, i.e. have the property $\lambda_i(\xi_1 \cdot \xi_2) = \lambda_i(\xi_1) + \lambda_i(\xi_2)$. Also, a unit u is q-th power if and only if $\lambda(u) = 0$. We can find a specific set of units $\nu_1, \nu_2, \ldots, \nu_r$, such that
$$\lambda_i(\nu_j) = \begin{cases} 1, & \text{if } i = j; \\ 0, & \text{if } i \neq j. \end{cases}$$

Thus we can write any unit as
$$u = \zeta^q \prod_{i=1}^{r} \nu_i^{\lambda_i(u)}, \tag{5.11}$$
with ζ also an unspecified unit.

In equation (5.2) we can multiply elements $\pi_{1,j}$ by a well-defined unit to obtain
$$\pi'_{1,j} = \pi_{1,j} \prod_{i=1}^{r} \nu_i^{-\lambda_i(\pi_{1,j})},$$
such that $\lambda(\pi'_{1,j}) = 0$. Now we can rewrite equation (5.2) to $\xi_1^{h_1} = u'_1 \prod \left(\pi'_{1,j}\right)^{v_{1,j}}$, with $h_1 \lambda(\xi_1) = \lambda(u_1)$. Using (5.11) we get finally

$$\xi_1^{h_1} = \zeta^q \prod_{j=1}^{r} \nu_j^{h_1 \lambda_j(\xi_1)} \prod \left(\pi'_{1,j}\right)^{v_{1,j}}. \tag{5.12}$$

Applying mapping ϕ_1 and taking logarithms $\mod q$, we get
$$\log_g \phi(\xi_1) = \sum_{j=1}^{r} \lambda_i(\xi_1) \log_g \phi(\nu_j) + \sum_j v_{1,j} h_1^{-1} \log_g \phi\left(\pi'_{1,j}\right). \tag{5.13}$$

Note that the virtual logarithms of units $\Lambda_{i,j}$ from equation (5.8) correspond to $\log_g \phi(\nu_j)$, and the virtual logarithms of ideals $x_{i,j}$ correspond to $h_1^{-1} \log_g \phi\left(\pi'_{1,j}\right)$. The underlying algebraic numbers are in fact never computed, as we are only interested in actual values of $\Lambda_{i,j}$, and $x_{i,j} \mod q$ required to compute unknown discrete logarithms in $\mathbb{F}_{p^n}$. These values are found directly as a solution of the set of smooth equations.

5.2.3. Linear algebra. After finding enough smooth pairs (ξ_1, ξ_2) we can construct the matrix equation

$$(L_1|V_1)\begin{pmatrix}\Lambda_1\\ x_1\end{pmatrix} = (L_2|V_2)\begin{pmatrix}\Lambda_2\\ x_2\end{pmatrix} \pmod q, \tag{5.14}$$

where L_1, L_2 are matrices with rows containing Schirokauer maps, and V_1, V_2 are matrices with rows containing ideal valuations $v_{i,j}$. We must solve this system to find unknown virtual logarithms $\Lambda_1, \Lambda_2, x_1, x_2$. We are looking for a non-trivial solution, i.e. a solution with a property

$$(L_1|V_1)\begin{pmatrix}\Lambda_1\\ x_1\end{pmatrix} \neq 0 \pmod q. \tag{5.15}$$

To solve this system, we transform it to the following form:

$$(L_1|V_1| - L_2| - V_2)\begin{pmatrix}\Lambda_1\\ x_1\\ \Lambda_2\\ x_2\end{pmatrix} = 0 \pmod q. \tag{5.16}$$

A non-trivial solution of the system (5.14) is a set of virtual logarithms (with an unknown base). It must lie in a null space of the matrix $A = (L_1|V_1| - L_2| - V_2)$.

Let us suppose that we are able to find a smooth element $\gamma \in \mathcal{O}_{K_1}$, such that $\phi(\gamma) = g$ is a generator of $\mathbb{F}_{p^6}^*$. We can then construct an equation

$$\log_g \phi_2(\gamma) = 1 = \sum_j \lambda_{1,j}(\xi_1)\Lambda_{1,j} + \sum_j v_{1,j}x_{1,j} \pmod q. \tag{5.17}$$

If we add this equation to the linear system (5.14), or (5.16) respectively, we can force the solutions of individual logarithms to be in base g. In our experiments we were working with just the system (5.16) without any additional equations for the generator (it was never required).

Matrix A is a large sparse matrix with $O(B)$ rows and columns. Let us suppose that we have found more smooth equations than we have unknown virtual logarithms, and the matrix A has a non-trivial null space. The entries in columns of A containing Schirokauer maps are usually all non-zero $\lfloor \log_2 q \rfloor$-bit numbers. The number of non-zero entries in columns containing valuations is inversely proportional to the norm of a corresponding prime ideal, i.e. $w_H(A_{\mathfrak{p}_j}) \sim O\left(B/N(\mathfrak{p}_j)\right)$. Non-zero valuations are usually only numbers ± 1, or small positive or negative numbers (their number in a column is again inversely proportional to a corresponding power of the prime ideal's norm). The number of non-zero entries in a row is $R = 2n + \omega(N(\xi_1)) + \omega(N(\xi_2))$, where ω denotes the number of distinct prime divisors of a number. A typical bit size of the norm of a smooth ξ is $\log_2 B^n$. Most of the prime ideals have norm with the bit size near $\log_2 B$. Thus we can expect that $\omega(N(\xi)) = O(n)$, and also $R = O(n) << O(B)$.

To solve the linear system we decided to use the Lanczos algorithm as follows.

ALGORITHM 1. Lanczos algorithm

INPUT: Symmetric $r \times r$ matrix B, column r-vector $w \neq 0$.

OUTPUT: Column n-vector x, such that $Ax = w$.

(1) Compute:
   (a) $w_0 = w$.
   (b) $v_1 = Bw_0$.
   (c)
$$w_1 = v_1 - \frac{v_1 \cdot v_1}{w_0 \cdot v_1} w_0.$$
(2) For $i = 2 \ldots$:
   (a) If $w_i = 0$, set $j = i$, exit to step 3.
   (b) Compute $v_{i+1} = Bw_i$.
   (c) If $v_{i+1} \cdot v_i = 0$, algorithm FAILS.
   (d) Compute
$$w_{i+1} = v_{i+1} - \frac{v_{i+1} \cdot v_{i+1}}{w_i \cdot v_{i+1}} w_i - \frac{v_{i+1} \cdot v_i}{w_{i-1} \cdot v_i} w_{i-1}.$$
(3) Compute and report
$$x = \sum_{i=0}^{j-1} \frac{w_i \cdot w}{w_i \cdot v_{i+1}} w_i$$

There exists some $j \leq r$, for which $w_j = 0$ and algorithm terminates. There are however two main problems with Algorithm 1. First of all, the matrix A is not a symmetric square matrix, but a general $r \times c$ matrix. Moreover, we want to solve an equation with zero right hand side. Instead of the original equation $Ax = 0$, we can generate random r-vector y, and compute $Ay = y'$. Now let us solve the equation $Ax' = y'$, in an unknown x'. The solution yields $Ax' - y' = 0$, or $Ax' - Ay = 0$ respectively. I.e. $x = x' - y$ is the solution of the original equation. To solve $Ax' = y'$, we can multiply both sides by A^T. A solution x' of $A^T Ax' = A^T y'$ is most likely also the solution of $Ax' = y'$. Now $B = A^T A$ is a symmetric $m \times m$ matrix, and $w = A^T y'$ is a non-zero column n-vector, thus we can solve $Bx' = w$ by Algorithm 1.

To take advantage of the sparsity of A, we never compute B directly. Instead of computing $v_{i+1} = Bw_i$ in the Algorithm 1, we compute $v_{i+1} = A^T(Aw_i)$ using two sparse matrix-column multiplications. The complexity of the method then depends mostly on the number of iterations in Step 2, and the cost of the sparse matrix-column multiplications. If matrix A has full column rank, then the number of iteration in Step 2 is c (the number of columns). The number of finite field operations required for the sparse matrix-column multiplications is proportional to the number of non-zero elements of A, so the total complexity of Lanczos algorithm is $O(crR)$, where R is the average number of non-zero elements in a row. The linear algebra step of the NFS thus requires $O(nB^2)$ steps.

The main disadvantage of the Lanczos algorithm is the fact, that it cannot be parallelized in an efficient manner on a distributed network of computers. Another prohibitive part of the algorithm is the storage required for the matrix A. We can see, that the memory complexity is also determined by the number of non-zero elements of A, i.e. is $O(rR)$. To reduce the complexity of the Lanczos algorithm, we try to reduce the original matrix A to some smaller but still sparse matrix $\hat{A}$. We remove some of the rows and columns in such a way, that the solution $\hat{x}$ of $\hat{A}\hat{x} = 0 \pmod{q}$ can be used to compute the original x (in a efficient way). This process is similar to the Gaussian elimination.

The classical Gaussian elimination is not suitable for a special system arising in the NFS, because it does not take advantage of the sparsity of the system, and its special statistical properties. There is a large number of specific adaptations of Gaussian elimination which take into account a specific structure of matrix A. The method was introduced under the name Structured Gaussian Elimination (SGE) by LaMacchia and Odlyzko [**58**]. A refined version of this method from [**59**] first splits the matrix A into a heavy part (dense columns) and a light part (sparse columns). Then specific row reductions (such as removing entire columns with a single non-zero entry as well as corresponding rows) are performed. The number of non-zero coefficients in the light part should never increase. A different type of SGE is used by Joux and Lercier [**52**], which uses weight change estimation to select pivots during the SGE process. We have implemented yet another specific type of SGE. Details of our SGE adaptation as well as optimized Lanczos solver used for finding virtual logarithms in our "record" solution are described in Section 8.4.2.

SGE is usually very fast (depends on the used heuristics). The result of SGE is a significantly smaller sparse matrix $\hat{A}$. Matrix $\hat{A}$ is usually denser than A, but still very sparse (number of non-zero entries in a row is small in comparison to the number of columns). Thus the complexity of solving $\hat{A}\hat{x} = 0 \pmod{q}$ by the Lanczos algorithm is significantly smaller than solving original problem $Ax = 0 \pmod{q}$. After finding a non-trivial solution of $\hat{A}\hat{x} = 0 \pmod{q}$, we can either proceed directly to the individual logarithm phase (Section 5.2.4) with a smaller set of known virtual logarithms; or more preferably we can backtrack SGE, and find a non-trivial solution of the whole $Ax = 0 \pmod{q}$. The total cost of SGE + Lanczos + backtrack can be controlled by the parametrization of the SGE. We should stop SGE, if it starts to increase the value of crR (number of field multiplications in the Lanczos solver), or when the additional processing cost required to decrease it by 1 is higher than the field multiplication cost in the Lanczos solver.

5.2.4. Individual logarithms. Let $f_0(x)$ be a polynomial of degree n irreducible over $\mathbb{F}_p$, and let $f_0|f_1$ and $f_0|f_2$ in $\mathbb{F}_p[x]$. Let t be a root of $f_0(x)$. Represent $\mathbb{F}_{p^n}$ as $\mathbb{F}_p[t]/(f_0(t))$.

Our aim is to find an unknown discrete logarithm of an element $y \in \mathbb{F}_{p^n}$. Let $h \in \mathbb{F}_{p^n}$ be the element corresponding to the algebraic number with the largest smooth norm found during the sieve phase. Using equation (5.10), and computed virtual logarithms, we can compute the discrete logarithm of h.

Let $z = y^i h^j$ for some $i, j \neq 0$. We can write z in the form

$$z = \frac{\sum a_i t^i}{\sum b_i t^i}, \tag{5.18}$$

such that both $\xi_a = \sum a_i \alpha_1^i$ and $\xi_b = \sum b_i \alpha_1^i$ have B_1-smooth norms, with $B_1 = L_{p^n}\left(2/3, (1/3)^{1/3}\right)$. We could also use lifts to K_2 (change t to α_2) instead, but the corresponding norms would be higher. If we are able to find logarithms of $a = \sum a_i t^i$ and $b = \sum b_i t^i$, then we can also compute the logarithm of y using equation

$$\log_g y = 1/i(\log_g a - \log_g b - j \log_g h) \mod q. \tag{5.19}$$

According to [**55**], to find the fractional representation (5.18), it is possible to use the lattice reduction (LLL algorithm) as follows: Apply the LLL algorithm to the following lattice (vectors in columns):

$$L = \begin{pmatrix} z & tz & \cdots & t^{n-1}z & p & tp & \cdots & t^{n-1}p \\ 1 & 0 & \cdots & 0 & 0 & 0 & \cdots & 0 \\ 0 & 1 & \cdots & 0 & 0 & 0 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & 1 & 0 & 0 & \cdots & 0 \end{pmatrix}.$$

Field elements in the first line are represented as column vectors of their coordinates. In the reduced lattice, every column represents coordinates of both the numerator a (upper half), and the denominator b (lower half). If n is not too large, we expect to find a short vector with the L_0 norm $\sqrt{p}$, i.e. the corresponding norm in K_1 is $O(p^3)$.

Now we can find decomposition of (ξ_a) and (ξ_b) respectively, and write corresponding equations (5.10). Part of the equation can be simplified using known virtual logarithms. Any ideal with the norm above B will represent a new unknown virtual logarithm we need to find.

Let $\mathfrak{q}$ be a prime ideal with norm q and an unknown virtual logarithm. We can use the special-q sieve with the new smoothness bound $B_2 < q$ to find a semi-smooth equation similar to (5.9). The special-q sieve means that we sieve a lattice defined by the ideal $\mathfrak{q}$. Every sieved algebraic integer generates a principal ideal divisible by $\mathfrak{q}$. Thus, its virtual logarithm is in every equation (5.9) found by the special-q sieve.

New unknown virtual logarithms produced by the additional sieving define a new virtual logarithm we need to compute. During the special-q sieve we only allow large prime factors below q^c, $c < 1$. Thus we can descend to smaller and smaller special $\mathfrak{q}$'s. Finally, we can reduce the problem finally to a semi-smooth equation, where only special $\mathfrak{q}$ is larger than B. We compute the corresponding $\mathfrak{q}$, and backtrack the descent. Commeine and Semaev in [**24**] give optimal parameter choices for reductions, and show why the algorithm stops, along with its complexity.

5.2.5. Practical implementation for $\mathbb{F}_{p^6}$. The algebraic notation of the NFS can be quite confusing from the implementation point of view. We give here a simplified description of the NFS for the case $\mathbb{F}_{p^6}$ with related comments and remarks for every step of the algorithm.

INPUT: Primes p, $q|p^6 - 1$.

(1) Find suitable polynomials:
   (a) Find a monic degree 6 polynomial $f_1(x)$ irreducible over $\mathbb{F}_p$, with small coefficients.
   (b) Let $f_2(x) = f_1(x) + p$, or $f_2(x) = f_1(x) - p$.
   (c) Check that discriminants $D(f_1)$ and $D(f_2)$ are not divisible by q.

   More discussion on polynomial selection is provided in Section 6.3. The ideal arithmetic is simplified in the case, when for both fields K_1 and K_2 (defined by roots of f_1 and f_2 respectively) $\mathcal{O}_K = \mathbb{Z}[\alpha]$. This can be checked while generating the factor base.

(2) Let $B = L_{p^6}\left(1/3, (8/9)^{1/3}\right)$.

   The chosen formula for B is derived from the asymptotic optimum as discussed in section 6.2. It is possible to specify a different value of B, which can be chosen from quite a broad interval. However, for smaller B the smoothness density decreases, and we need a larger sieve region. In practice, if B is chosen too small, or too large, it is not possible to find enough smooth equations. The choice of B can further be influenced by the implementation constraints, e.g. by the memory constraints.

(3) Construct factor bases. For each prime $p_j < B$ compute

$$f(x) \equiv \prod_{i=1}^{k} f_i(x) \pmod{p_j}.$$

   We add ideal $\mathfrak{p}_{j,i}$, represented by two element representation $(p_j, f_i(\alpha))$, to the corresponding factor base. The ideal has degree $\deg f_i$ and norm $p_j^{\deg f_i}$.

   Ramified primes, i.e. the primes where $f(x) \mod p_j$ have multiple factors, are slightly more difficult to process. As their number is limited, we can skip them during the sieve. For efficiency reasons we do not sieve with prime ideals with norm smaller than some $B_{\min}$. If we sieve only elements of degree 2, only the ideals of degree at most 2 can appear in any usable smooth equation. If B is too large, it is possible to precompute only a part of the factor base, and fill in the rest during the sieve.

(4) Compute sieve bounds X, Y, Z, and sieve the region of triples $(x, y, z) \in [-X, X] \times [-Y, Y] \times [1, Z]$. Every triple corresponds to a pair of algebraic integers $(x + y\alpha_1 + z\alpha_1^2, x + y\alpha_2 + z\alpha_2^2)$. Sieve reports only triples (x, y, z) that correspond to smooth pairs having $\gcd(x, y, z) = 1$, and $D = y^2 - 4xz$ is not square in $\mathbb{Z}$.

   We have implemented a sieve that is not deterministic (see Chapter 7). Thus the reported points have only a certain probability of corresponding to smooth integers. This probability must be high, otherwise the postprocessing (computing norms, factoring and computing valuations) would be too costly. We are checking gcd *and D to avoid linearly dependent relations. Our siever also produces a partial factorization of the corresponding norms for every reported point to speed up the postprocessing.*

(5) For every triple (x, y, z) construct equation (5.9) by computing corresponding Schirokauer's maps, and exact ideal valuations. If the number of equations is less than the number of unknown virtual logarithms, either extend the sieve region, or terminate the algorithm with FAIL.

Computation of Schirokauer's maps is described in Section 5.2.2. Ideal valuation can be usually computed directly from the prime decomposition of a norm $N(x + y\alpha + z\alpha^2)$. *Let the norm's prime decomposition contain* p_i^e. *Ideals over* p_i *in ideal decomposition of* $(x + y\alpha + z\alpha^2)$ *can be found by computing* $r(X) = \gcd(f(X), x + yX + zX^2) \mod p_i$. *If* $r(X)$ *has degree 1, it determines exactly one degree 1 ideal* $(p_i, r(\alpha))$ *appearing in the e-th power. If* $r(X)$ *is irreducible of degree 2, it again determines a single degree 2 ideal* $(p_i, r(\alpha))$ *appearing in power* $e/2$ *(e must be even). A more complicated situation can appear if* $r(X)$ *can be factored* $\mod p_i$ *or in the case of ramified primes. It is thus better to compute exact ideal valuations by using Algorithm 4.8.17 of* [**23**].

(6) Find non-trivial solution of the system of equations modulo q. Store computed "virtual logarithms".

(7) Compute individual logarithms as described in Section 5.2.4.

In the XTR case, we should transform the XTR representation to a representation $\mathbb{F}_p[t]/(f_1(t))$ *as described in Section 4.3.*

CHAPTER 6

NFS complexity and polynomial selection

6.1. Implementation choices influencing NFS

The implementation of the NFS, and even the estimation of its cost for a specific purpose, is quite a complicated task. A large and detailed analysis of the cost of factoring RSA-1024 can be found in [**69, 62**]. A real-time performance is influenced by many implementation options, which (are believed to) influence the NFS only in the $o(1)$ term in $L_{p^6}(1/3, c + o(1))$ complexity notation. Even this $o(1)$ term is very important for the cost and security estimates, as it usually distinguishes possibilities of a publicly available software tool and a special "record" implementation. Among these options applicable to DLP in $\mathbb{F}_{p^6}$ are:

(1) *Choice of the polynomials.* The (asymptotically) optimal polynomial degree is usually given by a real number that must be rounded. It is not clear, whether it is better to use two algebraic sides, or a single rational and one or more algebraic sides. In the nowadays implementable case of $\mathbb{F}_{p^6}$, a polynomial with larger degree than optimal must be used. It is compensated by the higher dimensional sieve, but the exact relations are not yet fully understood. It is possible to choose the sieve polynomials from a large pool of irreducible polynomials, and there are many methods suggested for polynomial selection, including non-monic polynomials. Remarks on polynomial selection for XTR-DL solution are summarized in Section 6.3.

(2) *Choice of the smoothness bound.* The asymptotically optimal smoothness bound B is well-known. However, in a practical implementation quite a wide interval for actual value of B is possible [**104**]. This also complicates complexity extrapolations based on the existing NFS records. Experiments in [**62**] show that the parameter B used for RSA-512 factorization [**22**] was below optimal value, and the direct extrapolation leads to a wrong parametric setting for RSA-1024.

    Sometimes a smaller value of B is required due to the implementation constraints, such as the limited memory for the siever, or for the linear algebra phase. On the other hand, choosing slightly larger values of B can even speed up the sieve and improve the efficiency of the Structured Gaussian elimination [**52**].

(3) *Choice of the sieving region.* All elements in the optimal sieve region should have norms bounded by a certain constant derived from B. The shape of the planar region with smallest possible number of points required to sieve is however quite complex [**104**]. In practice only a rectangular region is used.

(4) *Smoothness detection methods.* The cost of the norm computation for all elements of the sieve region is quite large, as well as the cost of multiplication/division of large numbers. Thus logarithmic estimates are used instead of an exact smoothness detection (see Section 7.2). Small primes are usually excluded from the sieve due to the efficiency reasons, as well as higher powers of ideals. The smoothness detection must balance the precision of the logarithmic estimates, and various error factors coming from the excluded factor base elements.
(5) *Large primes and semismooth equations.* A large number of equations of type (5.9) in the sieve region is almost smooth, i.e. contain only one (or small number) of prime factors only slightly larger than the smoothness bound B. These primes can easily be detected during the sieve and used to construct semi-smooth equations [**67**]. They were successfully employed in most of the factoring applications of the NFS [**22**]. However, the use of the large primes for DLP is questionable [**52**]. We discuss large primes in more details in Section 7.5.
(6) *Lattice sieve.* In the DLP solution, the large primes extend the set of unknowns for the linear system. Thus we must find at least two semismooth relations with the same large prime, for this prime to be useful. Even if we cannot find another occurrence of the large prime directly during the sieve, we can usually construct relations containing this prime by the lattice sieve [**86, 44**]. Only a specific sublattice (given by the chosen prime ideal) is sieved. The lattice sieve can be used not only for large primes, but also for ideals from the factor base. The efficiency of the lattice sieve is decreased in the case of higher degree polynomials, as the norms on the sublattice grow faster [**12**]. A fast lattice reduction algorithm is required, which provides a problem when sieving higher degree elements.
(7) *Optimal higher dimensional sieve region.* The norms of the elements grow not only with the absolute value of its coefficients, but also with the growing degree. The shape of the optimal sieve region in higher dimensions is thus more complicated as in the two dimensional case. Another problem is the ratio of computing required to sieve the higher dimension to the number of relations found by extending the sieve region to this dimension.
(8) *More than two algebraic fields.* The method of Coppersmith [**26**] uses a single rational side and multiple algebraic sides to decrease the constant c in the asymptotic complexity estimate. It is possible to extend the NFS in $\mathbb{F}_{p^6}$ to use more than two algebraic fields (with a similar reasoning as in [**24**]). It is however not quite clear, if it is not more efficient to just extend the factor base in the original setting. The method can only be efficient if different (non-sieve) effective smoothness test can be employed [**62**].
(9) *A different smoothness detection and factorization techniques.* Sieve methods for smoothness detection are quite convenient on the network of connected multipurpose computers. There are also special-purpose hardware designs that exploit the nature of sieving [**10, 100, 101**]. The NFS variant of Coppersmith requires a fast method of detecting factors below some fixed bound. A suitable method to apply is the Elliptic Curve Factoring method (ECM) [**71**]. Already a specialized hardware employing ECM (along with sieving) has been proposed [**42**]. A

different approach is to use factoring based on product/remainder trees [**13, 14**], but this approach does not yet seem practical.

6.2. Selection of the smoothness bound

For a successful NFS computation, we must find at least the same number of equations as the number of ideals having norms below B, plus some small number for unknowns from character maps. Gathering more than the minimal number of equations can lead to a more efficient linear algebra phase if Structured Gaussian Elimination is employed. For the sake of simplicity we assume that the number of required equations is B.

The probability of an algebraic number being B-smooth is given by the size of its norm. Let smoothness bound be chosen as $B = L_{p^n}(1/3, c_1)$, and let all norms of the sieved elements be bounded by $N_{\max} = L_{p^n}(2/3, c_2)$. Then the probability of element being B-smooth is given by $L_{p^n}(2/3 - 1/3, -c_2(2/3 - 1/3)/c_1)$ [**21**], with negligible error term for large enough p^6. We are sieving two fields at once for smooth equations. If we consider the smoothness probability in both fields to be independent, then we have to sieve on average $L_{p^n}(1/3, c_1 + (2/3)c_2/c_1)$ elements (all with norms below $N_{\max}$) to obtain the required B equations.

Minimal running time can be obtained if $c_1 = (2/3)^{2/3}$, and $c_2 = (2/3)^{1/3}$. In this case the NFS complexity is $L_{p^n}\left(1/3, 2(2/3)^{2/3}\right)$. This is obtained by choosing square region with each side equal to B, or skewed region with the same area. Norms in this region are below the bound $N_{\max}$ only in some special cases, such as SNFS, where special polynomial with small coefficients can be constructed. In the general case (GNFS) the constant c_2 must be chosen higher due to the limits imposed by the polynomial selection. The asymptotically optimal constants in this case are $c_1 = (8/9)^{1/3}$, $c_2 = (8/3)^{1/3}$, so the final complexity becomes $L_{p^n}\left(1/3, 2(8/9)^{1/3}\right)$.

The matrix constructed after the sieving phase is sparse with $O(B)$ non-zero elements. The asymptotic complexity of the linear algebra step is thus the same as of the sieving step. In practice it is easy to execute the sieving step in parallel, but not the linear algebra step. Thus we are forced to choose as low B as possible, and compensate the lower smoothness probability by sieving a larger sieve region. However, in a larger sieve region the norms of elements rise above limit $N_{\max}$. In our practical experiments, the norm growth in degree 6 fields is too steep, and we are unable to find enough smooth equations already for a halved smoothness bound.

Let us consider the choice of B in the case of $\mathbb{F}_{p^6}$ more precisely. Let polynomials on both sides have degree 6. The polynomial f_1 has only small coefficients (with a negligible effect on the norm), while the polynomial f_2 has a large coefficient of the order p. Elements in the form $x + y\alpha_1 + z\alpha_1^2$ have norms bounded by $\max\{x^6, y^6, z^6\}$. Elements in the form $x + y\alpha_2 + z\alpha_2^2$ have norms bounded by $\max\{x^6, py^6, p^2z^6\}$, if the size of the sieve region is large enough (see also the remarks in Section 8.2).

Let us suppose that we choose smaller smoothness bound $B = L_{p^6}\left(1/3, (2/3)^{2/3}\right)$, like in SNFS. If we sieve the classical 2D region, with $B \times B$ elements, norms in K_1 are bounded by $N_1(p) = L_{p^6}\left(1/3, 6(2/3)^{2/3}\right)$, which is well below the required

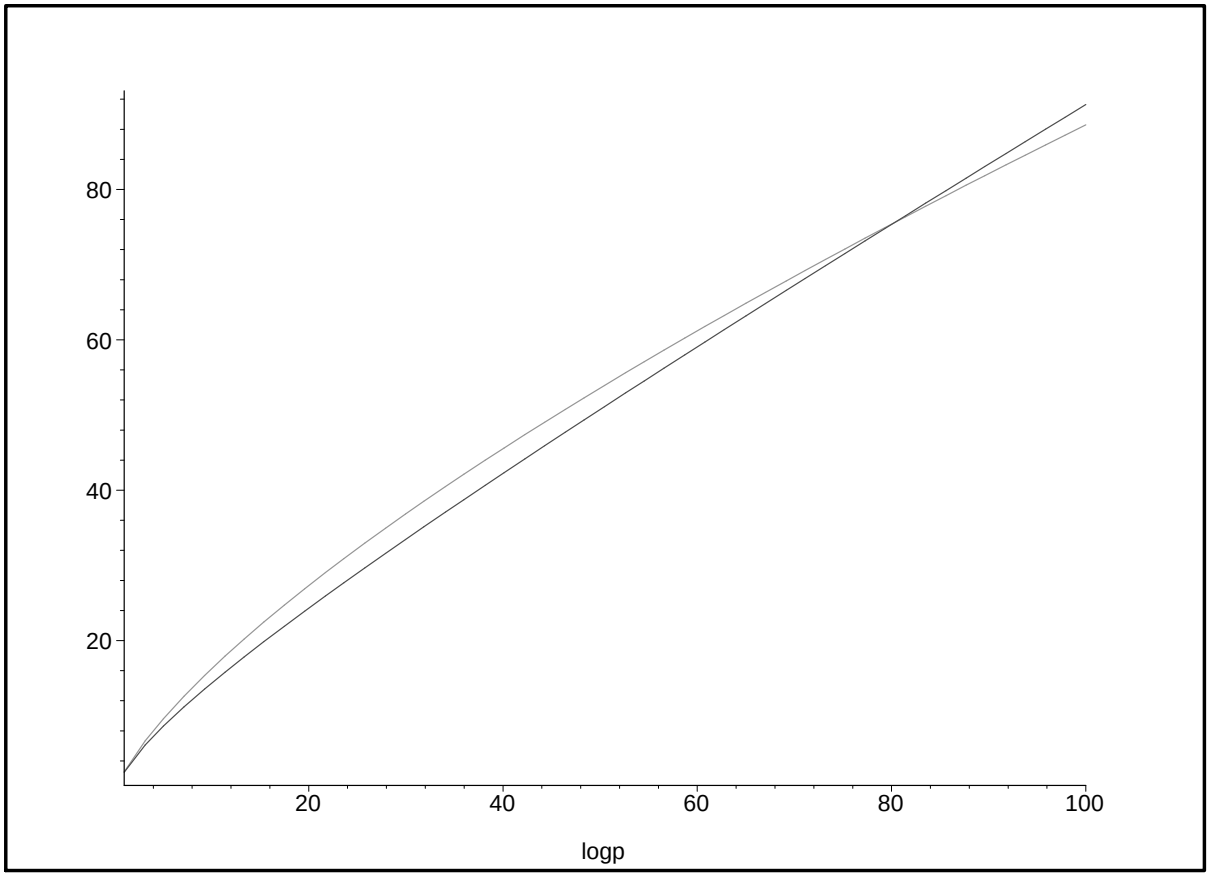

FIGURE 6.1. Log-log comparison of $M_2(p) = pL_{p^6}\left(1/3, 6(2/3)^{2/3}\right)$ (darker curve) and $L_{p^6}\left(2/3, (2/3)^{1/3}\right)$ (lighter curve) in logarithmic scale. Crossover point is near $p = 2^{80}$.

$L_{p^6}\left(2/3, (2/3)^{1/3}\right)$. Norms in K_2 are bounded by $N_2(p) = pL_{p^6}\left(1/3, 6(2/3)^{2/3}\right)$, which is above $L_{p^6}\left(2/3, (2/3)^{1/3}\right)$ for $p > 2^{80}$. For a very large p, this problem is compensated by a different polynomial selection algorithm. If we compare the function $N_2(p)$ with the function $L_{p^6}\left(2/3, (8/3)^{(1/3)}\right)$ (from GNFS settings), we get that the limit on norms is observed for $p < 2^{570}$, even with some additional tolerance (see Figures 6.1 and 6.2). On the other hand, even for small p's the SNFS bound is very tight, and as we have tried experimentally, the corresponding choice of B does not work in practice.

A better model is obtained, if we work with more than just norm bounds and asymptotic estimates. The smoothness probability in the sieve region is not distributed uniformly. Norms near the origin, and along specific lines determined by the real roots of sieve polynomials, are small, thus leading to a higher smoothness probability. On the other hand, norms in K_2 are already large near the origin leading to a lower overall smoothness probability than expected. The asymptotic estimate is not really accurate for small p's, as shown by experiments in Section 8.2.1. The recommended method to choose the best B is to conduct smaller preliminary sieving near the origin, as well as near the boundaries of the sieve region, and fine-tune the choice of B and the region size

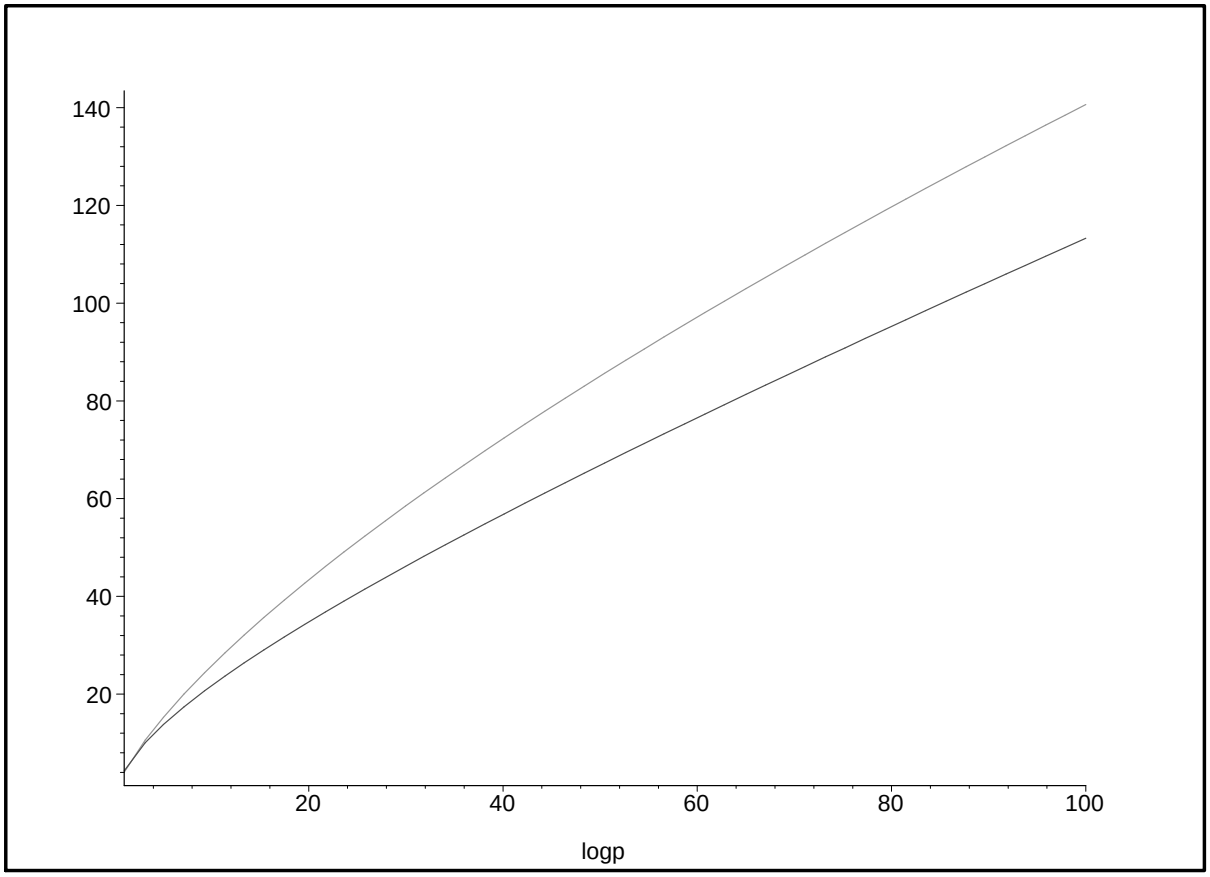

FIGURE 6.2. Comparison of functions $pL_{p^6}\left(1/3, 6(8/9)^{1/3}\right)$ (darker curve) and $L_{p^6}\left(2/3, (8/3)^{1/3}\right)$ (lighter curve) in logarithmic scale. Crossover point is near $p = 2^{570}$.

experimentally. However, in our experiments, we were unable to significantly decrease B from its original choice $B = L_{p^6}\left(1/3, (8/9)^{1/3}\right)$.

6.3. Remarks on polynomial selection

The polynomial selection is a critical part of the Number Field Sieve when used for factoring large integers. Special integers (e.g. Cunningham numbers, $C^{\pm}(b, n) = b^n \pm 1$), can be efficiently factored using Special Number Field Sieve (SNFS) with asymptotic complexity $L_n\left[1/3, (32/9)^{1/3}\right]$, i.e. the constant in the complexity formula of SNFS is $c \approx 1.523$. Classical version of the (General) Number Field Sieve method has $c \approx 1.923$, and NFS variant with multiple polynomials has $c \approx 1.902$ [**26**].

The different constant is due to a different possible polynomial selection. SNFS uses the specific form of the number being factored to construct a suitable sieve polynomial with only small coefficients. On the other hand, GNFS is limited in the polynomial selection phase to polynomials with larger coefficients. An optimal deterministic algorithm for choosing these polynomials is not known.

Classical method of polynomial selection is presented in [**20**]. Let N be a value we want to factor. Optimal degree of the sieve polynomial is given by

$$D = \left(3^{1/3} + o(1)\right) \left(\frac{\log N}{\log \log N}\right)^{1/3}. \tag{6.1}$$

In practical situations we usually ignore the $o(1)$ term and instead round the result to the nearest integer.

Set $m = \lceil N^{1/(D+1)} \rceil$, and write N in the base m:

$$n = c_D m^D + c_{D-1} m^{D-1} + \cdots + c_0.$$

This determines a polynomial $f(x) = c_D x^D + c_{D-1} x^{D-1} + \cdots + c_0$, with $f(m) = N$. For every coefficient $0 \leq c_i < m$.

A better method of polynomial selection for MPQS was presented by Murphy and Brent in [**79**]. It was further modified for GNFS by Kleinjung [**57**]. The experimental polynomial selection was used in 512-bit RSA factorization [**22**]. The method was fully applied in all subsequent GNFS factoring records, e.g. RSA-576 factoring record [**38**]. The largest coefficient of the polynomial in all these methods has an absolute value $O(N^{\frac{1}{D+1}})$.

Alternative method of polynomial selection for computing discrete logarithms in $\mathbb{F}_p$ was suggested by Joux and Lercier [**52**]. They do not recommend using "algebraic" and "rational" sides, i.e. one polynomial with degree D and another one with degree 1. Instead, two polynomials of degree D and $D+1$ are used. The polynomial of degree $D+1$ has extremely small coefficients, and the polynomial of degree D is computed using LLL algorithm. It has again coefficients of the order $O(p^{1/(D+1)})$. Degree D is chosen to be smaller than in the case with a rational side. Commeine and Semaev in [**24**] present a multiple polynomial variant of NFS, where they recommend another polynomial selection method with a rational side.

The algorithm to compute discrete logarithms in extension fields [**55**] uses another type of the polynomial selection. In extension fields we cannot use rational side. As we need (at least two) polynomials with a common root in $\mathbb{F}_{p^n}$, they must have a common irreducible factor of degree n. Let D be a degree of the polynomial computed from the field size p^n using equation (6.1). If $D \leq n$, we use two polynomials of degree n. One is any degree n polynomial $f_1(x)$ irreducible over $\mathbb{F}_p$ with extremely small coefficients. The other polynomial is simply one of the two polynomials $f_2(x) = f_1(x) \pm p$. If $D > n$, i.e. if p is too large, a polynomial selection involving lattice reduction can be used [**55**]. This polynomial selection works as follows:

(1) Select a polynomial f_0 of degree n with small coefficients.
(2) Choose a constant W, and let $f_1(x) = f_0(x + W)$.
(3) Polynomial f_2 of degree D with $f_1 | f_2$ is found by reducing the lattice (generating vectors are polynomial coefficients written in columns):

$$L = \begin{pmatrix} f_1(x) & x f_1(x) & \cdots & x^{D-n} f_1(x) & p & px & \cdots & px^D \end{pmatrix}$$

The optimal size of the coefficients in f_1, f_2 is achieved in the case when $W^n \approx p^{n/(D+1)}$. In our experiments $n = 6$. The need for larger degree D of sieve polynomials occurs only for very large p's that are out of our range for practically implementable experiments.

We have conducted a series of experiments to determine the optimal selection of the polynomial f_1, as described in Section 8.1. These results show that the selection of polynomial f_1 has a significant impact on the smoothness density in small regions near origin. However, the actual sieving region is much larger, and so the final impact of the choice of f_1 does not justify too complicated selection algorithm. Instead we concentrate on the polynomial f_2, and the simplicity of the implementation. The first selection criterium was that $[\mathcal{O}_{K_1} : \mathbb{Z}[\alpha_1]] = [\mathcal{O}_{K_2} : \mathbb{Z}[\alpha_2]] = 1$ (to avoid fractional ideals). In further practical experiments we have observed, that norms in $\mathcal{O}_{K_2}$ tend to grow faster if discriminant $D(f_2)$ is higher. As we needed to compute $D(f_i)$ anyway (to determine $[\mathcal{O}_{K_i} : \mathbb{Z}[\alpha_i]]$), we have used it as a second criterium for polynomial selection. From the set of suitable polynomial pairs (f_1, f_2) we choose a pair with the lowest $D(f_2)$.

6.4. Multiple polynomials

The two polynomials $f_2(x) = f(x) \pm p$ are not the only possible choice with comparable size of the coefficients. We can consider the choice of any $f_2(x) = f(x) + ph(x)$, with $\deg h < \deg f$, where coefficients of $h(x)$ are small in absolute value (typically equal to ± 1). On the other hand, the norms of algebraic numbers in fields defined by $h(x)$ with higher degree are larger (in absolute value).

These polynomials can be useful to implement a multiple polynomial variant of NFS. Classical multipolynomial NFS to solve discrete logarithms in $\mathbb{F}_p$ is described in [**24**]. It is possible to adapt the method also for $\mathbb{F}_{p^n}$. A general outline of the new method is as follows:

(1) Let $f(x)$ be a monic polynomial of degree n irreducible over $\mathbb{F}_p$. Let $K_0 = \mathbb{Q}(\alpha)$, with $\alpha \in \mathbb{C}$, $f(\alpha) = 0$.
(2) Let $f_i(x) = f(x) + ph_i(x), i = 1, 2, \ldots$ with $\deg h_i < \deg f$; all h_i's are distinct with small coefficients in absolute value. Let $f_i(\beta_i) = 0, \beta_i \in \mathbb{C}$, and let $K_i = \mathbb{Q}(\beta_i)$.
(3) For each $i = 1, 2, \ldots$ find points in the sieve region that correspond to smooth algebraic numbers in field K_0 as well as in the field K_i.
(4) If the number of smooth points accumulated using K_i is smaller than the size of the factor base in this field (plus small constant for logarithmic maps), the results of sieving K_i should be discarded.
(5) Create the system of equations similar to equation (5.9) with left-hand side corresponding to factorization of elements from K_0, and right-hand side to factorization of corresponding elements from K_i's which were not discarded in step 4.
(6) After computing the solution of the system of equations, individual logarithms of $\mathbb{F}_{p^n}$ can be computed using the descent method.

In the linear system unknowns correspond to prime ideals in K_0 and in every K_i used plus $O(1)$ logarithmic maps or virtual unit logarithms (we need at most n of them

for each K_i). Let the size of factor base for K_i be c_i. Suppose that we have r_i points $(a_0, \ldots, a_t)$ corresponding to elements $\sum a_j \alpha^j$, $\sum a_j \beta_i^j$ smooth in both K_0 and K_i. Then we can create r_i new equations at the cost of $c_i + n$ unknowns. If $r_i < c_i + n$, the NFS with the given K_i is unsuccessful. If for some j we get $r_j > c_j + c_0 + 2n$, we do not need more K_i's, just the K_j. Thus using multiple polynomials is only suitable if the expected NFS output is above the maximal c_i but below the minimal $c_i + c_0$.

As it is difficult to precisely estimate the NFS output, we should start with sieving K_1 given by one of the $f_2(x) = f(x) \pm p$. Using the number of smooth equations found, we can estimate the number of K_i's required. It should be taken into account that for $h_i(x)$ with higher degree, the norms of algebraic numbers are larger. Thus the expected number of equations found is lower. This can be compensated by setting different smoothness bound B for each K_i, or K_0 respectively.

CHAPTER 7

Remarks on sieving

The Number Field Sieve algorithm requires a fast identification of all B-smooth integers from within a larger set of (related) integers. A related task was solved already in Ancient Greece by Eratosthenes of Cyrene. Sieve of Eratosthenes [**50**] in its basic form allows enumeration of all primes up to a fixed bound N.

The classical method of Eratosthenes starts by writing all odd numbers from 3 up to N. The first new prime is 3, and all its multiples are crossed out. The first uncrossed number is the next prime, and the process is repeated until we identified all primes up to N. More modern variants are examined in [**89**]. Their complexity is a matter of time-memory trade-off. Either we use $O(N)$ within $O\left(N^{1/2}(\log\log N)/\log N\right)$ bits of memory, or $O(N/\log\log N)$ operations and $N^{1+o(1)}$ bits of memory. Atkin sieve [**8**] is a sieve variant based on quadratic forms with complexity further reduced to $O(N/\log\log N)$ additions and $N^{1/2+o(1)}$ bits of memory.

Enumeration of B-smooth numbers within N consecutive numbers (for $B < N$) can be implemented by an algorithm similar to the Sieve of Eratosthenes. We initialize the sieve array with N elements, such that i-th cell will have value i. We find the first value p, $1 < p < B$, and mark positions $p, 2p, \ldots$ Instead of crossing numbers, we divide out memory cell content by p (all its powers). At the end of the sieve, all cells with value 1 identify B-smooth numbers. The algorithm can be easily extended to any sequence of successive values of a polynomial, using the fact that $f(x) \equiv f(x+p) \pmod{p}$. A more complex description and algorithms can be found in the Chapter 3 of [**28**]. An evolution of the sieving algorithms in the context of integer factoring is described in [**88**].

7.1. Sieving techniques for the NFS

Sieving algorithm used in the NFS solves the following problem:

Problem 1 (2D sieve). Given a polynomial $F \in \mathbb{Z}[x, y]$, a smoothness bound B, and a sieve region $S \subset \mathbb{Z} \times \mathbb{Z}$, determine the set $\{(x, y) \in S | F(x, y) \text{ is } B\text{-smooth}\}$.

Transformation between classical NFS and Problem 1 is usually quite straightforward. If α is a root of monic irreducible polynomial f, then the norm of algebraic number $x - y\alpha$ is given as $N(x - y\alpha) = y^{\deg f} f(x/y) \in \mathbb{Z}[x, y]$. The same holds for the second algebraic number field used. If the NFS have rational side, values of polynomial $x - ym$ are tested for smoothness. Sieve can either be used only on one side of the equation, and the second side is tested by fast factoring methods (e.g. ECM, [**71**]). Or both sides can be tested by a single polynomial $(x - ym)y^{\deg f} f(x/y)$, or $y^{\deg f_1 + \deg f_2} f_1(x/y) f_2(x/y)$

respectively. It is also possible to directly construct a homogenous polynomial used for sieving, as described in Section 12 of [**20**].

An original implementation of the sieving algorithm within NFS is described in [**15**]. A good description of the sieve algorithm implementation can also be found in [**34**]. These are also good resources concerning many implementation heuristics, that can speed up the sieve, such as using logarithmic estimates, excluding small primes from the sieve, etc. We describe them in more detail in Section 7.2, with some remarks concerning the choice of the sieve tolerance (see also Section 8.3 for an experimental approach).

Some of these optimizations lead to use of not only smooth equations, but partially smooth equations with one or more large factors. These large factors are usually easily found as a by-product of the sieve. Large primes are often used in factoring experiments [**72, 19, 32**]. On the other hand, DLP records were computed without large primes using larger factor bases than usual [**52**]. We were trying to employ large primes as well (see Section 8.4.3), to reduce some complexity connected with large factor bases. However, the results were not satisfactory. Our remarks to the large prime method are summarized in Section 7.5.

In our experiments, we are using NFS to determine discrete logarithms in $\mathbb{F}_{p^6}$. As we show later (see Section 8.2), we were forced to abandon the classical 2D sieve as defined by Problem 1. In this case, we must sieve algebraic integers in the form $x + y\alpha + z\alpha^2$. However, their norm is again given by a polynomial function even if more complicated than before. In fact, we can generalize Problem 1 to any number of dimension:

PROBLEM 2 (d-dimensional sieve). Given a polynomial F in d variables with integer coefficients, a smoothness bound B, and a sieve region $\mathcal{M} \subset \mathbb{Z}^d$, determine the set $\{a = (a_0, a_1, \ldots, a_{d-1}) \in \mathcal{M} | F(a) \text{ is } B\text{-smooth}\}$.

A basic d-dimensional sieve algorithm is described in Section 7.3. A more practical variant used in actual NFS computations is described in section 7.4. This variant is similar to a classical 2D-sieve, which can help to extend the existing software and hardware sievers to accommodate higher dimensions.

On modern computer architectures, specific care must be taken when the algorithm often accesses the memory. The time required to access a particular memory element depends on the locality issues. Elements often accessed (or whole blocks) are loaded into the cache with faster access time. In the case of a random access across a larger memory region, the average time needed is higher. A sieve algorithm is very memory intensive, and the memory is accessed "pseudo-randomly", as the distance between consecutive marks depends on the prime used. There are many cache friendly implementation techniques that can reduce the average memory access time, such as specific block sieving [**108**], or sieving based on bucket sort [**7**]. We have implemented a similar scheme described in more details in Section 7.4.1.

A particularly important variant of the sieve is the lattice sieve. The first motivation was to skip pairs (x, y) with $\gcd(x, y) = 2$, as it can save 25 % of the work. This can be done by sieving only 3 sublattices with $x \equiv 1 \mod 2$, or $y \equiv 1 \mod 2$. In the sieve, we start with the first odd point divisible by p, and continue in the strides of doubled length $2p$. Pollard in [**86**] suggested a more efficient method: Sieve only a lattice where

every point has the norm divisible by a certain large q. Norms of the sieved elements can be reduced by the factor q, but they grow faster on the lattice. Thus the sieve region should be much smaller than in classical sieve, but multiple q's must be taken [**44, 12**]. A more efficient version of the lattice sieving was presented in [**39**].

There are other options how to find enough smooth elements for the NFS. A classical version of the sieve takes only one sieve polynomial F. Another possibility is to use multiple polynomial version of the GNFS [**26, 35**]. One approach is to combine sieve (on one side) with fast method of determining smoothness, e.g. ECM [**71**], or new method proposed by Bernstein [**13**]. In case of $\mathbb{F}_{p^6}$ DL computations, this can be quite a promising variant, due to a large discrepancy of smooth densities in the two algebraic fields used.

A specific related topic are the proposals of various specific sieving machines for factorization purposes, from TWINKLE [**99, 68**], Bernstein's proposal [**10**], to TWIRL [**100, 101**]. Recently more new designs were proposed: SHARK [**40**], combination of TWIRL and ECM [**42**], or a more realistic design of [**43**]. As the sieving part for factoring and discrete logarithm problems are essentially the same, it seems possible to quickly adapt such a machine to solve the equivalent DLP problem. In the case of XTR-DL, such machines should be adapted to a 3D sieve, but as we show in Section 7.4, the algorithm can be formulated in a very similar manner to a classical 2D sieve. Thus the machine design adaptation should be easy in comparison with the overall machine design effort.

7.2. Logarithmic estimates, small factors and tolerance

Although the asymptotic complexity of the sieve is already relatively small, the sieve region in the NFS is quite large, as well as the integers sieved. Effective implementation requires many optimizations. The computation of the norm can be a very costly operation, especially if higher degree polynomials are used. Another costly operation is the division, especially when norms concerned are large.

To get rid of the division, we can replace the sieved integers with their logarithms. Division of integers is changed to subtraction of real numbers. If we were working with exact real numbers (which is not really possible on the digital computer), we would compute the logarithm of the norm for each point in the sieve region and place it in the associated counter. During the sieve we would subtract logarithms of norms of prime ideals. At the end, each point with the counter containing 0 would have a smooth norm. Instead of computing with exact values of logarithms (norms inside the sieve region, and of the prime ideals), we round them to some precision. The logarithm base and rounding precision is usually chosen in such a way that an 8-bit arithmetic is sufficient[1]. Another convenient choice is to use the number of bits, $\mathrm{nb}(x) = \lfloor \log_2 x \rfloor + 1$. If we are already working with imprecise numbers, we can additionally relax also the computation of the norm. Instead of computing its exact value, we just try to estimate its logarithm by faster methods, e.g. by interpolation, etc. In our experiments, we have computed

[1]We have used full 32-bit integers, as there was enough memory, and access to 32-bit integers is faster than to 8-bit integers on modern architectures.

exact norms only near the origin, all other norm logarithms were estimated from the size of point coordinates, i.e. $\log |N(\sum a_i \alpha^i)| \approx 6 \log \max\{a_i\}$.

The sieve algorithm thus uses an array s, that is associated with a specific set A of algebraic numbers. For every $\xi \in A$, the corresponding array element is initialized by the estimated logarithm of its norm, i.e. $s[\xi] \approx |\log N(\xi)|$. During the sieve, we subtract $\log N(\mathfrak{p}_i)$ for each prime ideal $\mathfrak{p}_i$ from the factor base that contains ξ. We expect that the array value $\overline{s}[\xi]$ after the sieve is near zero for most of the smooth ξ's. If R_N is a random variable modelling rounding and norm estimation errors, and R_i are random variables modelling rounding errors for $\log N(\mathfrak{p}_i)$, then the final error is given by a random variable

$$R = R_N - \sum_{\mathfrak{p}_i | N(\xi)} R_i. \tag{7.1}$$

If we knew distributions of R_N and R_i's, we could create a statistical test with given error α to identify smooth values based on the realization of R. In practice, the situation is less complicated, as the rounding can be accustomed to a desired effect.

Suppose that we sieve with all prime ideals with the norm below B, and with all their powers. Suppose that the original norm estimate is an upper bound, i.e. $s[\xi] \geq \log N(\xi)$ for all ξ. Subtract values $\lfloor \log N(\mathfrak{p}) \rfloor$ during the sieve. Then every $\xi \in A$ with $\overline{s}[\xi] < \log B$ is B-smooth. However, not all smooth ξ's are detected, e.g. if the norm estimate is too high, or if cumulative rounding errors are greater than $\log B$.

The situation is different, if we use the lower bound for the norm, i.e. $s[\xi] \leq \log N(\xi)$, and we subtract values $\lceil \log N(\mathfrak{p}) \rceil$ during the sieve. Then every $\xi \in A$ that is B-smooth will have $\overline{s}[\xi] < \log B$. However, some of the ξ's with $\overline{s}[\xi] < \log B$ are not B-smooth, due to rounding errors, and possibly due to too low norm estimate. This situation is however more favorable as the previous one, because smooth numbers are scarce, and we do not want to miss any of them. A penalty in this case is the number of falsely reported smooth numbers, for which we must compute the exact norm and its factorization. As a side-product we can use almost-smooth numbers, that were found, in a large prime variant of NFS (see Section 7.5).

There are some other factors, that complicate the real-world situation:

(1) higher powers of ideals are usually not sieved at all,
(2) lower bound for the norm is harder to compute in some areas (corresponding to real roots of the sieve polynomial),
(3) we do not sieve with some of the primes (with small norms) for efficiency reasons. E.g. the number 2 divides every second point in the sieve region, number 3 every third, etc. Updating the sieve with small primes is thus very costly. On the other hand, their logarithms are very close to zero (can even be rounded down to 0).

So in fact, if we want to use these optimizations, we cannot avoid stochastic behavior of the sieving algorithm. The detection of the smooth numbers is based on the premise, that the smooth number is more likely to have small final $\overline{s}[\xi]$ than a general number, but it still can be greater than 0 (or $\log B$). In practice, we set a fixed tolerance value

T. We ignore all points with $\overline{s}[\xi] > T$ (even if there is a small chance they may be smooth). Points with $\overline{s}[\xi] \leq T$ are further processed, the exact value of the norm is computed and it is factored to find out, whether $N(\xi)$ is really smooth (this can be done in the postprocessing, after the whole sieve is executed, or in the second run of the sieve).

Let us examine the influence of leaving out small primes in more detail. If we sieve only with primes $p_i > B_{\min}$, then the final sieve value is $\overline{s}[\xi] = r + s_{B_{\min}}$. Here r is a contribution of rounding errors (a realization of the random variable R from (7.1) with contributions of all $p_i \leq B_{\min}$ left out), and $s_{B_{\min}}$ is an approximation of the logarithm of $B_{\min}$-smooth part of $N(\xi)$. We would like to impose some bounds on $s_{B_{\min}}$, so we can quickly identify (potentially) smooth numbers. We will simplify the situation by requiring $B_{\min} = 2^b$, for some small b.

Let m be a random k-bit integer, with large k. We denote by $s_b(m)$ a 2^b-smooth part of m. Let S_b be a random variable, and its value is the number of bits of $s_b(m)$ for m chosen from uniform random distribution (on the interval $[2^{k-1}, 2^k - 1]$). If we know the cumulative distribution function of S_b, we can choose error of estimate α, and set the tolerance T to $(1-\alpha)$-percentile of S_b (if we take only the influence of small primes below 2^b into account).

If $b << k$, we can expect that also $s_b(m) < 2^k$, and the parameter k can be safely ignored. For any m we can write

$$s_b(m) = \sum_{p_i^{e_i} | m} e_i \log p_i. \tag{7.2}$$

To find the distribution of S_b, we must compute $Pr\left(s_b(m) < x\right)$ for a randomly chosen integer m. We will simplify the situation by intensionally ignoring higher powers of small integers (i.e. set all $e_i = 1$). Let P_i be a probability, that prime p_i is a divisor of a random (large) integer m. Clearly $P_i = 1/p_i$. The average value of S_b is then

$$E(S_b) = \sum_{p_i < 2^b} \frac{\log_2 p_i}{p_i}. \tag{7.3}$$

To compute approximation of this sum[2], we can replace summation through primes by summation through all integers, if we use additional probability $1/\ln j$ (approx.) that a given integer j is prime:

$$E(S_b) \approx \sum_{j=2}^{2^b-1} \frac{\log_2 j}{j} \cdot \frac{1}{\ln j}. \tag{7.4}$$

We can model these two probabilities (integer is prime, and divides a random m) into a single random variable X_j. X_j has value 1 with probability $1/(j \ln j)$, and 0 otherwise. Thus the sum $\sum_{j=2}^{2^b-1} x_j \log_2 j$ for random realization x_j of X_j should approximately

[2]The similar idea is used in [**97**]

model the values $s_b(m)$ for a randomly chosen m, i.e.

$$S_b = \sum_{j=2}^{2^b-1} x_j \log_2 j. \tag{7.5}$$

Its mean is as derived before in equation 7.4. Moreover we can compute its variance as

$$\sigma^2(S_b) = \sum_{j=2}^{2^b-1} \sigma^2(X_j)(\log_2 j)^2 = \frac{1}{(\ln 2)^2}\left(\sum_{j=2}^{2^b-1}\frac{\ln j}{j} - \sum_{j=2}^{2^b-1}\frac{1}{j^2}\right). \tag{7.6}$$

By approximating sums with integrals, and simplifying the results by omitting insignificant terms we can find approximate values of (7.4) and (7.6) by

$$E(S_b) \approx (b-1) \tag{7.7}$$

$$\sigma^2(S_b) \approx \frac{(b-1)^2}{2} \tag{7.8}$$

A suitable well-known random distribution with the same mean and variance is Erlang distribution with parameters $k = 2$, $\lambda = 2/(b-1)$. Its cumulative distribution function (CDF) is

$$F(x; k, \lambda) = 1 - \sum_{i=0}^{k-1} e^{-\lambda x}(\lambda x)^i / i! \tag{7.9}$$

Using this formula, we can now compute e.g. that expected average contribution from small factors up to $2^7 = 128$ is 6 bits, and from small factors up to $2^8 = 256$ is 7 bits. The 99-percentile for S_7 based on this approximation is 19.9 bits, and for S_8 it is 23.2 bits.

We have made many simplifications in the process of approximation of S_b, as well as in choosing distribution function based on just the mean and variance. To justify the results, we performed a statistical experiment. 10000 integers of bit size 100, 200 and 300 were generated (with uniform random distribution). Smooth parts of integers divisible by primes only up to 128, and 256 respectively, were found by the trial division. A comparison of theoretical and practical results is summarized in Figure 7.1. The results for different bit sizes were very similar, thus were merged them under a single S_b result. The CDF of $\mathrm{Erl}(2, 2/(n-1))$ is a good approximation for the real distribution of S_b only in the upper part (for larger 2^b-smooth parts). Small smooth parts are affected by rounding of logarithms, and repeated factors, which were not considered in the estimate. As we are interested in the upper bound for smooth parts, we can conclude that the estimate is suitable to set the sieve tolerance.

When choosing the sieve tolerance, our goal is to minimize the number of smooth points that are incorrectly ignored (with $\overline{s}[\xi] > T$), while also minimizing the work-factor required for postprocessing. The cost of the post-processing is in practice influenced by

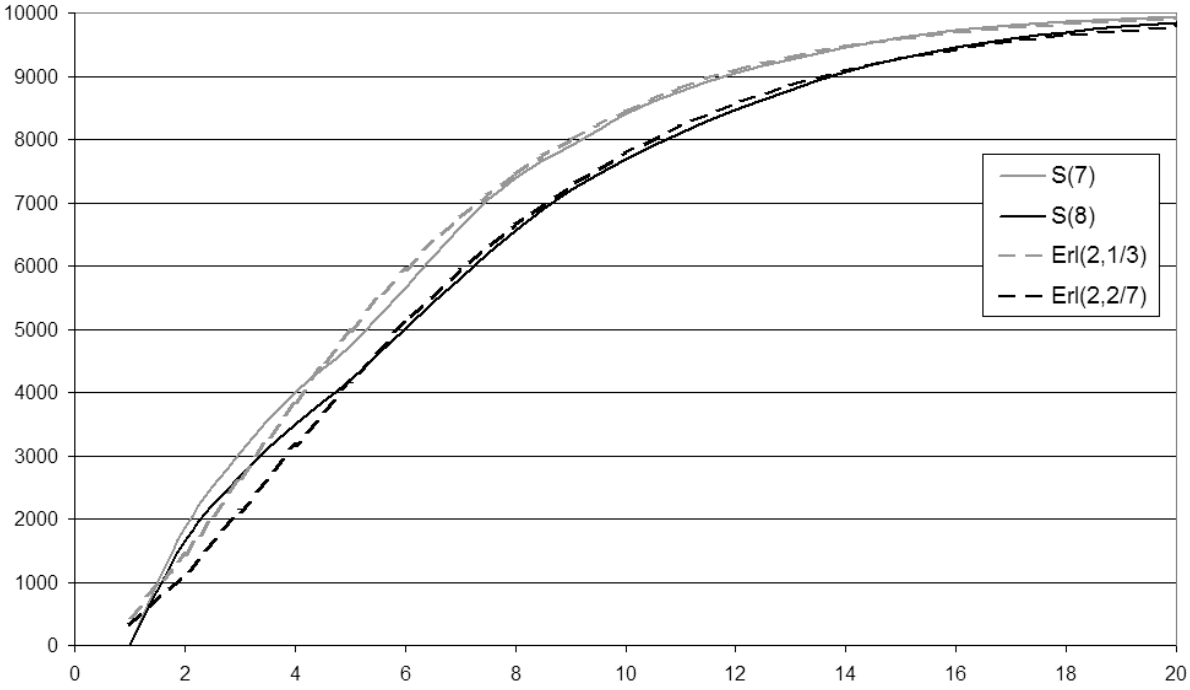

FIGURE 7.1. A comparison of practical values of S_b with theoretical approximation $\mathrm{Erl}\,(2, 2/(b-1))$. Chart depicts measured and expected fraction of 10000 numbers with smooth parts up to b bits, for $b = 7, 8$ respectively.

the cost of the norm estimates, the sieving, and the time spent on precise norms computation and factorization for false positives. These values are mostly implementation specific. In Section 8.3 we provide experimental results leading to our final choice of the tolerance value.

7.3. Generalized line sieve

Experiments show that in the case of $\mathbb{F}_{p^6}$ and a practical range of p's, the density of smooth algebraic numbers $a + b\alpha$ is not sufficient enough to lead to a solvable set of equations. We are then forced to use either special techniques to find and pair large factors (outside of the factor base), or to use algebraic numbers with higher degree. This means that we sieve algebraic integers in the form of a polynomial $\sum_{i=0}^{d-1} a_i \alpha^i$. The optimal dimension based on experiments seems to be $d = 3$ for $\mathbb{F}_{p^6}$ (see Section 8.2 for more thorough discussion). Different fields can have another optimal value for d, working hypothesis is that d should be near $n/2$.

If we want to effectively sieve algebraic numbers in higher dimensions, we need to adapt the sieving algorithm. Both the line and lattice sieve can be transformed to their d-dimensional variants.

Let us suppose, that we have enough memory to store $s[\xi]$ for the whole d-dimensional sieve region. We can find a (reduced) basis of d-dimensional subspace of $\mathbb{Z}[\alpha_i]$ for

every ideal from the factor base. Then we can enumerate all points of any given ideal within the sieve region and mark it within the d-dimensional array in the memory. We call this type of algorithm a full sieve. In practice, however, the memory size is limited and the random memory access is expensive (causing a number of CPU wait cycles). That's why various types of segmentations of the sieve region are required for effective implementation. A generalized line siever processes the d-dimensional space in a sequence of lines. The lattice sieve fixes a sublattice of the whole d-dimensional region. Then it sieves this sublattice either with a line siever (if the sublattice is large), or with a full siever if the sieve array for the whole sublattice can be fitted into the memory.

Lattice sieve in higher dimensions requires a fast algorithm to find a reduced basis of an ideal (within given sublattice/order). Deterministic algorithms (such as [**39**]) working in 2-dimensional space cannot be used for higher dimensions. We can however use LLL algorithm [**66**] for the reduction, but it is slower (see also Chapter 2 of [**23**]).

As the sieve based on a direct enumeration of points in ideals (within the sieve region) is not practical, we focus on extending the line sieve algorithm, to provide both scalable, cache friendly and distributable algorithm. First idea is a recursive implementation usable for general dimension that works by constricting the sieve space to subspaces of lower dimension until we are able to apply the line sieve. This algorithm is more suitable for higher dimension d and nearly hypercube-shaped region.

First, we must find all ideals over small primes by factoring a sieve polynomial $f(x)$ modulo primes $p_i < B$. Let

$$f(x) = f_{i,1}(x) \cdots f_{i,r}(x) \pmod{p_i},$$

where $\deg f_{i,j} = \delta_{i,j}$ are the degrees of respective prime ideals. Algebraic numbers lying in ideal $\mathfrak{p}_{i,j}$ corresponding to $f_{i,j}(x) = \sum_{k=0}^{d_{i,j}} b_k x^k$, can be written as a $\mathbb{Z}$-linear combinations of rows of the $n \times n$ matrix (where n is the degree of the number field/the degree of $f(x)$).

$$L_{i,j} = \begin{pmatrix} p & 0 & 0 & \dots & & & 0 \\ 0 & p & 0 & \dots & & & 0 \\ & & \ddots & & & & \\ b_0 & \dots & b_{\delta-1} & b_\delta & 0 & \dots & 0 \\ & & & & \ddots & & \\ 0 & \dots & 0 & b_0 & \dots & b_\delta & 0 \\ 0 & \dots & & 0 & b_0 & \dots & b_\delta \end{pmatrix}, \tag{7.10}$$

where $\delta = \delta_{i,j}$. The norm of the ideal $\mathfrak{p}_{i,j}$ is p_i^δ. Let us suppose that we want to sieve algebraic integers in the form $\sum_{i=0}^{d-1} a_i \alpha^i$, i.e. the d-dimensional subspace of $\mathbb{Z}[\alpha]$. Any ideal with $\delta > d$ will contain only those points from this subspace, that have all coordinates divisible by p. Thus we only use ideals with $\delta \leq d$ in the sieve. Factor base is a set of all these ideals.

We associate a single d-dimensional vector with each ideal in the factor base. It represents a chosen point of the associated ideal. During the sieve, we will constrict d-dimensional space into $(d-1)$-dimensional, etc., until we reach a single line. The idea of the algorithm can be written in a recursive way:

ALGORITHM 2. Generalized line sieve

INPUT: Set $\mathcal{B}$ of $\mathfrak{p}_{i,j}$, sieve region bounds M, tolerance T.

OUTPUT: Set $\mathcal{S}$ of probably smooth points.

Init: (a) Compute HNF $L_{i,j}$ of each $\mathfrak{p}_{i,j}$ in factor base.
(b) Compute approximation of the logarithm of norm of $\mathfrak{p}_{i,j}$, $l_{i,j} = \lceil d_{i,j} \log_2 p_i \rceil$.
(c) Associate a vector $v_{i,j} = (0, 0, \ldots, 0)$ with each $\mathfrak{p}_{i,j}$. Denote a set of all these vectors V.
(d) Let $\mathcal{S} = \emptyset$.

Recursion: Begin with dimension $D = d$, set $V_D = V$.

(1) **Reduction:** For each vector $v_{i,j} \in V_D$ add such multiple of D-th row of $L_{i,j}$, so that $\mathrm{proj}_D(v_{i,j})$ is the least possible in sieve region.
(2) **Line sieve:** If $D = 0$, do the line sieve.
   (a) Let v be a vector such that for each $v_{i,j} \in V_0$, $\mathrm{proj}_k(v_{i,j}) = \mathrm{proj}_k(v)$, for every $0 < k \leq n$.
   (b) Let s be an array of size given by M. For each array index x, let $\theta_x = x + \sum_{k=1}^{n} \mathrm{proj}_k(v)\alpha^k$. Let $s[x] = \lfloor \log_2 N(\theta_x) \rfloor$.
   (c) For each $v_{i,j} \in V_0$: Let $x_0 = \mathrm{proj}_0(v_{i,j})$. Subtract the value $l_{i,j}$ from every array element $s[x_0 + mp_i]$ in sieve bounds.
   (d) For each array index x: If $s[x] < T$, let $\mathcal{S} = \mathcal{S} \cup \{\theta_x\}$.
(3) **Sieve update:** If $D > 0$, let x_D run through all values given by M. For each x_D do:
   (a) Let $V_{D-1} \subset V_D$ such that for each $v_{i,j} \in V_{D-1}$: $\mathrm{proj}_D(v_{i,j}) = x_D$.
   (b) Take *recusive step* with $D - 1$ and V_{D-1}.
   (c) For each updated $v'_{i,j} \in V_{D-1}$ add D-th row of $L_{i,j}$ to a corresponding $v_{i,j} \in V_D$.

The algorithm recursively searches subspaces with decreasing dimension. For each ideal we maintain a working vector $v_{i,j}$, and we are fixing and updating its coordinates from the highest dimension. For further processing only those vectors are used, which have some point in given fixed subspace. This is tested by checking the coordinate x_D. Working vector is initialized in dimension D to a minimal value, which can be computed from M modulo p_i or a_d (depending on the actual dimension and the ideal degree, see the **Reduction** step). Due to a lower triangular HNF matrix of the ideal, by changing lower dimensions of $v_{i,j}$ we leave higher dimensions intact.

When the dimension 0 is reached, only the last coordinate is not fixed. This means we have reduced the sieving area to a single line and we can proceed with the classical line sieve. We initialize the array s with norm estimates for every point on the sieve line. From this array we subtract logarithms of ideals that have intersection with the sieve line (active ideals). After all active ideals are processed, we compare the final value of s with the tolerance bound T. Only those algebraic numbers on the sieve line that are

below tolerance limit are sent for further processing. The sieve is stopped either after enough smooth numbers is found, or after the whole region is sieved.

7.4. Implementation of the 3D sieve for XTR-DL solution

When implementing the sieve for XTR-DL solution we can use some specific facts that are not addressed by the algorithm from Section 7.3. A sieve region is only three dimensional, making a focus on generalization for a general dimension unnecessary. In fact, a recursive iteration over dimensions represent unnecessary overhead. Moreover, experiments with the general algorithm implementation have confirmed the intuition[3] that ideals of higher degree do not play an important role in NFS, and thus can be ignored during the sieve.

The norms of the elements $\sum_{i=0}^{d-1} a_i \alpha^i$ of the sieve region can be bounded (in general) by $(n+d)^{(n+d)} M_a^n M_f^d$ [**55**], where M_a is the maximum of absolute values of a_i, and M_f is the maximum of absolute values of coefficients of sieve polynomial. As the largest coefficient of f_2 was chosen to be $\pm p$, norm bounds quickly rise with the dimension of the sieve. For optimal performance the sieve region should be chosen to be skewed (see Section 8.2), such that $M_i/M_{i+1} \approx p^{1/n}$, where M_i is the maximum of $|a_i|$ sieved in the dimension i. Thus the sieve region contains the longest lines along the dimension 0, and should be sieved along these longest lines by the line sieve.

Some new implementation problems arise when sieving higher degree elements. Higher degree elements can generate principal ideals divisible by prime ideals of higher degree than 1. The higher degree factors are however very rare. Prime ideal over p_i of degree t has norm p_i^t. The chance of the factor with norm N appearing in factorization is $\sim 1/N$. Thus ideal over p_i of degree t has $1/p_i^{t-1}$ smaller chance to appear than degree one ideal over p_i. Usually only higher degree ideals that appear are those lying over small primes. These small primes are already excluded from the sieve for efficiency reasons. We can also exclude all higher degree prime ideals without a serious impact on the number of smooth algebraic equations collected. However, there is an easy way to detect a degree 2 ideal excluded in 3D sieve. If the remainder of the norm, after dividing out small primes and sieved primes, is less than B, then clearly the remainder is caused by excluded degree 2 ideals (it must be square). Moreover, if the remainder is less than B^2, and is square, it is caused by the excluded degree 2 ideal(s).

In a multidimensional sieve we must furthermore check that the sieved element, when taken as polynomial $\sum_{i=0}^{d} a_i x^i \in \mathbb{Z}[x]$, is irreducible over $\mathbb{Z}$. Otherwise we can take the corresponding factors and write equations directly for them (if their product is smooth, then they are certainly smooth as well). If dimension d is fixed, we can leave out the irreducibility check. The problem is that in that case we do not know exactly how many equations are required for the linear system to be solvable. In the worst case, when $d = 3$ every third equation can be a linear combination of previous 2 equations.

[3]The ideal of degree d over p have norm p^d, and thus has $1/p^{d-1}$ smaller probability to appear in any equation than the ideal of degree 1 over p.

In the sieve algorithm we use the Hermite Normal Form (HNF) of the ideal base[4]. Excluding small finite number of cases, all remaining bases of d-dimensional subspace have form[5] (base vectors in rows):

$$\begin{pmatrix} p_i & 0 & 0 & \dots & 0 \\ r_1 & 1 & 0 & \dots & 0 \\ r_2 & 0 & 1 & \dots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ r_{d-1} & 0 & 0 & \dots & 1 \end{pmatrix}. \tag{7.11}$$

Let us find some point $A_0 = (a_0, a_1, \dots, a_{d-1}) \in \mathfrak{p}_i$ (ideal $\mathfrak{p}_i$ has the basis as above). All points $A_k = (a_0 + kp_i, a_1, \dots, a_{d-1})$ also belong to ideal $\mathfrak{p}_i$. Fixing $a_1, \dots, a_{d-1}$ we define a sieve line (linear subspace of sieve region of dimension 1). We initialize each sieve line by logarithmic norm estimates. For various $\mathfrak{p}_i$ we find the "starting point" A_0 on the sieve line, such that $a_0 - p_i < m_0$, where m_0 is the minimum of possible a_0's on the sieve line, i.e. the point $(a_0 - p_i, a_1, \dots, a_{d-1})$ is outside the sieve region bounds. Then we make "jumps" of size p_i through points A_k, $k > 0$, until we find point such that $(a_0 + kp_i, a_1, \dots, a_{d-1})$ is already outside the sieve region. On every jump we "mark" points A_k by subtracting $\log p_i$ from logarithmic norm estimate stored at point A_k. After processing all p_i's, those points on the sieve line, whose remaining logarithmic estimates are near to zero, are considered to be smooth, and sent for further processing.

Consider now the situation that we have processed all prime ideals for the given sieve line. Next sieve line can be obtained by changing some a_j to $a_j + 1$, $j > 0$. Then we must certainly mark points in the form $(a_0 + r_j + kp_i, a_1, \dots, a_j + 1, \dots, a_{d-1})$, for k in some interval. If the starting point on the line given by $a_1, \dots, a_j, \dots, a_{d-1}$ was $m_0 + s$, then the new starting point on the line given by $a_1, \dots, a_j + 1, \dots, a_t$ is $m_0 + (s + r_j)$ mod p_i (if m_0 is fixed, it is easy to change the formula for variable m_0).

If the sieve region is cuboid, we can use counter approach and change all coordinates in previous dimensions when changing starting coordinates in higher dimension. Let m_j, M_j be the smallest and highest coordinate in dimension j, respectively. After we sieve line $M_1, \dots, M_{j-1}, a_j, \dots, a_{d-1}$, we change coordinates immediately to line $m_1, \dots, m_{j-1}, a_j+1, \dots, a_{d-1}$. This is accomplished by changing the HNF from equation (7.11) to the form:

[4]We are using ideal base in $\mathbb{Z}[\alpha]$, but we can find the HNF also in a case of a so called special-q sieve. In this case we choose a specific ideal $\mathfrak{q}$ with large norm q. Points of $\mathfrak{q}$ form a module Q (a subset of $\mathbb{Z}[\alpha]$). All points of Q are guaranteed to have factor q in its norm, but these norms grow faster. For every ideal $\mathfrak{p}_i$, we can find the basis of $\mathfrak{p}_i \cap \mathfrak{q}$, and its HNF. We can then sieve module Q in a similar manner then $\mathbb{Z}[\alpha]$, using new HNF's of small prime ideals. Special-q sieve is mainly used to compute individual logarithms, see Section 5.2.4.

[5]This is a special case derived from equation (7.10) for degree one ideals, which are the only ones used in the sieve.

$$\begin{pmatrix} p_i & 0 & 0 & \dots & 0 \\ r'_1 = r_1 \mod p_i & 1 & 0 & \dots & 0 \\ r'_2 = r_2 - r_1(M_1 - m_1) \mod p_i & -(M_1 - m_1) & 1 & \dots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ r'_{d-1} = r_{d-1} - \sum_{j=1}^{d-2} r_j(M_j - m_j) \mod p_i & -(M_1 - m_1) & -(M_2 - m_2) & \dots & 1 \end{pmatrix}. \tag{7.12}$$

By adding an appropriate vector all higher line coordinates get properly updated. We only need to change the starting point with the same formula, but using new r'_i, for every ideal.

The algorithm can be outlined as follows:

(1) For every ideal $\mathfrak{p}_i$ compute starting point $m_0 + s_i$ on the line $a_1 = m_1, \dots, a_{d-1} = m_{d-1}$.
(2) Sieve line $a_1, \dots, a_{d-1}$ by marking $m_0 + s_i + kp_i$ within the sieve region.
(3) If $a_1 < M_1$ update $s_i = (s_i + r'_1) \mod p_i$. Goto STEP 2.
(4) If $a_{d-1} = M_{d-1}$, STOP the sieve.
(5) If $a_1 = M_1$ find the first $a_k < M_k$, increment $a_k = a_k + 1$, update $s_i = (s_i + r'_k) \mod p_i$ and reset $a_1 = m_1, \dots, a_{k-1} = m_{k-1}$. Goto STEP 2.

The complexity of the algorithm consists of the number of sieve updates on a line, and the number of starting point updates when moving between lines. From the implementation point of view, the optimal running time is achieved when $M_0 - m_0 \geq B$. Then every ideal must have at least one point on every sieve line, thus the update steps are never "wasted".

The algorithm can be distributed on more computers by partitioning the sieve region into smaller subregions. The partitioning can be done in the highest sieved dimension. If we have more computers than $M_{d-1} - m_{d-1}$ we can fix the highest dimension on each computer and only sieve remaining $d - 1$ dimensions. If we want to balance the sieve workload on N computers, we can also sieve the region by strides. Instead of updating highest dimension by 1, we update it by N, initializing all computers with sieve lines with $m_{n-1}, m_{n-1} + 1, ...m_{n-1} + N - 1$ as their highest coordinate.

7.4.1. Block sieving. To further optimize our implementation, we wanted to remove the inner loop from STEP 2 of the sieve. We have excluded small primes from the sieve, thus we use only primes $B_1 < p_i < B$. Each sieve line was further subdivided into smaller blocks of size $2^b \leq B_1$. Thus, in every block, we expect only a single mark per each prime ideal. If properly implemented (see below), this change of the sieving algorithm can eliminate the inner sieve loop (STEP 2) executed for every prime in traditional implementations, thus reducing the possible pipeline stalls on processor.

We have further used the division to blocks to speed up norm computations. Only one logarithm of the norm is computed for the whole block if the number of bits of x_s (starting x-coordinate), and x_e (ending x-coordinate) resp. are the same. Otherwise

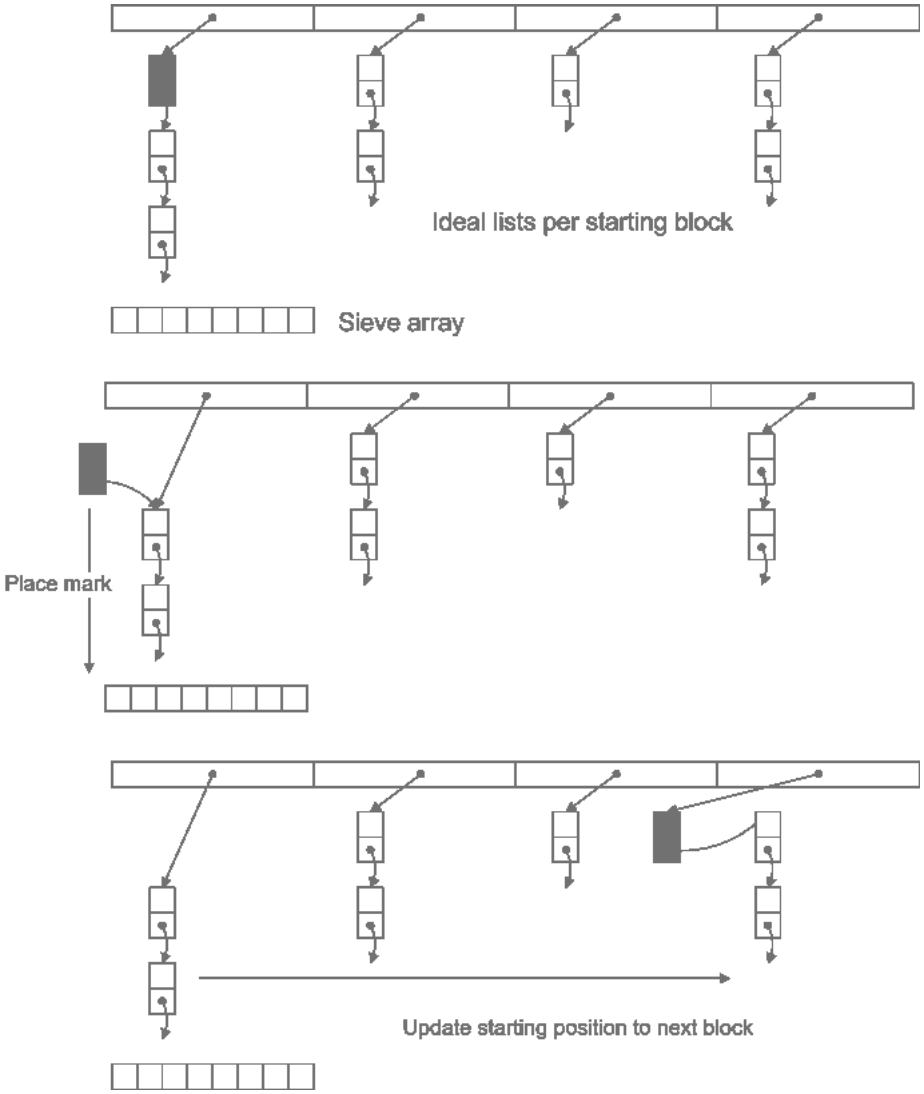

FIGURE 7.2. Sieving algorithm with ideal lists. Each mark in sieve requires 3 memory updates: Detach ideal from list do not update memory, only a pointer in register, placing mark requires 1 update, attaching ideal to new list requires update of 2 pointers. If number of blocks and sieve array size are carefully balanced, all memory updates are local (in cache). The algorithm seems faster in practice than classical sieve (too small ideals are excluded).

the logarithm of the norm is interpolated between the end-points, or computed exactly if the norm change is too large (e.g. if $x_s = 0$).

Let each line have size $l = k \cdot 2^b$, where 2^b is the block size, and the number of blocks is k. Each block has an associated array of 2^b elements initialized with logarithms of norms. We only store one such an array at a time, and reuse it for each block.

For each ideal we compute its starting point on the current line s_i. The first block we must mark is clearly $c_i = \lfloor s_i/2^b \rfloor$ (if we index blocks from 0). We must subtract a

logarithm of p_i from the array offset $d_i = s_i \mod 2^b$ in block c_i. Instead of doing it immediately, we store c_i, and d_i with further information about the ideal $\mathfrak{p}_i$. For each block c_j, we create a set C_j of ideals (in practice a dynamic list), that must be processed when we are working with this block.

After we have have computed all c_i, d_i, and filled sets C_j, we process individual blocks. In each block c_j, $j = 0, 1, \ldots, k-1$, we initialize the sieve array with logarithms of norms. For each ideal $\mathfrak{p}_i \in C_j$ we subtract the logarithm of p_i from the sieve array at offset d_i. Then we update the position of the ideal to $c_i' \cdot 2^b + d_i' = c_i \cdot 2^b + d_i + p_i$ (note that $c_i = c_j$). If $c_i' \geq k$, then we have finished sieving the line with $\mathfrak{p}_i$. Otherwise we store the new d_i', and move $\mathfrak{p}_i$ from set C_j to set $C_{c_i'}$. We implement this moving by using dynamic lists, as denoted on Figure 7.2. It requires 3 memory updates for each sieve mark. However, the sieve array is now very small. No inner loop for determining positions is required, we only process a limited list of ideals for each block. Final sieving algorithm is thus as follows:

ALGORITHM 3. 3D-block sieve

INPUT: Factor base $\mathcal{B}$, sieve region bounds $[m_0, M_0] \times [m_1, M_1] \times \cdots \times [m_d, M_d]$, tolerance T, block size 2^b.

OUTPUT: Set $\mathcal{S}$ of probably smooth points.

(1) **Init:** For each $\mathfrak{p}_i \in \mathcal{B}$:
   (a) Compute HNF of $\mathfrak{p}_i$, equation (7.11).
   (b) Compute and store $r'_{i,k} = r_{i,k} - \sum_{j=1}^{k-1} r_{i,j}(M_j - m_j)\ L_{i,j}$, equation (7.12).
   (c) Compute s_i, such that $s_i + \sum_{j=0}^{d-1} m_j \alpha^j \in \mathfrak{p}_i$.
   (d) Let $k = (M_0 - m_0)/2^b - 1$. Let $a_j = m_j$ for $j = 0, 1, \ldots d-1$.

(2) **Block init:**
   (a) For $j = 0, 1, \ldots k$, let $C_j = \emptyset$.
   (b) For each $\mathfrak{p}_i$: Compute $c_i = \lfloor s_i/2^b \rfloor$, $d_i = \lfloor s_i/2^b \rfloor$. Let $C_j = C_j \cap \{< \mathfrak{p}_i, d_i >\}$.

(3) **Block sieve:** For $j = 0, 1, \ldots k$:
   (a) Create sieve array s, and for $x = 0, 1, \ldots, 2^b - 1$: let

$$s[x] \approx \log \left| N\left(x + j2^b + \sum_{i=1}^{d-1} a_i \alpha^i \right) \right|.$$

   (b) For each $< \mathfrak{p}_i, d_i > \in C_j$:
      (i) Let $s[d_i] = s[d_i] - \log p_i$.
      (ii) Let $c_i' 2^b + d_i' = j2^b + d_i + p_i$, and $d_i' < 2^b$.
      (iii) If $c_i' < k$: Let $C_{c_i'} = C_{c_i'} \cap \{< \mathfrak{p}_i, d_i' >\}$.
   (c) For $x = 0, 1, \ldots, 2^b - 1$: If $s[x] < T$, let

$$\mathcal{S} = \mathcal{S} \cap \{x + j2^b + \sum_{i=1}^{d-1} a_i \alpha^i\}.$$

(4) **Line updates:**
   (a) If $a_1 < M_1$ update $s_i = (s_i + r'_1) \mod p_i$. Goto STEP 2.
   (b) If $a_{d-1} = M_{d-1}$, RETURN $\mathcal{S}$.

(c) Find l, such that $a_l < M_l$ and $a_j = M_j$ for every $j < l$.
(d) For each $\mathfrak{p}_i$: Update $s_i = (s_i + r'_{i,l}) \mod p_i$.
(e) Let $a_1 = m_1, \ldots, a_{l-1} = m_{l-1}$, and $a_l = a_l + 1$. Goto STEP 2.

The block size influences the final running time of the algorithm, but it depends on a computer architecture. In our experiments, the final size of the block was fixed to 2^7. This value was determined experimentally (on a computer with 512 KB cache size and 64-bit AMD architecture), leading to fastest sieving for both $B_1 = 128$ and $B_1 = 256$, and for various NFS setups. Experiments show approx. 25 % speedup using this optimization in comparison to sieving the whole lines at once. Small changes of the block size parameter around $b = 7$ can lead to only ± 1 % change of the total sieve time, so it is better to leave this parameter fixed, instead of fine-tuning it for each NFS experiment.

7.4.2. Further implementation remarks. We have used the sieve with $d = 3$ for all further XTR-DL NFS computations described in Section 8.4. Coordinates were denoted x, y, z, instead of an indexed notation. A region was cuboid with coordinates $[-X, X-1] \times [-Y, Y-1] \times [1, Z]$. The z coordinate should not be negative, as multiplying the elements by -1 leads to the same equation. X and Y were chosen as powers of 2 to simplify some of the modular arithmetic.

The sieve program integrates the possibility to execute the special-q sieve. This is required for the individual logarithm phase, when we need to find semi-smooth equations with a fixed single ideal $\mathfrak{q}$. This ideal determines the lattice Q to be sieved. For every ideal $\mathfrak{p}_i$ we compute a basis of $\mathfrak{q} \cap \mathfrak{p}_i$ (within Q). The line siever is then applied with these new bases.

If we find a potentially smooth point in the actually sieved block, we re-sieve the block again, and store also exact factors for potentially smooth points. Furthermore, we compute the exact norm for each of them, divide it by known factors from the second sieve and trial-divide out small factors below 2^7. Points are reported either if their norm is smooth, or when the remaining large factor is below a given bound B_1 (see Section 7.5). After sieving each plane, siever also reports total number of smooth (or semismooth) equations found, and timing information.

Results of one of the executed experiments are summarized in Table 7.1. As expected, the NFS output is highest for $z = 1$. It decreases approximately with the factor $1/2 \log z$. The NFS output is lower, when z has small prime factors, because we have removed points with $\gcd(x, y, z) > 1$. Interesting fact is that we get significantly more equations for $z = 1$ than for $z = 0$ (only points corresponding to irreducible polynomials over $\mathbb{Z}$ were used). This could lead to some optimizations even in existing sieves.

7.5. Large prime variant

In some cases, we have "almost" reached the norm, up to some large factor n. If all primes below B were used in the sieve, then certainly $n > B$. If also $n < B^2$, then it is clearly a prime. Let $B < B_1 \leq B^2$. If $B_1 \leq n < B_1^2/B$, then either n is prime or n has two (not necessarily distinct) prime factors $B < p_1, p_2 < B_1$. We can thus identify some

TABLE 7.1. NFS output by z. NFS parameters were: $B = 80000$, $f_1(x) = x^6 - 2x + 2$, $f_2(x) = x^6 - 2x - 529041$. Sieving region was $[-2^{16}, 2^{16}] \times [-2^{12}, 2^{12}] \times [1, 256]$, and for comparison corresponding (x, y)-halfplane with $z = 0$, $y > 0$. Total NFS output was 29477 equations in 15642 unknowns.

z	NFS output
0	303
1	1103
2	584
3	724
4	463
5	654
6	323

z	NFS output
250	31
251	46
252	22
253	50
254	35
255	40
256	23

additional factors almost for free. Single large prime requires one additional comparison. Composite factors require primality testing, and factoring of relatively small number, which are both fast.

These large factors are obtained as a side effect almost for free, but their effective use requires some further post-processing. Every large factor represent one new unknown in the linear system. Thus only large factors that occur more than once are usable.

The customary method that combines both large primes variant and large factor base is to use two-stage sieving. We use two smoothness bounds $B, B_1 < B^2$. First, a classical sieve is applied for every (degree one) prime ideal over $p_i < B$. Large primes below B_1 are identified (both single and double). If we do not have enough B-smooth equations, large primes that occur at least twice are added to the factor base, along with corresponding equations. If we still do not have enough equations, we can use special-q lattice sieve also for remaining single large primes. This means, we construct a lattice corresponding to a prime ideal over large q_i, and sieve elements on this lattice. Norm of every algebraic number associated with points on this lattice certainly has q_i as its factor. We can even skip medium step and directly use special-q sieve for every prime $B < q_i < B_1$.

For an easy detection of large primes we should use $B_1 < B^2$ for a single large prime, or $B_1^2 < B^3$ for a double large prime variant. Practical experiments show that the upper bound is too large, if we want to avoid special-q sieve. With larger B_1 we gain more partial equations, but most of them are useless, since the corresponding prime ideals appear only in a single equation. Recommended practical choice is $B^{1.2} < B_1 < B^{1.4}$ [**65**].

Standard approach of using large primes as a byproduct of sieve is however quite ineffective (see our experiment in Section 8.4.3), due to a very small probability of the repeated large prime factor. On the other hand, large primes can be used if optimal factor base size is too large:

(1) Compute the optimal factor base size B, and choose some $B_1 = B^c$, $c < 1$.
(2) Sieve the sieve region with the size based on B using a reduced factor base $\mathfrak{B}_1$ bounded by B_1.

(3) Collect partial equations with large primes $B_1 < p_i < B$.
(4) Count the number of occurrences of large primes, denoted by $n(p_i)$.
(5) Remove all partial equations, with all large primes having $n(p_i) = 1$.
(6) Add all remaining p_i's to new factor base $\mathfrak{B}_2$.
(7) For each remaining p_i's with $n(p_i) = 1$ try to find more $\mathfrak{B}_2$-smooth equations by special-q sieve.

The algorithm can stop (with success) after step 2 if $f > |\mathfrak{B}_1| + o(1)$, where f is number of full relations. We can also abort the algorithm after step 5, if $p+r+|\{p_i|n(p_i) = 1\}| < |\mathfrak{B}_2| + o(1)$, where p is number of remaining partial equations. The second condition is based on the fact that if only a single p_i-partial equation was found using regular sieve with smoothness bound B_1, then it is not reasonable to expect more p_i-partial equations with the same sieve bound. However, with the new factor base we hope for at least one new $\mathfrak{B}_2$-smooth equation (on average) per each of the special sieves.

An alternative approach can use the fact that each ideal corresponding to a large prime can be represented as $\mathbb{Z}$-module. Short vectors of this $\mathbb{Z}$-module (found by LLL), and some of their linear combinations, can provide us with the required new $\mathfrak{B}_2$-smooth equation. We can even use higher dimensional $\mathbb{Z}$-module to find elements of $\mathfrak{p}_i$ with very small norm. Use of higher dimensions in LLL reduction also eliminates the possible collisions between existing p_i-partial equations and constructed equations.

CHAPTER 8

Experimental results

The main goal of our research was to find XTR discrete logarithms using the Number Field Sieve algorithm, or a new algorithm with similar or lower complexity. However, we suppose that a new algorithm to solve XTR-DL (or DLP/IFP in general) would require a (radically) new approach from some new mathematical ideas. Although we are currently limited by the asymptotic complexity of NFS, a practical implementation has still many challenging points that are not well understood. We have made many experiments while preparing and later fine-tuning the implementation. Many experiments just confirm the known theoretical results. Others are very important for the best performance, especially where $o(1)$ asymptotics or implementation details come to play. In this chapter we present our various experimental results from the preparation of NFS, as well as results of concrete NFS computations.

8.1. Influence of the polynomial f_1 on the smoothness density

Algorithms for the polynomial selection, as discussed in Section 6.3, are concerned with the size of coefficients as the most important factor for the selection. However, in our $\mathbb{F}_{p^6}$ experiments, there is a large number of candidate polynomials f_1 (see Table 8.1). On the other hand, the size of coefficients of polynomial f_2 is fixed. There arise new unanswered questions: What is the optimal choice of the polynomial f_1? Does the polynomial selection have any significant influence on the sieving algorithm? How can we choose the best polynomial f_1? As the number of possible polynomials is large, we have tried to find some of the answers experimentally. These results were also published in [**116**].

Our goal in the NFS is to find enough B-smooth pairs, when sieving algorithm is applied over a fixed area in $\mathbb{Z}^d$. A common choice is to sieve points $(a, b) \in \mathbb{Z}^2$ corresponding to pairs $(a + b\alpha, a + b\beta)$, with $0 < |a|, b \leq M$, $\gcd(a, b) = 1$. Another possibility of sieving points of $\mathbb{Z}^d$ with $d > 2$ is shown in Chapter 7.

Parameter M defines the size of an area, which is going to be sieved for smooth numbers. If we increase the size of the sieve area, we increase the number of points we need to inspect using the sieve. We can expect to gain more B-smooth numbers by increasing M. However, as we are moving with the sieve further from the origin, the probability that the corresponding numbers are B-smooth tends to zero.

To estimate this probability with respect to B and M we can take the fraction of B-smooth algebraic numbers in the area given by M. Smoothness probability depends also on the exact number field used in NFS. For the sake of simplicity we investigate a single polynomial case only, and suppose that smoothness probabilities in number fields

TABLE 8.1. Number of degree 6 irreducible polynomials over $\mathbb{Z}$ and randomly chosen finite fields with absolute value of coefficients under given bound.

Irred. over:	$\mathbb{Z}$	$\mathbb{F}_p$, $p = 101$	$\mathbb{F}_p$, 28-bit p	$\mathbb{F}_p$, 60-bit p
$\lvert a_i\rvert \leq 1$	292	64	48	76
$\lvert a_i\rvert \leq 2$	4678	828	808	809
$\lvert a_i\rvert \leq 3$	28178	5092	4700	4688
$\lvert a_i\rvert \leq 4$	101651	18206	16786	17200

defined by f_1 and f_2 are independent. In this section we work only with the selection of the polynomial f_1. To simplify the notation, we will denote this polynomial just by f, and indexes will be used only to denote different concrete selections of the polynomial.

DEFINITION 8.1.1. Let $f(x) \in \mathbb{Z}[x]$ be a monic polynomial irreducible over $\mathbb{Z}$, and $\alpha \in \mathbb{C}$ be its root, $f(\alpha) = 0$. Let $\mathcal{M} = \{a + b\alpha | 0 < |a|, b \leq M, \gcd(a, b) = 1\}$ and $\mathcal{S} = \{\xi | \xi \in \mathcal{M}, \xi \text{ is } B\text{-smooth}\}$. We call $P_s = |\mathcal{S}|/|\mathcal{M}|$ the smoothness probability w.r.t. B, M, and f.

The polynomial f clearly influences concrete norms of algebraic integers in the corresponding number field. Let $f(x) = \sum_{i=0}^{n} a_i x^i$ and denote by $A = \max\{|a_i|\}$ the maximal coefficient of f. Then norms of corresponding algebraic integers in the area given by M can be bounded by $O(nAM^n)$. Clearly, higher A means smaller smoothness probability. However, even if we choose two different polynomials f_1, f_2, with the same maximal coefficient A, the respective smoothness probability P_s is usually different.

8.1.1. Scope of the experiments. We have conducted following experiments to investigate the smoothness probability distribution w.r.t. B, M, and especially f, where f is a degree 6 irreducible polynomial with small coefficients.

We have generated a fixed set $\mathcal{F}$ of degree six monic irreducible polynomials over $\mathbb{Z}$ with small coefficients. Polynomials were enumerated over all possible coefficient choices within given bound on the absolute value of their coefficients $|a_i| \leq 4$. Irreducibility tests were performed using the algorithms implemented in library NTL.[1] Table 8.1 summarizes number of irreducible polynomials over $\mathbb{Z}$ (within given bounds) and compares it with the number of polynomials that are also irreducible over randomly chosen degree 1 finite fields $\mathbb{F}_p$. It should be noted that a polynomial, irreducible over some $\mathbb{F}_p$, is also irreducible over $\mathbb{Z}$, but in general we can expect that degree 6 polynomial irreducible over $\mathbb{Z}$ is also irreducible over given $\mathbb{F}_p$ with probability near 1/6.

For further experiments we have reduced the set $\mathcal{F}$, so that it contains just the first 1000 polynomials irreducible over $\mathbb{Z}$. We have fixed the upper smoothness bound $B_{\max} = 2^{24}$. For each polynomial f_i with root α_i we have computed all norms (using NTL) of algebraic numbers $a + b\alpha_i$, from set $\mathcal{M}$ with $M_{\max} = 64$. Using the adapted fast multi-factorization algorithm [**13**] we have found for each algebraic integer the largest factor

[1] NTL – a library for doing numbery theory – version 5.4, Release date: 2005.03.25, Author: Victor Shoup, GPL licence. The latest version of NTL is available at `http://shoup.net/ntl/`.

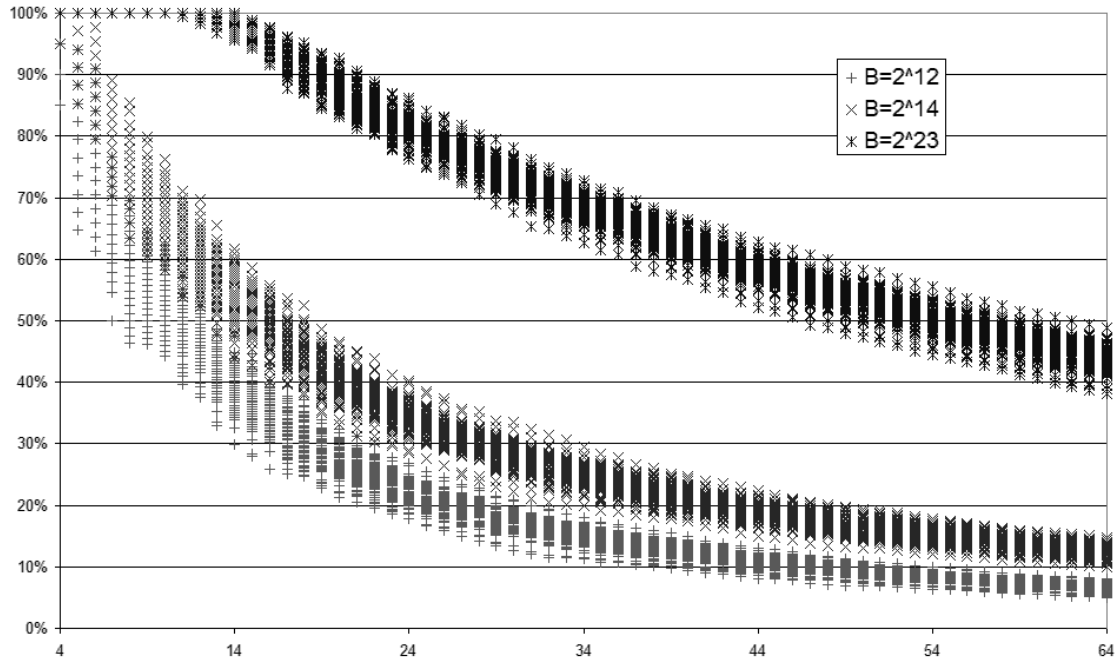

FIGURE 8.1. Smootness probability distribution with respect to M for different smoothness bounds B and first 1000 irreducible polynomials.

of the norm, if it was smaller than $B_{\max}$. Thus for each choice of $B \leq B_{\max}$, and $M \leq M_{\max}$ we are able to compute P_s as a function of B, M and f_i, where i indexes the concrete choice of polynomial $f \in \mathcal{F}$.

8.1.2. Experimental results. We present some basic findings based on our experimental results:

(1) The smoothness probability distribution w.r.t. B, M and f is shown in Figure 8.1. As expected, with increasing the sieve area size (parameter M) we are reducing the smoothness probability (over the whole area). This means that we are able to find more equations by increasing the sieve size, but with more work per equation.
(2) We can compensate the smoothness probability decrease w.r.t. M by increasing the smoothness bound B. In our experimental range we have gained an average increase of 5% in smoothness probability by doubling the smoothness bound B.
(3) For a fixed M, and B, the smoothness probability w.r.t. the polynomial choice can be approximated by a normal distribution. E.g. in our experiment P_s the distribution for $M = 32, B = 2^{12}$ w.r.t. polynomial choice is $N(\mu = 0.160, \sigma = 0.014)$ with χ-test p-value 0.203 (see Figure 8.2).
(4) From the Figure 8.1 it is also apparent that P_s varies widely when we change the polynomial. E.g. the smoothness probability for the best polynomial at given $B = 2^{12}, M = 32$, is as high as the smoothness probability for the worst polynomial at $(4B, M)$. This means that with the best choice of the polynomial, we need four times smaller factor base than in the worst case, or two times

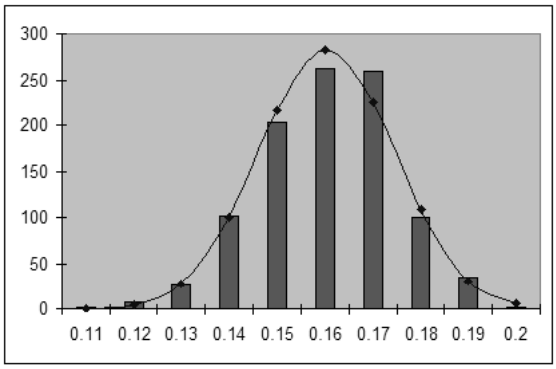

FIGURE 8.2. Histogram of smootness probability distribution with respect to polynomial choice (first 1000 irreducible polynomials) for $M = 32$ and $B = 2^{12}$.

TABLE 8.2. Correlation of smoothness probability w.r.t. polynomial selection for fixed M and different values of B.

$\log_2 B$	12	15	18	21
15	0.833	1		
18	0.782	0.888	1	
21	0.755	0.811	0.911	1
24	0.755	0.776	0.845	0.915

smaller than in the average case. Thus we can expect reduced sieving time, and linear algebra complexity.

We can formulate the following hypothesis H: If some polynomial f_i have a high/small P_s in one setting B, M, it would also have relatively high/small P_s for any chosen B, M. If this hypothesis H was true, we could pick some good polynomials based on P_s measured in a small area/with a small bound B, and then use them for a larger experiment.

Let $\overline{x}_1 \in [0,1]^{1000}$ be a vector created of different values $P_s(B_1, M_1, i)$, with B_1, M_1 fixed, and $i = 1, \ldots 1000$ indexing all $f \in \mathcal{F}$. Now let $\overline{x}_2$ be a similar vector with different B_2, M_2. Hypothesis H can be reformulated as follows: The vectors $\overline{x}_1$ and $\overline{x}_2$ are strongly correlated for any choice of B_1, M_1, B_2, M_2.

Table 8.2 denotes a situation, where M is fixed, i.e. $M_1 = M_2 = 16$. The distributions are strongly correlated, even with a significant change of B. E.g. the correlation coefficient between distributions with $B_1 = 2^{12}$, and $B_2 = 2^{24}$ is 0.755. This means that we can expect a polynomial with high P_s for one smoothness bound to also have a relatively high P_s for another smoothness bounds. For a fixed $B = 2^{12}$, and distinct $M_1 = 16, M_2 = 64$, we get $\rho = 0.703$. Thus P_s seem to be more sensitive to a change of sieve area size M. When we change both parameters, i.e. $B_1 = 2^{12}, M_1 = 16$, and $B_2 = 2^{24}, M_2 = 64$, we get $\rho = 0.567$. In this case the correlation is weaker.

The experiments show, that hypothesis H (as formulated) is too strong. We can expect that some "good" and "bad" polynomials exist (with high and low P_s respectively). We

TABLE 8.3. List of TOP-10 polynomials for $B = 2^{24}$, $M = 32$, 2D case.

Rank	Polynomial	P_s	Rank, $B = 2^{12}$	P_s, $B = 2^{12}$	Rank, 3D
1	[2 -2 1 0 -1 2 1]	0.8020	27	0.1864	829
2	[2 -2 1 1 -2 2 1]	0.8014	119	0.1762	390
3	[2 -2 1 0 2 0 1]	0.7952	16	0.1901	96
4	[2 -2 0 0 -2 -2 1]	0.7949	38	0.1850	268
5	[2 -2 -2 2 2 -1 1]	0.7937	7	0.1924	58
6	[2 -1 -2 0 1 0 1]	0.7937	15	0.1901	151
7	[1 -1 0 0 1 1 1]	0.7926	75	0.1788	104
8	[2 -2 -1 0 -1 -1 1]	0.7921	514	0.1607	484
9	[2 -2 -1 0 -2 0 1]	0.7913	332	0.1669	781
10	[2 -2 -1 0 2 -1 1]	0.7913	11	0.1908	67

have a high chance to experimentally select a "good" polynomial only if we select large enough parameters B_1, M_1. Unfortunately, the real size of the sieve region is much larger than the reasonable scope of polynomial selection phase, as it is not practical to measure exactly the number of smooth elements in a larger region for so many polynomials.

The last experiment conducted was a comparison between two- and three-dimensional sieve area (see further Section 8.2). We have fixed the parameter $M = 32$. A three dimensional equivalent of a set $\mathcal{M}$ was a set $\mathcal{M}_3 = \{a + b\alpha + c\alpha^2 | 0 < |a|, |b|, c \leq M, \gcd(a, b, c) = 1\}$. We imposed an additional condition that norms of the examined numbers must be no greater than in the two dimensional region (so we can expect similar smoothness probabilities in both experiments). This meant, that we had to compute and (partially) factor 42830 norms for each of 1000 polynomials. The results when considered separately were similar to the 2D case (approximately normal distribution, the qualitative difference between the best and the worst polynomial, correlations between sets with different B, M). The main difference is notable, if we compare the lists $P_s^{(2)}(B, M, i)$ (2D case) with $P_s^{(3)}(B, M, i)$ (3D case). The respective correlation coefficients are only 0.256 for $B = 2^{12}$ and 0.191 for B^{24}. This means we are unable to conclude, which polynomials are good for 3D sieving, even if we have a list of good polynomials suitable for 2D sieving.

In the practical NFS setup we can either select a random polynomial f_1, or use a short preliminary sieving with more polynomials (e.g. polynomials from the Table 8.3 or the Table 8.4). Another possibility is to base the polynomials selection on the properties of polynomial f_2 (from NFS notation), as we have used in our final polynomial selection method (see Section 6.3).

TABLE 8.4. List of TOP-10 polynomials for $B = 2^{24}$, $M = 32$, 3D case.

Rank	Polynomial	P_s	Rank, $B = 2^{12}$	P_s, $B = 2^{12}$	Rank, 2D
1	[2 -2 2 0 2 0 1]	0.7731	3	0.1684	629
2	[2 -1 -1 1 2 1 1]	0.7665	1	0.1717	56
3	[1 -1 1 0 1 0 1]	0.7664	25	0.1623	137
4	[1 1 1 0 1 0 1]	0.7664	26	0.1623	901
5	[1 0 1 0 1 -1 1]	0.7664	28	0.1618	902
6	[1 0 1 0 1 1 1]	0.7664	29	0.1618	903
7	[2 -2 2 2 2 -1 1]	0.7660	16	0.1640	793
8	[2 -2 1 2 2 0 1]	0.7654	8	0.1664	169
9	[2 -2 2 2 1 -1 1]	0.7649	12	0.1646	75
10	[2 -2 2 2 2 2 1]	0.7646	2	0.1698	327

TABLE 8.5. List of BOTTOM-10 polynomials for $B = 2^{24}$, $M = 32$, 2D case.

Rank	Polynomial	P_s	Rank, $B = 2^{12}$	P_s, $B = 2^{12}$	Rank, 3D
990	[-1 0 -1 -1 1 0 1]	0.6955	993	0.1206	715
991	[-1 0 -1 1 1 0 1]	0.6955	994	0.1206	716
992	[2 -2 1 -1 1 -1 1]	0.6949	870	0.1443	285
993	[2 -2 0 0 1 -1 1]	0.6929	955	0.1364	511
994	[2 -2 1 -1 2 1 1]	0.6918	992	0.1206	419
995	[1 -1 1 1 1 -1 1]	0.6889	999	0.1146	604
996	[1 1 1 -1 1 1 1]	0.6889	1000	0.1146	605
997	[2 -1 -2 1 0 0 1]	0.6880	988	0.1267	415
998	[2 -2 1 2 0 0 1]	0.6832	978	0.1329	485
999	[1 0 -1 -1 -1 0 1]	0.6770	968	0.1345	996
1000	[1 0 -1 1 -1 0 1]	0.6770	969	0.1345	997

TABLE 8.6. List of BOTTOM-10 polynomials for $B = 2^{24}$, $M = 32$, 3D case.

Rank	Polynomial	P_s	Rank, $B = 2^{12}$	P_s, $B = 2^{12}$	Rank, 2D
990	[2 -2 -2 0 -2 2 1]	0.6570	968	0.1155	89
991	[2 -1 -2 2 -2 -1 1]	0.6567	998	0.1072	563
992	[2 -2 -2 -1 -2 1 1]	0.6531	993	0.1104	767
993	[1 -1 -1 1 -1 -1 1]	0.6501	972	0.1148	940
994	[1 1 -1 -1 -1 1 1]	0.6501	973	0.1148	941
995	[2 -2 -1 -1 -2 2 1]	0.6498	997	0.1076	541
996	[1 0 -1 -1 -1 0 1]	0.6465	979	0.1128	999
997	[1 0 -1 1 -1 0 1]	0.6465	980	0.1128	1000
998	[2 -1 -1 -2 -2 2 1]	0.6456	996	0.1083	873
999	[2 -2 -2 0 0 2 1]	0.6424	1000	0.1051	766
1000	[2 -2 -1 0 -2 1 1]	0.6395	999	0.1055	539

8.2. Optimal sieve region

In our preliminary sieving experiments we have sieved a classical two-dimensional region, i.e. the algebraic numbers $a - b\alpha$. The main advantage of the classical sieving region is the fast computation of the norms, due to the fact that $|N(a-b\alpha)| = |b^n f(a/b)|$, where f is a sieve polynomial. Moreover, there are many optimized software packages already developed, and can be used freely for sieving experiments. Recommended region size should be $O(B^2)$, for the NFS complexity to hold.

Unfortunately, we were never able to find enough smooth equations in this 2D region. Increasing the sieve region size didn't help, due to the fast growth of norms and a considerable drop of the smoothness probability. Increase in B led to a larger factor base, and more required equations. We have found out experimentally, that to increase the smoothness probability (in a fixed region) by a constant factor, we have to double B. If we double the sieve region size (so we extended the sieve plane further from origin), the number of new equations in the original region plus new equations from the new part of the sieve region is less than the increase in the factor base size. The situation is different if we take instead an another sieve plane from a higher dimensional region, that is nearer to the origin. We cannot however extend the sieve region to all 6 dimensions of the number field, due to a faster growth of norms near origin in higher dimension, caused by a (single) large coefficient of the sieve polynomial f_2. We have conducted a series of sieving experiments to find the optimal sieve region size and shape (for a fixed B).

8.2.1. Smoothness probability in a square region. We have experimentally sieved two number fields $K_1 = \mathbb{Q}(\alpha_1)$ with α_1 a root of $f_1(x) = x^6 + x^5 - x^2 - x - 1$, and $K_2 = \mathbb{Q}(\alpha_2)$ with α_2 a root of $f_2(x) = f_1(x) + p$, $p = 193224091$ is a 28-bit prime. We have computed two smoothness bounds $B_1 = 2^{15} \approx L_{p^6}\left(1/3, (2/3)^{2/3}\right)$ (similar to SNFS), and $B_2 = 2^{19} \approx L_{p^6}\left(1/3, (8/9)^{1/3}\right)$ (similar to GNFS). Using the sieve we identified B_i-smooth elements of the sets $\mathcal{M}_{i,j,z} = \{x + y\alpha_j + z\alpha_j^2 || x| < B_i, 0 < y < B_i, \gcd(x,y) = 1\},$, with $z = 0, 1$, and computed the respective smoothness probabilities. Set of B_i-smooth elements of $\mathcal{M}_{i,j,z}$ is denoted by $\mathcal{S}_{i,j,z}$. The results are summarized in Table 8.7.

TABLE 8.7. Measured smoothness probability in square region $\mathcal{M}_{i,j,z} = \{x + y\alpha_j + z\alpha_j^2 || x| < B_i, 0 < y < B_i, \gcd(x,y,z) = 1\}$, where α_1 is a root of $f_1(x) = x^6 + x^5 - x^2 - x - 1$ and α_2 is a root of $f_2(x) = f_1(x) + p$, $p = 193224091$ is a 28-bit prime.

$\log_2 B$	$\log_2 \lvert N_{\max}\rvert$	z	$\lvert\mathcal{S}_{i,1,z}\rvert$	Probability	$\lvert\mathcal{S}_{i,2,z}\rvert$	Probability
15	30	0	47319	7.24E-05	229	3.50E-07
		1	97073	9.04E-05	327	3.05E-07
19	38	0	10239964	6.12E-05	64027	3.83E-07
		1	19767480	7.19E-05	86536	3.14E-07

The expected smoothness probabilities based on the asymptotic estimates were $p_1 = 0.005$, and $p_2 = 0.001$ for B_1 and B_2 respectively. A more accurate estimate is based on

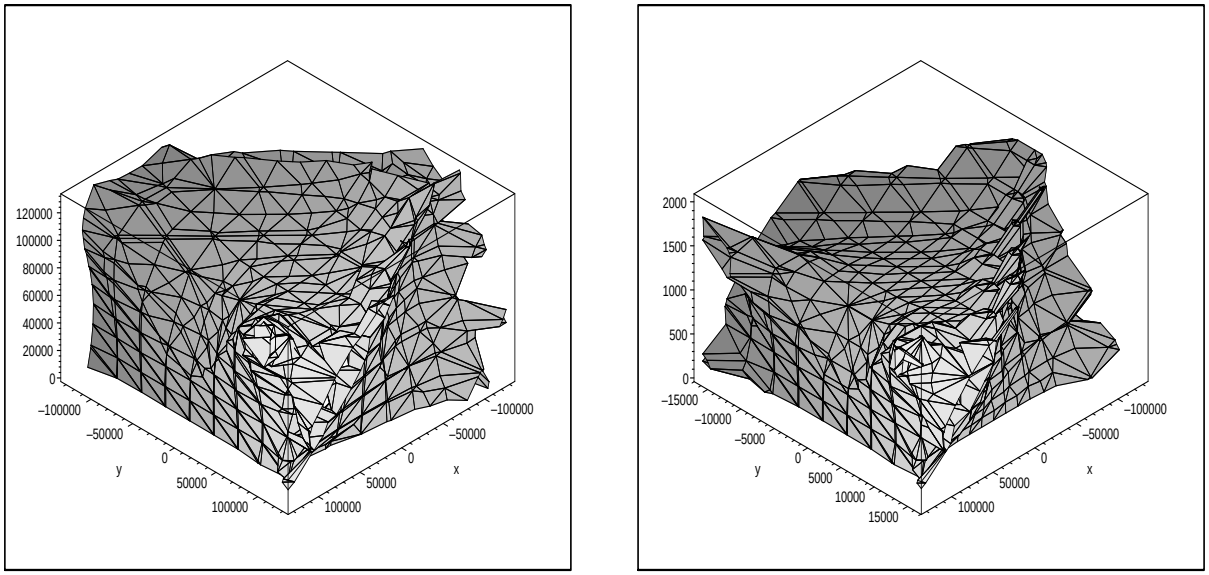

FIGURE 8.3. Solutions of the equation $|N(\alpha)| = 2^{100}$ (within a given parallelepiped), in the fields defined by polynomial $f_1(x) = x^6 - x + 1$ (on the left) and $f_2(x) = f_1(x) + p$ (on the right), where $p = 1099511627831$ is 40-bit prime. All elements within the given boundaries have (absolute) norms below 2^{100}.

$P_s = \frac{\Psi(y^u,y)}{y^u} \approx u^{-u}$ (see Section 2.2.2). Here u is the degree of the polynomial f_1, and f_2 respectively. Estimate for P_s is 2.14E-05, which is near the measured $P_{s_1} = |\mathcal{S}_1|/|\mathcal{M}_1|$. As expected, the measured P_{s_2} is lower, due to higher norms near origin. Another observation is that the number of equations obtained is much higher than in the case when smoothness probabilities are independent, e.g. $|\mathcal{S}_{2,1} \cap \mathcal{S}_{2,2}| = 1414$ instead of expected $P_{s_{2,1}}|\mathcal{S}_{2,2}| = 6$. This is caused by the fact that most of the equations are found near origin, where the smoothness probability is higher than in the whole region.

Our experiments also show that there is not enough smooth elements in $S_{i,1} \cap S_{i,2}$ to construct equations (5.9). By resizing the 2D sieve region, we increase the norms as well as the work required to collect the equations. It is much more efficient to increase the sieve region to the third dimension (compare the results from Table 8.7 for $z = 0$ and $z = 1$).

8.2.2. Shape of the sieve region. A certain care must be taken when selecting optimal sieve region. One natural (theoretical) construction of the sieve region is such that we compute the norm bound N and sieve all algebraic numbers having norm below N. As shown in Figure 8.3, such a region is very irregular, especially if a corresponding sieve polynomial has real roots. On the other hand, from an implementation point of view, rectangular (resp. cuboid) region is desired. If we use a line sieve, such as the algorithm from Section 7.4, we would like a sieve region to be skewed along some axis (to sieve longer lines, and spare line updates).

In our experiments with XTR-DL, the polynomial f_2 had a single large absolute coefficient p. This deforms a sieve region based on a fixed bound for N as shown in Figure

8.3. Norms along y axis are p-times larger than along x, and norms along z axis are p-times larger than along y. Used number field have degree 6, so the norms grow with the sixth power of x, y, or z, respectively. Thus a cuboid region that best copies the N bound has sides X, Y, Z with

$$\frac{X^6}{Y^6} = \frac{Y^6}{Z^6} = p.$$

8.2.3. A comparison of 2D and 3D sieve. Let us fix $B = L_{p^6}(1/3, (8/9)^{1/3})$. We have to sieve a region of size $V = B^2$. Let this region be of a (hyper-)cuboid shape with d dimensions ($d \leq n = 6$), and for each successive dimension $X_i^6/X_{i+1}^6 = p$. Then we have $V = X_{n-1}^d p^{d(d-1)/12}$, or $X_0 = V^{1/d} p^{(d-1)/12}$. The Highest norm in this region is expected to be $N \approx X_0^6 = B^{12/d} p^{(d-1)/2}$. With B fixed, we can write the norm bound as a function of p, and d respectively:

$$N(p, d) = p^{(d-1)/2} \cdot L_{p^6}\left(1/3, \frac{12}{d}\left(\frac{8}{9}\right)^{1/3}\right). \tag{8.1}$$

For a large p, this function is dominated by the exponential term $p^{(d-1)/2}$. Thus the asymptotically optimal degree is the lowest possible one.

Let us plot the behavior of function $N(p, d)$ denoting the expected maximal norm in the region of dimension d for a given small and medium sized p (see Figure 8.4). We can see that for small p, norm bounds in two dimensions ($d = 2$) are actually much larger than for higher d's. Optimal dimension for p's between 50 and 175-bits is $d = 3$. For smaller p's it is even higher. However, the actual difference between cases $d = 3, 4, 5$ is very small if compared to the case when $d = 2$.

The disadvantage of using higher dimensions than $d = 3$ for $p < 2^{50}$ is not readily apparent here. But the comparison based on the norm bound ignores the opposite side, norms of elements near the origin. This becomes more clear, if we compare norms of elements $\xi_1 = x + y\beta + \beta^{d-2}$, and $\xi_2 = x + y\beta + \beta^{d-1}$, where β is a root of the polynomial f_2 (having large absolute coefficient $\approx p$), and d is the dimension of the sieve. Figure 8.5 shows the comparison in a base 2 logarithmic scale for $d = 3$ and $d = 4$. The main difference is centered around the origin. For higher x, y the term x^6, y^6 dominates the norm size for both ξ_1 and ξ_2. The logarithmic norm difference near the origin is nearly $\log_2 p$, as $N(\xi_2) \approx pN(\xi_1)$. For larger d's, the impact of the term with p to the norm is stronger in a larger region, influencing negatively the number of equations sieved. The region with higher norms for $d = 3$ is roughly $2^6 \times 2^{10}$, for $d = 4$ it is roughly $2^{12} \times 2^{20}$, or 2^{16}-times larger.

The practical experiment with 3D and 4D sieve was conducted to verify our hypothesis (that $d = 3$ is better choice than $d = 4$ in our range of parameters). In this case we have used $p = 38939741891$ (a 36-bit prime). Sieve polynomials were $f_1(x) = x^6 - x^4 + x^3 + x^2 - x + 1$ and $f_2(x) = f_1(x) - p$. The sieve region for 3D sieve was[2] $[-2^{16}, 2^{16}] \times [-2^{12}, 2^{12}] \times [0, 512]$, and for 4D sieve the region was shifted by adding

[2]In practice we do not use half of plane $(x, y, 0, 0)$.

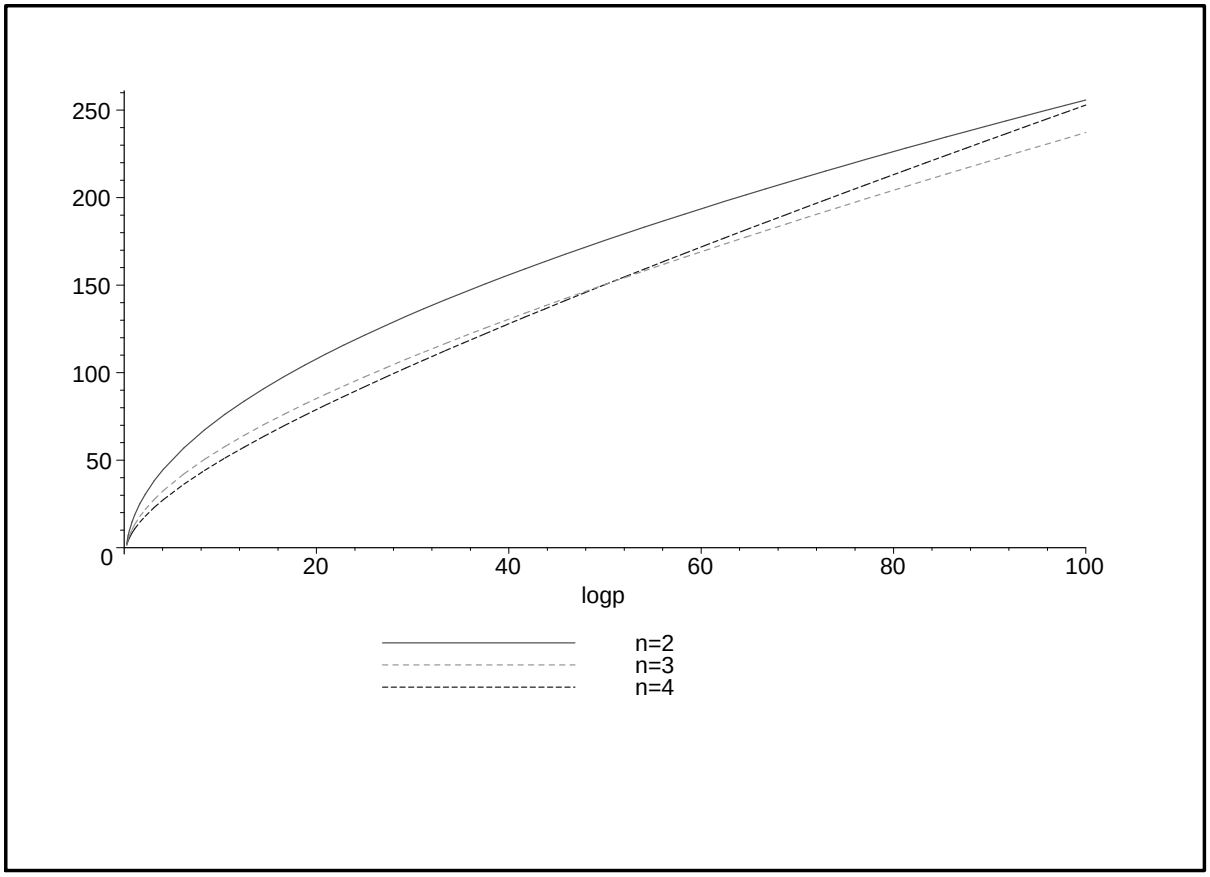

FIGURE 8.4. Base 2 logarithmic plot of function $N(p, d)$, defined by Equation (8.1), denoting the norm bound in sieve region for given p and dimension d. Dimension $d = 3$ is optimal between $p \approx 2^{50}$ and $p \approx 2^{175}$.

TABLE 8.8. Practical comparison of 2D, 3D, and 4D sieve.

Sieved space	Number of equations
$(x, y, 0, 0)$	2255
$(x, y, 1, 0)$	8106
$(x, y, 0, 1)$	108
$(x, y, 1, 1)$	86
$(x, y, z, 0)$	282828
$(x, y, z, 1)$	14368

β^4. The results are shown in the Table 8.8. The number of equations with $d = 4$ is significantly lower than for $d = 3$.

There is yet another important dimension-dependent factor that influences the number of equations we are able to find. In a classical 2D sieve, all equations with $\gcd(x, y) > 1$ are discarded, i.e. about 40 % of the sieve region is unusable[3]. Moreover, we cannot use points with $y \leq 0$. In the case when $z = 1$, we can sieve the whole plane $(x, y, 1)$,

[3]The number of equations discarded is in fact much higher. In example from Table 8.8, number of equations in full region $[x, y, 0, 0]$ was 30850. This is due to fact that with equation for point (x_1, y_1)

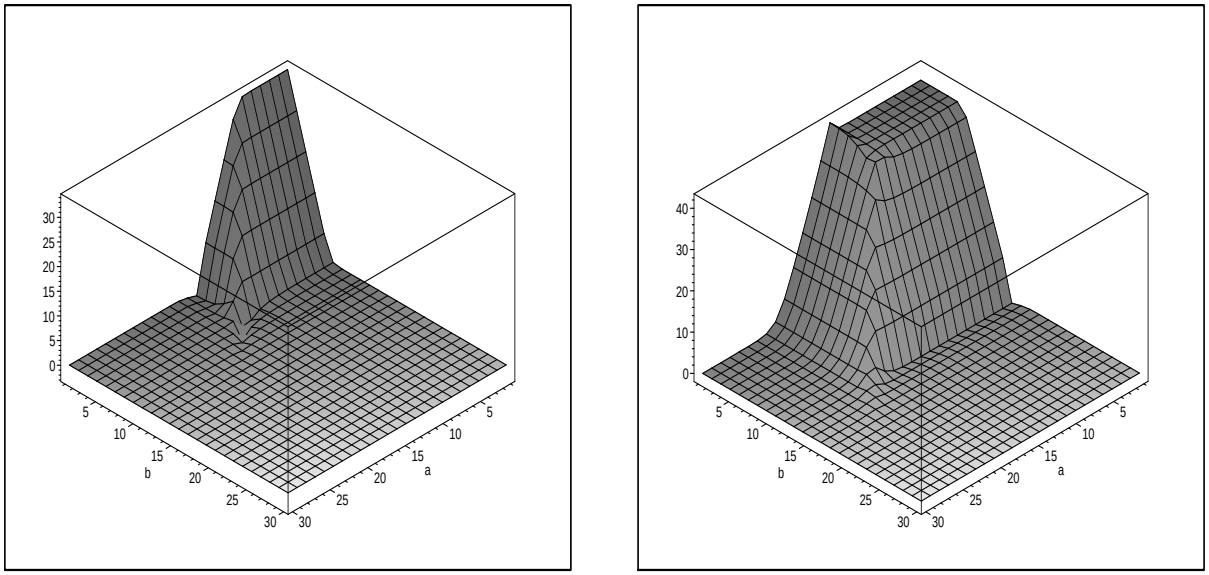

FIGURE 8.5. Plot of the function $\log_2 |N(2^X + 2^Y\beta + \beta^{d-1})| - \log_2 |N(2^X + 2^Y\beta + \beta^{d-2})|$ for $d = 3$ (on the left) and $d = 4$ (on the right). β is a root of polynomial $f_1(x) = x^6 - x + 1 + p$, where $p = 1099511627831$ is 40-bit prime.

and in the case of general z we can still sieve all points with $\gcd(x, y)$ coprime to z. This leads to a significant increase in the number of smooth equations, especially when using polynomial f_1 with small coefficients (see Figure 8.6). This increase might also be applicable in case of NFS to compute discrete logarithms in $\mathbb{F}_p$ using polynomial selection from [**52**].

8.2.4. A practical selection of the sieve region size. The sieve region selection was further refined after executing initial sieving experiments (see Section 8.4.1). In the first DLP experiment, parameters were chosen ad-hoc, just using the assumption that x-axis must be longer than B. Moreover, its size was larger than expected optimal size B^2. After the sieving, we could see, that there was enough smooth equations contained in a substantially smaller region.

However, using the data obtained from sieving the too large region, we have found out, that it is possible to quite accurately estimate the optimal region size by sieving only a small region on a single plane with $z = 1$. When increasing z, we expect to find almost the same constant number of equations in every subspace $[-x_{\max}, x_{\max}] \times [-y_{\max}, y_{\max}] \times [z_0, 2z_0]$ (i.e. we must double our effort to gain only a constant increase in the number of equations). Similar observation is valid for increase in x- and y-direction respectively. Due to the construction of our sieving algorithm (sieving along x axis, saving some sieve updates in memory, see Figure 7.1) we expect that the sieving time is doubled, when we double the sieve region along axes y and z respectively. On the other hand, when we double the sieve region in x-direction, we expect the new time to be less than doubled.

we also get equations in all points (kx_1, ky_1), $kx_1 < X, ky_1 < Y$, and most of the smooth equations are found near origin.

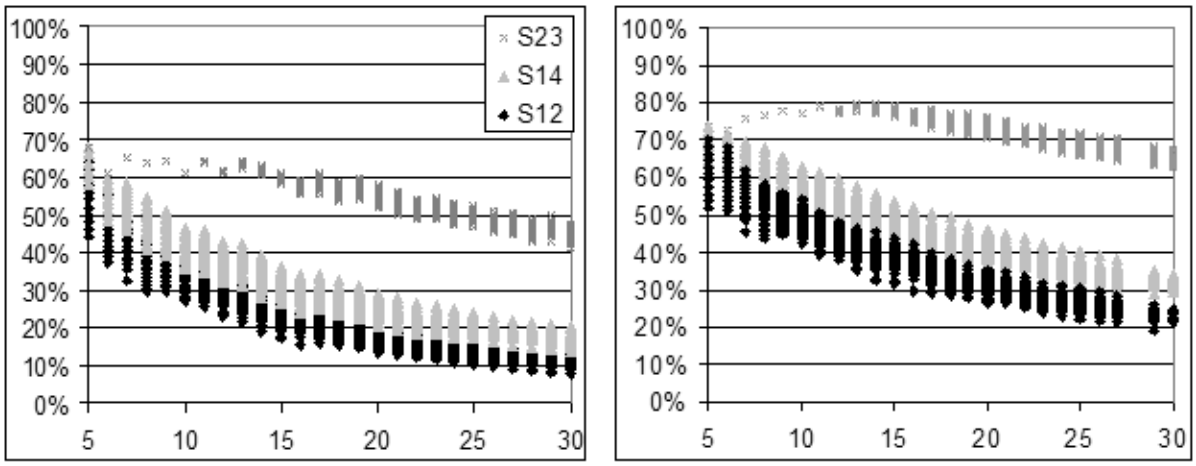

FIGURE 8.6. Comparison of smoothness densities in a region $[-M, M] \times [1, M]$ (on the left), $[-M, M] \times [-M, M] \times [1, M]$ (on the right) for various degree 6 fields (defined by roots of polynomials with small coefficients). Series S12, S14 and S23 denote 2^{12}, 2^{14} and 2^{23}-smoothness.

TABLE 8.9. Time t to sieve a single rectangle $[-2^X, 2^X] \times [2^Y, 2^Y + 2^{10}] \times [1, 1]$, and number of equations found in this rectangle (different columns for different choice of Y).

X	t	$Y = 0$	$Y = 10$	$Y = 11$	$Y = 12$	$Y = 13$	$Y = 14$
10	18	1076	48	16	4	2	1
11	18	1414	102	39	9	3	1
12	19	1703	158	74	16	8	2
13	21	1913	209	95	30	10	3
14	24	2084	257	122	42	14	3
15	31	2195	308	147	62	18	6
16	44	2245	344	174	75	19	7

The difference in the sieving time growth along axes x, y, z should be taken into account when selecting the optimal sieve region size. However, the time growth along x axis is very implementation-specific, and depends also on the actual factor base used. It is thus easier to estimate the optimal region size experimentally, by preliminary sieving specific smaller regions. However, we must take into account not only the sieving time, but the number of potential equations as well.

The method is demonstrated for the case p_{32} (see Section 8.4.6 for exact parameters). We have sieved a single rectangle $[-2^X, 2^X] \times [2^Y, 2^Y + 2^{10}] \times [1, 1]$, for various X, Y. The results are summarized in Table 8.9. As expected, the time to sieve the region does not double with increasing X by 1, but is instead growing slowly (it is doubled only after increasing the region size 64-times). Again, time to sieve the rectangle $[2^X, 2^X + c] \times [-2^Y, 2^Y] \times [1, 1]$ doubles with every Y increase, as the processing of every line for various Y's is essentially the same.

The number of equations gained by increasing X or Y is relatively small. To find the optimal region size, we have computed an estimate of the average number of equations per second for each sieving region $[-2^X, 2^X] \times [-2^Y, 2^Y] \times [1, 1]$ (from the data in Table 8.9). The results are shown in Figure 8.7. By increasing Y the average number of equations per second quickly decreases (doubled time, less new equations). On the other hand, by increasing X, the average number of equations per second grows, until optimal X-size is reached, and only then it decreases. Optimal X grows slowly when increasing Y (the decrease in the smoothness density along x-axis is steeper for lower Y). Final region shape is selected by combining the criteria for region size based on norms (see Section 8.2) and criteria based on actual sieve timings from Figure 8.7. In our further experiments, the estimated region size was sometimes too small (producing not enough equations). This could however be easily remedied by sieving the additional neighboring regions.

8.3. Sieve tolerance

We have conducted sieving experiments to check the influence of the tolerance value T (see Section 7.2) on the real world behavior of our implementation of the NFS. The sieve polynomial was $F(x) = f_1(x)f_2(x)$ (both sides sieved together), where $f_1(x) = x^6 - 2x + 2$ and $f_2(x) = f_1(x) - p$, $p = 529043$ is a 20-bit prime. Sieve region was $[-2^{14}, 2^{14}] \times [0, 2^{12}] \times [1, 2^{10}]$. Smoothness bound was $B = 478741$ (19 bits, factor base in the initial experiments was chosen to have exactly 80000 prime ideals). All sieve updates and logarithmic estimates were measured in the "number of bits", which is $nb(x) = \lfloor \log_2 x \rfloor + 1$. Logarithmic estimate of the norm was based on the number of bits of coordinates, with a more careful handling near origin. We have verified experimentally that the error of the estimate was at most 12 bits. Ideals with norms below 128 were not used in the sieve. Their omission was compensated by a constant based on theoretical values discussed in Section 7.2.

To estimate the optimal sieve tolerance setting, we sieved the region with different tolerance setting, $T = 28{+}6k$ bits[4] for $k \in \{-10, -9, \ldots, 10\}$. For every k, we have measured the total sieve time, the number of reported smooth equations, and the number of really smooth equations.

In the first experiment, we measured the number of reported equations $N = N(T)$ (not necessary smooth) and the total time $t = t(T)$ needed (on average) to produce smooth equations from sieving a single plane (with fixed z) with a given tolerance T. The number of equations reported grows exponentially with T. This is due to the fact that by increasing a tolerance by a single bit, we effectively double the interval of acceptable numbers. Up to the tolerance $T = 28$, the growth of N had almost a negligible effect on the total time, as the dominant time was the line sieving time $t_0 \approx 36s$. Later the growth of N had more significant impact, and for the largest tested tolerance $T = 88$ we had $t = 190s$. After subtracting the constant term t_0 we were able to estimate additional time required to process a single reported candidate, $t_c = 67.6 \cdot 10^{-9}s$. Thus

[4]Constant 28 comes from 12 bits to compensate norm estimate errors and 16 bits to compensate small factors, being 95-percentile of function S_7 from Section 7.2.

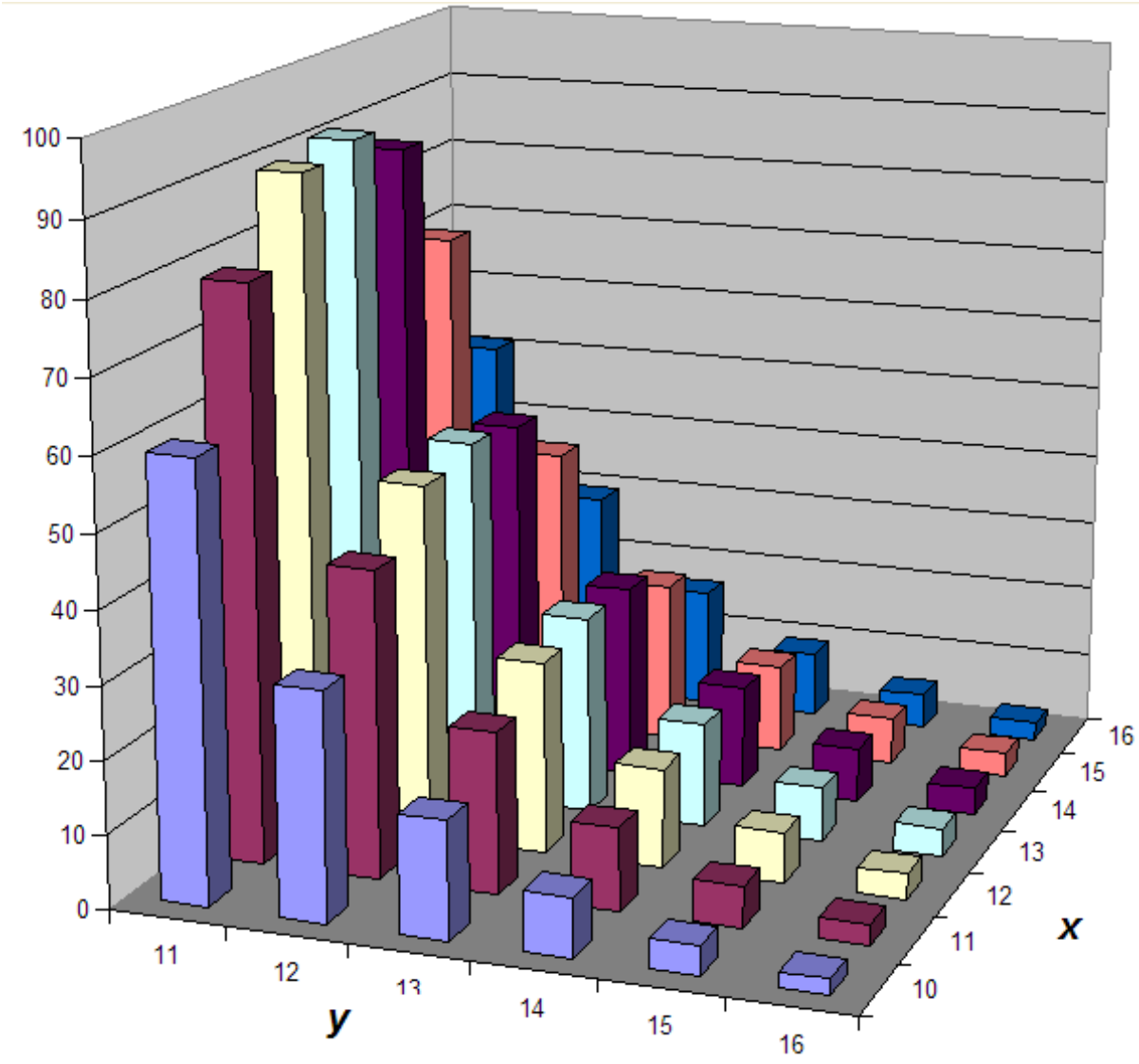

FIGURE 8.7. Experimental setup for p_{32}. Number of equations found per second in a region $[-2^X, 2^X] \times [-2^Y, 2^Y] \times [1, 1]$.

the expected sieve time with the constant tolerance can be estimated by functions

$$t(T) = t_0 + t_c N(T), \quad N(T) = \alpha e^{\beta T} \tag{8.2}$$

where α and β are some constants dependent on the number field and the norm estimation (in our experiments $\alpha = 1.3 \cdot 10^5, \beta = .1219$). The values of t_0 and t_c are very implementation specific (both hardware and software). Parameter α depends mainly on the smoothness density, and can be found by a single sieving (with $T = 0$). We expect, that the parameter β depends only on the degree of number field n, $\beta = \frac{\ln 2}{n}$. The reasoning is as follows: to double the number of reported points, we can allow to double some coefficient (increase it by 1 bit); these points have 2^n-times higher norms (increase by n bits). If this hypothesis is correct, we can determine the whole function $N(T)$ by a single measurement of α.

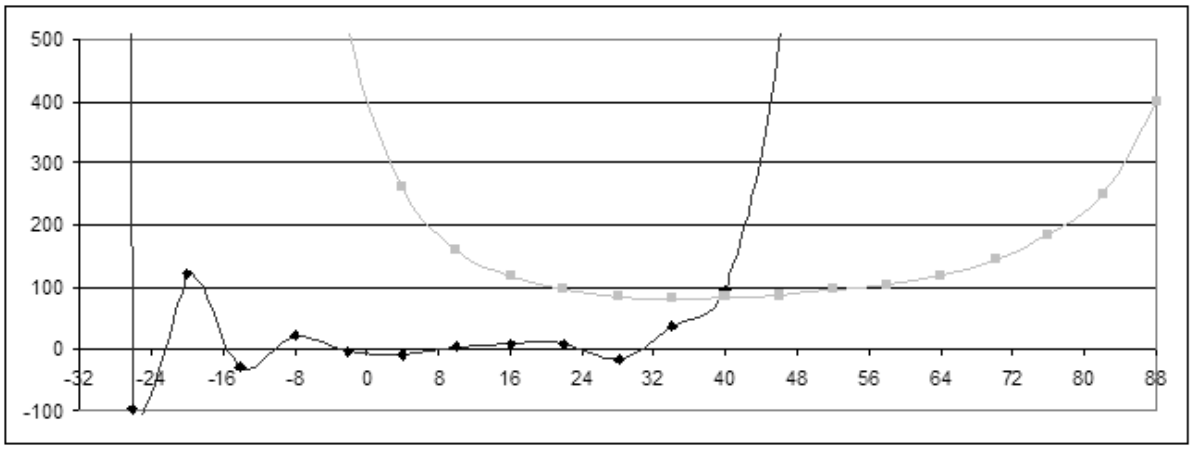

FIGURE 8.8. Average time per smooth equation (light-grey) and marginal average time needed to find one more equation (dark-grey). Times in μs.

Another important function when evaluating the tolerance setting is the number of smooth equations that is not reported. If we set the tolerance too low, we can miss quite a large fraction of smooth equations, if we set the tolerance too high, the time required grows much quicker than the number of additional equations.

The two sources of errors (for smooth equations) are the rounding/norm estimate error, and the influence of small primes. Both of them can be approximated by a normal distribution, as well as total sieve error. This is confirmed by our experiments, where the error term had distribution $E \sim N(\mu, \sigma^2)$, with $\mu = 14.1$, and $\sigma = 13.17$. If we want to collect 95 % of all smooth equations, we must set tolerance to $T = 40$ bits. From previous experiments, average time for $t(40) = 38s$. The number of candidates reported by the sieve was $N(40) = 25216694$, out of which 466238 was really smooth (only 1.8 % of reported equations). Using the error term distribution E we can now define function $s(T)$ returning the number of expected smooth equations for a given tolerance setting T.

To locate the optimal value of T we computed functions $\frac{t(T)}{s(T)}$ — average time required to find a smooth equation, and $\frac{\Delta t(T)}{\Delta s(T)}$ — average time required to find one additional equation (by increasing tolerance). Experimental results are plotted in Figure 8.8. Although the average time per equation is low in the interval $[16, 64]$, the marginal time grows quickly after $T = 40$. That is, we can find more[5] smooth equations in the same interval by increasing the tolerance above $T = 40$, but the additional time required can be better used e.g. in increasing the sieve region size.

We emphasize that it is not practical to conduct the above measurements for every sieving experiment, just to determine the optimal tolerance value. To estimate optimal tolerance we can measure (or estimate) the sieve time t_0, the additional processing time per candidate t_c, parameters of the distribution E, and the function $N(T)$. Then we

[5]As shown above, there are approximately 5 % of all smooth equations with error term above 40 bits.

can choose optimal T as the value, where function $\frac{\Delta t(T)}{\Delta s(T)}$ (average time required to find an additional equation) gets above some arbitrary limit.

8.4. Sieving experiments

The goal of our research was to solve the XTR-DL problem for a given choice of p. Most of our experiments (all except the largest one) were done on a single computer with the following parameters:

```
AMD Athlon(tm) 64 Processor 3200+
cpu MHz     : 2202.901
cache size  : 512 KB
MemTotal    : 1 GB
MIPS        : 4410.67
```

For the last experiment we have created a small network of 8 computers working in parallel. Their parameters[6] were:

```
AMD Sempron(tm) Processor 3400+
cpu MHz     : 2010.302
cache size  : 256 KB
MemTotal    : 2 GB
MIPS        : 4023.09
```

The source code of the siever, and some of the helper programs and data are available on a dedicated web-page `http://147.175.106.2/kaivt/Vyskum/XTRDL` (at the date of the publication).

8.4.1. Preliminary experiments. Our preliminary experiments were published in [**114**] The XTR system was defined in a field with characteristic $p = 529043$ (a 20-bit prime). XTR group size was $q = 93295322269$ (a 37-bit prime). The XTR-DL was easily solvable by Pollard's rho-algorithm [**87**]. On the reference computer this computation took 51 seconds. The complexity of Pollard's rho is given by $O(\sqrt{q})$, thus the expected running time is doubled with every 2 additional bits in the size of q.

The polynomial selection was based on results of [**116**]. In this initial experiment we have chosen the polynomial $f_1(x) = x^6 - 2x + 2$, which was the first applicable polynomial with the highest smoothness density found in experiments described in Section 8.1. The second polynomial was chosen to be $f_2(x) = f_1(x) - p$. Unfortunately, as we found later the index of $\mathbb{Z}[\alpha_2]$ in $\mathcal{O}_{K_2}$ is 7, and some extra work was required to handle the ideals over prime 7 in this field. Further observations led to simplification of the polynomial selection, as well as to exclusion of polynomials with $[\mathbb{Z}[\alpha] : \mathcal{O}_K] \neq 1$.

The factor base size was larger than optimal. With bound $B = 80000$, it contained 15620 prime ideals of degree 1 with norm above 128. Only these ideals were used to

[6] All computers were the same, except one, which had halved RAM clock. This has caused that our algorithm required 45 % more time to run on this computer. This shows that the algorithm implementation is far from cache-optimal.

sieve region $[-2^{16}, 2^{16}] \times [-2^{12}, 2^{12}] \times [1, 256]$, i.e. 2^{38} points altogether. The total sieve time was 24691 seconds (approx. 7 hours). We have found 29477 full equations, i.e. smoothness density was 10^{-7}. After throwing away some of the excess equations, we have constructed a 19048×13959 matrix, which was solved using Lanczos algorithm in 4431 seconds (18 % of the sieve time). Afterwards we were able to compute XTR-discrete logarithms by lifting traces back to $\mathbb{F}_{p^6}$ and using descent method described in Section 5.2.4.

After these first successful experiments, we have focused on improving the sieving phase, and testing the algorithms for different fields with increasing characteristic $p_{24} = 16102169$, $p_{28} = 193224089$, $p_{32} = 3147768119$, $p_{36} = 38939741891$, and $p_{40} = 1081034284409$ (subscript denotes number of bits of given prime, see Section 8.4.6). We have tried to further optimize parameter selection (especially the sieve region size) based on the initial experimental results. The largest experiment we concluded was to solve the DLP in $\mathbb{F}_{p^6}$ with 40-bit prime p, i.e. the field size 240 bits.

8.4.2. Record solution. The largest solution we were able to compute by our NFS programs was a computation of discrete logarithms in $\mathbb{F}_{p^6}$ with 40-bits prime $p_{40} = 1081034284409$, and its respective XTR group with order $q = 389545041355532555398291$ (79 bits). Computed value of smoothness bound was $B = L_{p^6}\left(1/3, (8/9)^{1/3}\right) \doteq 6532326$. We have chosen the following pair of sieve polynomials: $f_1(x) = x^6 - 2x^5 + x^3 - x + 2$, and $g(x) = f(x) + p$.

We have constructed a factor base consisting of all degree one ideals with norm greater than 128 and less than B in both $\mathbb{Z}[\alpha_1]$, and $\mathbb{Z}[\alpha_2]$ respectively. The total number of ideals in the sieve factor base was 893707.

The estimated region size based on calculations from Section 8.2.3 (to sieve B^2 points with the smallest norms possible) is roughly $3.5 \cdot 10^6 \times 3.5 \cdot 10^3 \times 345$. In our sieving program we are using bounds that are powers of two, with x, y axes centered around the origin, i.e. the original sieve region is $[-2^{21}, 2^{21}] \times [-2^{14}, 2^{14}] \times [1, 345]$. To sieve this region would mean to examine approximately 2^{45} points. Our preliminary sieving showed that increasing x-axis bound above 2^{18} produces only a very limited number of equations. We have similarly reduced the sieve region along the y axis to one quarter of the original. Thus we have used 32-times smaller sieve region with the expected number of sieve points nearly 2^{40}. This size is comparable with the expected running time of Pollard's rho algorithm (2^{40} field exponentiations), however the sieving is much faster in practice.

The sieve region was divided to smaller blocks along x-axis, with 128 points each. Thus every ideal has at most one intersection with each sieved segment. Tolerance bound was set to 40, based on experiments from section 8.3. As the estimated sieving time was too large to run on a single computer, we have created a parallelization engine. This was distributing the sieving tasks among 8 computers. Each computer was sieving a single plane with a fixed z. After finishing the job, smooth equations were sent to the server, which assigned a new z plane to the siever.

The shortening of the sieve region along x-axis led to the smaller number of equations than required, so we let the sieving run longer to $z = 1149$. Server was stopped manually

after the number of equations per plane was too small to justify continued sieving. As we have found only 606040 equations in the sieve region $[-2^{18}, 2^{18}] \times [-2^{12}, 2^{12}] \times [1, 1149]$, we have restarted the sieve in neighboring regions along y axis. Thus we also sieved regions $[-2^{18}, 2^{18}] \times [-2^{13} - 2^{12}, -2^{12}] \times [1, 452]$, and $[-2^{18}, 2^{18}] \times [2^{12}, 2^{12} + 2^{13}] \times [1, 265]$, getting 259609 and 212335 equations respectively. The total number of equations was thus 1077984, which gave us also some spare equations for SGE. We have sieved a total of $2^{19} \times 2^{13} \times 1866$ points, which is approximately 1/6 of the originally estimated region size. The total sieve time was 2087070 s, i.e. slightly more than 3 days on 8 computers working in parallel (+distribution server, but the communication overhead is small compared to the sieving effort).

Sieving program produced the list of points along with valuations for ideals in the factor base. We have additionally computed 12 character maps for each point as well as valuations corresponding to ideals over small primes ($p_i < 128$). Finally, we have constructed a linear system with 1077984 equations and $854821 + 12$ unknowns. The number of unknowns is smaller than the original sieve factor base size, because some prime ideals were not used in any smooth equation.

8.4.3. Sieving with large prime method. Because the equation size was too large for our solver to tackle, and we have seen that some of the larger ideals were unused, we have tried to reduce the system size by using smaller factor base and the method of large primes.

We have reduced the smoothness bound to $B = 2^{20}$, leading to a sieve factor base with 163485 unknowns. We have increased the sieve tolerance to 60 bits. Instead of automatically discarding misreported equations (that were in fact not smooth), we have stored the partial equations with large factors bellow $B^{1.2} = 2^{24}$. We have sieved a slightly larger sieve region $[-2^{20}, 2^{20}] \times [-2^{15}, 2^{15}] \times [1, 144]$ in total time 3636478 s (approx. 42 days). We have only found 40750 smooth equations. We estimate that approximately 14000 new equations can be found by doubling of the sieve region size (along z axis). If the trend holds infinitely (i.e. a constant number of new equations per each doubling of the sieve region size is achieved), than we can expect to collect enough smooth equations in more than 100 years.

The final number of partial equations was nearly 10-times higher than that of the smooth equations (exactly 382371:40750). However, the number of new unknowns was quite large: 731611; that is more than 1.9 new unknowns per equation (we allowed at most 2 large primes, so the maximum number of new unknowns is 2 per equation). We eliminated all equations that contained at least one large prime occurring only once. 87857 partial equations remained with 96606 new unknowns (nearly 1.1 new unknowns/equation). This filtering was repeated 10-times, and the result was 10126 partial equations with 9660 new unknowns (.95 new unknowns/equation). The usefulness of partial equations in itself was thus very small (equivalent of 466 smooth equations, i.e. 1.1%). One of the solutions would be to perform additional special-q sieve for each large prime found in partial equations. However, we have not pursued this way further, as we already had the full system of comparable size obtained with much less computational effort.

8.4.4. Solution of the linear system for p_{40}. Linear system obtained from p_{40} sieving experiments had $r = 1077984$ equations, $c = 854821$ unknowns corresponding to virtual logarithms of prime ideals and $2n = 12$ unknowns corresponding to character maps. It is very sparse, with only 17487557 non-zero elements (which are small numbers, mostly ± 1), i.e. 16 non-zero elements per row, not counting the character maps. Densities in columns follow the expected statistics, i.e. the number of non-zero elements in columns is near r/p_j^d, where p_j^d is the norm of an ideal $\mathfrak{p}_j$ (corresponding to the j-th column). This means there is a large number of sparse columns (with few non-zero elements), a small number of dense columns (with at most $r/2$ non-zero elements), and 12 columns with character maps (fully dense, containing large 80-bit numbers).

Let A denote the matrix of our linear system, let a_j denote the j-th column of A, and A_i the i-th row of A. Let us sort[7] the columns of the system by the number of its non-zero elements, such that $w_H(a_k) \geq w_H(a_j)$ for each $k > j$. Let us sort the rows of the matrix according to first non-zero element, such that if $a_{ij} = 0$, then also $a_{kj} = 0$ for each $k > i$.

If some column contains only a single non-zero element a_{ij}, we can compute the corresponding virtual logarithm x_j after we compute all virtual logarithms in the row A_i. If a_{ij} is the first non-zero element, we just need to compute x_k for $k > j$. In the sorted matrix A the corresponding rows create an upper triangular matrix. We can store these equations for post-processing (and remove corresponding row and column from A). If a_{ij} is not the first non-zero element (in the sorted matrix), we cannot compute x_j at all, and we can remove the whole equation from the system.

Let us consider the situation, that some column a_j contains only w non-zero elements, and they are all on the first positions in the corresponding rows. We can again compute the corresponding virtual logarithm x_j after we compute all virtual logarithms x_k with $k > j$. Furthermore, the knowledge of x_j is not required to compute any x_k with $k < j$ (all non-zero occurrences in the column are used on first places). We can thus store one of the rows (A_k) on disk for post-processing. We can then eliminate a_{lj} from remaining $w - 1$ rows by computing new row $a_{lj}A_k - a_{kj}A_l$ (to avoid modular inversion). These new rows are more dense, and its first non-zero element has higher index than j, so they must be put to their new corresponding positions in the matrix.

If some column a_j contains w non-zero elements, but only $v < w$ are on the first positions, we cannot remove these rows, as the still existing previous equations depend on knowing x_j. This situation however cannot happen, if we continually insert the reduced rows to proper positions. We can stop at some time and finish the solution using sparse matrix methods (in our case we have Lanczos equation solver).

Unfortunately, this method is just like the normal Gaussian elimination, but using repositioning of rows to find pivots. The density of the matrix quickly increases, as we are using the same rows for reduction repetitively (imagine matrix with only non-zero elements per row a_{ij} and $a_{i(j+1)}$). We have changed this preliminary method to the following:

[7]The sorting of A is fast, and it is not required to store the whole matrix in the computer memory.

(1) Set $A^{(i)} = A$, and let B be an "empty matrix" (with no rows, and the same number of columns as A).
(2) Mark all columns of A with w non-zero elements, all of them on the first position, and their corresponding rows (forming groups of w_k rows).
(3) Store all unmarked rows in matrix $A^{(i+1)}$.
(4) For each group of w_k rows: Store one of the rows to matrix B. Reduce remaining rows and store them to matrix $A^{(i+1)}$.
(5) [Optional] Remove heaviest rows of matrix $A^{(i+1)}$ (if the number of rows is significantly larger than remaining unknowns).
(6) If $A^{(i+1)}$ is still too large, sort matrix $A^{(i+1)}$, and repeat from the step 2 with $A^{(i)} \leftarrow A^{(i+1)}$.
(7) Solve system corresponding to $A^{(i+1)}$ by Lanczos algorithm.
(8) Solve equations from matrix B by backtracking in opposite order of adding equations. Some virtual logarithms cannot be recovered immediately, if the corresponding equations were eliminated in Step 5.

Using this method we have reduced the original system to a system of 226059 equations in 223474 unknowns. We have done 30 iterations of the above algorithm, with 2 reductions in step 5. One iteration took at start about 1 hour, and at the end about 20 minutes. Total time was 15 hours, but the running time can be misleading, as we were using only a simple PERL script. Using optimized version of our Lanczos solver we have finally solved this linear system in 1078641 seconds (12.5 day on a single computer). Backtracking took 8 hours, again with using different PERL script, but on a slower computer. We have finally found 845377 virtual logarithms (out of original 854821), and using these values we have been able to compute discrete logarithms of 1072363 elements of $\mathbb{F}_{p^6}$ (corresponding to smooth equations).

8.4.5. Individual logarithms for p_{40}. As a first task, we have computed discrete logarithms of the elements of $\mathbb{F}_{p^6}$ corresponding to smooth elements found by the sieve (to verify the correctness of the found virtual logarithms). We show the exact values for two such elements.

The field $\mathbb{F}_{p^6}$, with $p = p_{40} = 1081034284409$ was represented as $\mathbb{F}_p[x]/(f_1(x))$, with $f_1(x) = x^6 - 2x^5 + x^3 - x + 2$ (the sieve polynomial f_1). During the sieve we have found (among others) these smooth elements:

$$\begin{aligned}
\tau_1 &= 1118 - 4096\alpha + \alpha^2 & N(\tau_1) &= 2 \cdot 5 \cdot 31 \cdot 191 \cdot 347 \cdot 367 \cdot 769 \cdot 115883 \cdot 12251 \\
\tau_2 &= 1118 - 4096\beta + \beta^2 & N(\tau_2) &= 7 \cdot 17^2 \cdot 19 \cdot 1367 \cdot 589753 \cdot 3835763 \cdot 79411 \cdot 23021 \cdot 23473 \\
\zeta_1 &= 9547 - 4096\alpha + \alpha^2 & N(\zeta_1) &= 5^2 \cdot 101 \cdot 113^2 \cdot 20123 \cdot 229519 \cdot 1113997 \\
\zeta_2 &= 9547 - 4096\beta + \beta^2 & N(\zeta_2) &= 2^2 \cdot 29 \cdot 1097 \cdot 220919 \cdot 1438061 \cdot 1217423 \cdot 647489 \cdot 159571
\end{aligned} \tag{8.3}$$

Ideals corresponding to $5^2, 113^2$ were degree 2 ideals with valuation 1, ideals corresponding to $2^2, 17^2$ were degree 1 ideals with valuation 2. Character maps for these elements are summarized in Table 8.10. Virtual logarithms of character maps and selected virtual logarithms of ideals are summarized in Table 8.11, and Table 8.12 respectively.

TABLE 8.10. Character maps corresponding to smooth elements from equation (8.3).

$\lambda(\tau_1)$	$\lambda(\tau_2)$	$\lambda(\zeta_1)$	$\lambda(\zeta_2)$
332178357106423719517553	359154011446162056546912	169698805811873719103016	319288906937222296059014
234513379622851892776968	228456514497427458682278	156406108626642883750951	158707679012260335765672
303626561019476149554780	992071350673181021243 76	272870316408916122480733	4913849619169864276 1415
179194649149685766645386	172966748775552307115918	302738685729232602437854	583803755239235591615 19
138469171204904836823850	147925965590101529317944	355354891870071499369826	268356987667066844550474
249814026430200418398740	204760358080571620556694	1107583755142185704874	98390629948965261085568

TABLE 8.11. Virtual logarithms of character maps.

Λ_1	Λ_2
337847685506752680303506	278826722621497997483834
18534809193199884951 7424	1764500661451119403563
20022834392927148191528	19997012875429846048 7035
20606211993666021143938	80556290207775092778911
28136805441831750040 2223	63073511606345029008824
20588089472055433426 3651	28294890928196807671 5558

TABLE 8.12. Virtual logarithms of selected ideals.

$\mathfrak{p}_1$	x_1	$\mathfrak{p}_2$	x_2
$(2, \alpha)$	319333234119305752826732	$(2, \beta+1)$	216690832578263446432974
$(5, \alpha+1)$	170939175959701328091975	$(7, \beta+1)$	368080756269487538573716
$(5, \alpha^2+4\alpha+2)$	385996846441714183236553	$(17, \beta+9)$	111179551687758777408205

Using values of virtual logarithms, we computed discrete logarithms of elements $t = \phi_1(\tau_1) = \phi_2(\tau_2)$ and $z = \phi_1(\zeta_1) = \phi_2(\zeta_2)$ modulo $q = 389545041355532555398291$ (order of the XTR group):

$$\begin{aligned} \log_g(1118 - 4096x + x^2) &= 74993585068277971960398 \mod q \\ \log_g(9547 - 4096x + x^2) &= 31035200967597591007797 \mod q \end{aligned} \tag{8.4}$$

To verify the solution we have computed $g' = g^e$, $e = (p^6 - 1)/q$, a generator of a subgroup with order q. We can compute it from both t, and z:

$$g' = (t^e)^{(log_g t)^{-1}} \mod q = (z^e)^{(log_g z)^{-1}} \mod q \tag{8.5}$$

$$g' = 284166533795 + 1000474640296x + 697619380851x^2 + 782310152545x^3 + 531467318536x^4 + 831332532574x^5 \tag{8.6}$$

Let us find a discrete logarithm (modulo q) of a random element of $\mathbb{F}_{p^6}$. To simulate "verifiably random" number, we have taken the first 72 decimal digits of the number π, and created the element:

$$q = 314159265358 + 979323846264x + 338327950288x^2 + 419716939937x^3 + 510582097494x^4 + 459230781641x^5 \tag{8.7}$$

For $r = 0, 1, \ldots, 2^{10} - 1$, we computed $q(g')^r = a/b$. From this we can compute $\log_g q = \log_g a - \log_g b - re \mod q$. Fractions a/b were determined using LLL for each r. We

have partially factored norms of corresponding elements in $\mathbb{Z}[\alpha]$ with the Pollard's rho factoring algorithm limited to 2^{20} steps. Only pairs a/b with both B_1-smooth norms were stored for postprocessing, where $B_1 = L_{p^6}\left(2/3, (1/3)^{1/3}\right) \approx 4.85 \cdot 10^{17} \approx 2^{59}$.

Average time required for one LLL reduction and 6 corresponding factorizations was 12.2s, leading to a total time of approx. 3.5 hours. This time can be reduced, if we stop after the first suitable r (in our case it was immediately $r = 0$). However, the norms of a, b for the first possible pair would usually have higher factors, than if we check more pairs, and take the best solution. The pair of a, b for $r = 0$ had a total of 5 factors between B and 2^{55}. From the 434 found B_1-smooth pairs (7 % of the examined pairs) we have chosen a pair of a, b arising from $r = 470$:

$$g^{470e}q = \frac{441088 - 85469x + 336329x^2 - 125053x^3 + 239252x^4 - 25547x^5}{-33155 - 459090x - 661904x^2 - 483315x^3 - 47127x^4 - 440144x^5} \tag{8.8}$$

We had to compute 4 unknown virtual logarithms, lying over primes with 28, 34, 34, and 36 bits respectively. To do this, we must find a semi-smooth equation with a given large factor. We can sieve points in the given ideal $\mathfrak{p}$, which form a lattice with determinant p. We only need to find a single semi-smooth equation, instead of $O(B)$ smooth equations required for the whole sieve. This would mean, that the sieve region should have only $O(B)$ instead of $O(B^2)$ points. However, norms on the sieved lattice grow faster, thus the expected smoothness probabilities are lower.

For the individual logarithm sieving we have fixed the sieve region to be $[-2^{12}, 2^{12}] \times [-2^8, 2^8] \times [1, 2^7]$ (in the coordinates of the lattice defined by special $\mathfrak{p}$). The computation of basis for the sieve ideals within $\mathfrak{p}$ took approx. 50s for each special $\mathfrak{p}$, and the sieve took 35s (on average) for a single plane. Thus the total sieve time per single $\mathfrak{p}$ was 1.3 hour. This time can be reduced, if we stop the sieve immediately after a suitable equation is found.

The lattice sieve for 28-bit unknown $\mathfrak{p}_1$ produced a B-smooth equation (not accounting for $\mathfrak{p}_1$), allowing us to compute its virtual logarithm immediately. For larger primes, we have found only semi-smooth equations, with 2 large factors each, so we needed to repeat the sieving with these new unknown primes. We have always finished the whole sieve area, and chosen the semi-smooth equation with the smallest factors for the next step in the descent. After finding a B-smooth equation, we computed originally sought virtual logarithms by backtracking. To find all 4 required virtual logarithms, have used 14 lattice sieves in 3 levels of descent (4, 6, and 4 sieves respectively). The last level sieve time was reduced by sieving only 64 planes. The total sieve time was 62160s (ca. 17 hours). Along with LLL reductions and factorization of norms for pairs a, b the total time to compute the desired individual logarithm was approx. 20 hours, i.e. the individual logarithm stage took 2% of the total NFS time (incl. the initial sieve and matrix phase).

The final solution was[8]

[8]We can verify it by computing $(g')^{\log_g q} = q^e$.

TABLE 8.13. Smoothness bound and factor base size.

$\log_2 p$	$\log_2 B$	p	B	Sieve Ideals
24	17.8	16102169	250000	43828
28	19.0	193224089	600000	97816
32	20.3	3147768119	1300000	200137
36	21.3	38939741891	3000000	365666
40	22.6	1081034284409	6532326	893707

$$\log_g q = 254468168507936021353212 \mod q.$$

To further compare the results we have created a simple program to compute discrete logarithms using Pollard's rho method. It was using NTL library, and the implementation technique was similar to the techniques used for implementing the NFS and the Lanczos algorithm. The program was able to compute 2^{20} iterations of the algorithm on average in 24.4s (on the same computer as the single-computer sieving and Lanczos was performed). The expected number of iterations required to compute the same DLP was 2^{40}. Thus it would take approximately 10 months to compute a single DLP, which we were able to compute via NFS in about 1 month (if the computation was run on a single computer). The advantage of the NFS is more pronounced for larger p's, as its complexity grows subexponentially, while the complexity of Pollard's rho method grows exponentially.

8.4.6. Summary of sieving results. To observe the scaling of the NFS we have conducted a series of experiments with increasing prime p, from 24 bits to 40 bits (the largest experiment described in more details above). In every experiment the smoothness bound was chosen near $B = L_{p^6}\left(1/3, (8/9)^{1/3}\right)$ (summarized in the Table 8.13). Block sieving was used with the fixed sieve tolerance $T = 40$ (using experimental results from Section 8.3), and with the block size 128. Only ideals of degree 1 having norm above 128 were used in the sieve. Sieve polynomials f_1, f_2 were chosen in such a way, that $[\mathcal{O}_{K_1} : \mathbb{Z}[\alpha_1]] = [\mathcal{O}_{K_2} : \mathbb{Z}[\alpha_2]] = 1$, the absolute value of coefficients of f_1 was at most 2, and $D(f_2)$ was the lowest possible. Concrete polynomials are presented in Table 8.14.

The sieve region size was determined experimentally (see Section 8.2.4), in such a way as to maximize the estimated number of equations per second. The sieve region always had shape $[-2^X, 2^X] \times [-2^Y, 2^Y] \times [1, z]$. In the case of p_{32}, p_{36}, p_{40} the originally selected sieve had not produced enough equations. In these cases we also sieved the neighboring regions $[-2^X, 2^X] \times [-3 \cdot 2^Y, -2^Y] \times [1, z_-]$, and $[-2^X, 2^X] \times [2^Y, 3 \cdot 2^Y] \times [1, z_+]$. All sieve regions are presented in the Table 8.14.

Table 8.15 summarizes the results of the sieving. In the selected region the logarithm of the sieve time increases almost linearly with $\log_2 B$. However, the overall smoothness probability decreases, due to a large coefficient in f_2. This can be caused not only by the growth of norms, but also by additional resizing of the sieve regions.

TABLE 8.14. Sieve polynomials and sieve regions.

$\log_2 p$	f_1	f_2	$X, Y, z + z_- + z_+$
24	$x^6 - x^5 + x^4 - x + 1$	$f_1(x) + p$	14, 10, 162
28	$x^6 + x^5 - x^2 - x + 1$	$f_1(x) + p$	14, 11, 256
32	$x^6 + x^4 - x^3 + 1$	$f_1(x) - p$	15, 11, 256+256+256
36	$x^6 - x^4 + x^3 + x^2 - x + 1$	$f_1(x) - p$	16, 12, 512+512+448
40	$x^6 - 2x^5 + x^3 - x + 2$	$f_1(x) + p$	18, 13, 1149+452+265

TABLE 8.15. Sieving times and equation probabilities.

$\log_2 p$	$\log_2 B$	Sieve Time [s]	Equations	Eq. prob.
24	17.8	2746	48255	$4.4 \cdot 10^{-6}$
28	19.0	13636	149182	$4.3 \cdot 10^{-6}$
32	20.3	78891	220204	$1.1 \cdot 10^{-6}$
36	21.3	449744	401032	$2.5 \cdot 10^{-7}$
40	22.6	2087070	1077984	$6.7 \cdot 10^{-8}$

TABLE 8.16. Linear system size and (estimated) times for Lanczos algorithm.

$\log_2 p$	$\log_2 q$	Sieve Ideals	Equations	Unknowns	Non-zero/row (excl. maps)	Lanczos time [s]
24	48	43828	48255	41683	14.6	15692
28	56	97816	149182	95553	15.5	157160
			99998	95553	15.1	112438
32	64	200137	220204	189975	15.2	245017
36	72	365666	401032	349264	15.5	1747882
			110000	108705	50.2	251110
40	79	893707	1077984	854821	16.2	(est.) 9214833
			226059	223474	63.1	1078641

Table 8.16 summarizes the linear systems obtained after processing the sieve results. Matrices up to p_{36} were solvable by optimized Lanczos solver (uses less memory for storage and separate handling of columns with character maps). Average time per (row $\times$ column) was 10ns for unprocessed matrix. The size of the modul q did not have a significant impact on the average performance. The system for p_{40} was too large to fit in the memory, and the Lanczos time is only estimated using the above average time. This system was later reduced by our specialized SGE as described in Section 8.4.4 to a smaller, but denser linear system. For the SGE-reduced system the average time per (row$\times$ column) had risen to 21ns.

The Lanczos-only solution is 3-8 times slower than the respective sieve phase, as can be seen from Table 8.17. Moreover, this time cannot be effectively reduced by paralel execution. The time required for linear algebra phase can be reduced by using Structured Gaussian Elimination (SGE) prior to Lanczos algorithm (LA). We have observed, that using SGE always reduced the total linear algebra time, however, for small systems the time reduction is very small. Moreover, for larger systems the SGE step is necessary due

TABLE 8.17. Sieving times compared with linear algebra time.

$\log_2 p$	$\log_2 B$	Sieve Time [s]	Lanczos [s]	SGE+LA [s]
24	17.8	2746	15692	11640
28	19.0	13636	112438	21720
32	20.3	78891	245017	39519
36	21.3	449744	1747882	259870
40	22.6	2087070	9214833	1161441

TABLE 8.18. NFS time compared to Pollard's rho.

$\log_2 p$	$\log_2 q$	Sieve+SGE+Lanczos [s]	Pollard's rho [s]
24	48	14386	390
28	56	35356	6240
32	64	118410	99840
36	72	709614	1597440
40	79	3248511	25559040

TABLE 8.19. Extrapolation of sieving results for higher values of p.

$\log_2 p$	$\log_2 B$	Exp. Sieving Time
50	25	0.6 years
60	27	4 years
70	29	29 years
80	31	200 years
90	33	1400 years
100	34	3704 years

to memory restrictions. For experiments p_{24}, and p_{28}, the SGE+LA time was higher than sieving time. For p_{32}, p_{36}, and p_{40}, the SGE reduced the required linear algebra time below that of the sieving.

If we compare NFS times with (expected) running times of the Pollard's rho method (see Table 8.18), we can see that NFS is already faster for p_{36}. Unlike in Pollard's rho, after executing the sieve phase and the linear algebra phase, we can compute practically any discrete logarithm in the corresponding field much faster.

The logarithm of the sieving time scales linearly with B, when $B = L_{p^6}\left(1/3, (8/9)^{1/3}\right)$. We have used our sieve times to estimate times required for the sieve phase of DLP experiments with larger p's. These are summarized in Table 8.19. The smoothness bound B determines the size of the sieve region as well as the size of the linear system. The time required for the linear algebra phase depends mainly on the quality of the SGE. We expect, that the linear algebra time can always be reduced below the total sieve time. The main problem of these estimates is that they do not take into account the growing storage requirements (to store a larger factor base and equations). Furthermore, the experimental data cover only a short interval. Thus the extrapolations should only be used to give some basic expectations on the order of magnitude of the sieving effort.

CHAPTER 9

Conclusions

We have successfully implemented the NFS algorithm, and were able to solve the XTR-discrete logarithm problem. Preliminary computations were executed for the XTR system with 20-bit prime characteristic described in [**114**]. Further experiments focused more on the optimization of the sieve phase of the algorithm. The final largest experiment executed was the computation of discrete logarithms in $\mathbb{F}_{p^6}$ with 40-bit prime characteristic (240-bit field size). The sieve phase for this experiment took the computation cost equivalent of 266 MIPS years. If our results can be extrapolated thus far, then the sieve phase for DLP solution in XTR system with 110-bit characteristic would require 84 000 GIPS years. Such a system is equivalent to RSA-200 (the RSA-200 modulus has the same number of bits), which was solved in estimated 121 GIPS years (700-times faster) [**111**]. The discrete logarithm record in $\mathbb{F}_p$, with 532-bit field size (160 digits) was solved in est. 10 GIPS years [**56**]. The equivalent RSA-160 computation took 2.7 GIPS years [**37**], and our estimate for an equivalent XTR system is 3500 GIPS year (350, resp. 1300-times slower). The linear algebra phase for our computations is more difficult, as the involved linear systems are larger.

We do not claim that our implementation is the fastest possible, nor directly comparable to a highly optimized software used for integer factorization records. Suppose that the Infinitely Skillful Programmer [**86**] implements the NFS in three variants: to solve the integer factorization (IFP), DLP in the degree 1 field and DLP in the degree 6 field, respectively, all with the equivalent field/modulus size. The IFP version is simplified by the fact that the system of linear equations is computed modulo 2. Both degree 1 DLP and degree 6 DLP require computation of logarithmic maps, and the solution of linear system modulo a large prime. Moreover, NFS for degree 6 DLP requires two polynomials of (at least) degree 6. Thus the total degree is higher than in the degree 1 case (degree 5 + degree 1 polynomial was used for DLP-160 [**56**]). Moreover, a 3D sieve is required to obtain enough equations. This complicates the norm computation, and the whole sieving algorithm is slower. Thus, we expect that the practical total cost of NFS implementation and execution for equivalent systems is lowest for IFP, and highest for degree 6 DLP. In this respect, the XTR system is preferable to classical ones (RSA, DSA) from the security point of view.

Except of our main result, there are many NFS specific topics covered in this work. We summarize our further results in the following:

(1) Experimental results confirm the significant impact of a polynomial selection on the real-world behavior of the NFS algorithm [**116**]. In the case of NFS for DLP in degree 6 field, we have much freedom in the polynomial selection. We are seeking the irreducible degree 6 polynomial with extremely small coefficients,

such that the smoothness probability in corresponding algebraic number field is the highest possible. The experimental increase in the smoothness probability between the best and the worst polynomial was as high, as gained by the 4-times larger factor base. It is however not clear, how to select the best polynomial. The best strategy seems to be: precompute a list of polynomials ordered by the (statistically measured) smoothness probability, and choose the best one, which is irreducible also over the used $\mathbb{F}_p$.

(2) To obtain enough smooth equations to solve the degree 6 DLP, we must sieve a three dimensional sieve region, as shown in Section 8.2. Bounds of the sieve region should be chosen in such a way that norms of boundary elements are approximately the same. This leads to a skewed sieve region shape. Alternative criteria for the sieve region shape can be used for the effective sieve implementation. These criteria depend on the actual sieve algorithm implementation details. The final region shape can be adjusted by the preliminary sieving.

(3) If we are sieving a region of higher dimensions, we need to consider the influence of higher degree ideals. Actual implementation and test runs have confirmed that only a very small fraction of ideals with higher degree contributes to smooth equations, and thus can safely be omitted in the sieve algorithm. Higher degree ideals over small primes should be considered when preparing the linear system.

(4) To speed up the sieve algorithm, we can omit the small primes, and use logarithmic approximations of norms. These changes lead to a stochastic behavior of the sieve that needs to be fine tuned for the best sieve performance. Section 8.3 elaborates the general criteria, and possible testing methodology for the actual implementation.

(5) A comparison between 2D and 3D sieve results shows that there are more possible NFS equations for points $[x, y, 1]$ than for points $[x, y, 0]$. This is caused by the fact that (most of the) equations with $\gcd(x, y) = d > 1$ can safely be used in the 3D case, but in the 2D case they are linearly dependent (with the equation for point $[x/d, y/d, 0]$). Moreover, in the 3D case we can also sieve the half-plane with $y < 0$, moving the sieve region bounds nearer to the origin.

During our research we also tried to use the large prime variant of NFS. The results were, however, not satisfactory. Partial equations found by the sieve had too many new unknowns, and only a very small fraction of them survived the (costly) filtering. Much better results were obtained by using a larger factor base. The solution might be a new approach based on using the special-q sieve with an adaptive factor base, as presented in Section 7.5. A fast lattice reduction and (B-smooth) factoring algorithm is required in this case.

As a side effect of the 3D sieve we observed that the plane $[x, y, 1]$ generates more NFS equations than the plane $[x, y, 0]$. This remained true even if we restricted the search in the $z = 1$ case to $y > 0$ for better comparison, due to unusable points with $\gcd(x, y) > 1$ in $z = 0$ case. No gcd check, or modifications of the sieving, is required in $z = 1$ case. It is not difficult to modify the classical 2D sieve to sieve plane with $z = 1$ instead of $z = 0$. The number of equations obtained after this change depends mostly on the sieve polynomial(s). If we wanted to use this change in NFS for integer factorization, a different polynomial selection algorithm might be required.

Bibliography

[1] Adleman, L. A subexponential algorithm for the discrete logarithm problem with applications to cryptography. In *Proc. 20th IEEE Found. Comp. Sci. Symp.* (1979), pp. 55–60.

[2] Adleman, L. M. The function field sieve. In *Proceedings of the First International Symposium on Algorithmic Number Theory* (1994), vol. 887 of *Lecture Notes In Computer Science*, Springer, pp. 108–121. ISBN:3-540-58691-1.

[3] Adleman, L. M., and DeMarrais, J. A subexponential algorithm for discrete logarithms over all finite fields. In *Proceedings of CRYPTO'93* (1993), D. Stinson, Ed., vol. 773 of *Lecture Notes in Computer Science*, Springer, pp. 147–158.

[4] Adleman, L. M., and DeMarrais, J. A subexponential algorithm for discrete logarithms over all finite fields. *Math. Comp. 61*, 203 (1993), 1–15.

[5] Adleman, L. M., and Huang, M.-D. A. Function field sieve method for discrete logarithms over finite fields. *Inf. Comput. 151*, 1-2 (1999), 5–16.

[6] Aoki, K., Franke, J., Kleinjung, T., Lenstra, A., and Osvik, D. A kilobit special number field sieve factorization. Tech. Rep. 205, Cryptology ePrint Archive, 2007.

[7] Aoki, K., and Ueda, H. Sieving using bucket sort. In *Advances in Cryptology - ASIACRYPT 2004* (2004), vol. 3329 of *Lecture Notes in Computer Science*, Springer, pp. 92–102.

[8] Atkin, A. O. L., and Bernstein, D. J. Prime sieves using binary quadratic forms. *Mathematics of Computation 73* (2004), 1023–1030.

[9] Bach, E., and Peralta, R. Asymptotic semismoothness probabilities. *Mathematics of Computation 65*, 216 (1996), 17011715.

[10] Bernstein, D. Circuits for integer factorization: a proposal, 2001.

[11] Bernstein, D. Arbitrarily tight bounds on the distribution of smooth integers. In *Number Theory for the Millennium I* (2002), M. A. Bennett, B. C. Berndt, N. Boston, H. G. Diamond, A. J. Hildebrand, and W. Philipp, Eds., A. K. Peters, pp. 49–66.

[12] Bernstein, D. J. The multiple-lattice number field sieve. In *Detecting perfect powers in essentially linear time, and other studies in computational number theory* (1995), Thesis, University of California at Berkeley. URL: `http://cr.yp.to/papers.html`.

[13] Bernstein, D. J. How to find small factors of integers. Manuscript, 2001.

[14] Bernstein, D. J. Factoring into coprimes in essentially linear time. *Journal of Algorithms 54* (2005), 1–30.

[15] Bernstein, D. J., and Lenstra, A. K. A general number field sieve implementation. In Lenstra and Lenstra [**65**], pp. 103–126. ISBN: 978-3-540-57013-4.

[16] Bleichenbacher, D., Bosma, W., and Lenstra, A. K. Some remarks on lucas-based cryptosystems. In *CRYPTO '95: Proceedings of the 15th Annual International Cryptology Conference on Advances in Cryptology* (London, UK, 1995), Springer-Verlag, pp. 386–396.

[17] Brent, R. P. An improved monte carlo factorization algorithm. *BIT 20* (1980), 176–184. MR 82a:10007.

[18] Brouwer, A. E., Pellikaan, R., and Verheul, E. R. Doing more with fewer bits. In *Advances in Cryptology — ASIACRYPT* (1999), vol. 1716 of *LNCS*, pp. 321–332.

[19] Buchmann, J., Loho, J., and Zayer, J. An implementation of the general number field sieve. In *CRYPTO '93: Proceedings of the 13th annual international cryptology conference on Advances in cryptology* (New York, NY, USA, 1994), Springer-Verlag New York, Inc., pp. 159–165.

[20] Buhler, J. P., Lenstra, H. W., and Pomerance, C. Factoring Integers with the Number Field Sieve. In Lenstra and Lenstra [**65**], pp. 50–94. ISBN: 978-3-540-57013-4.

[21] Canfield, E. R., Erds, P., and Pomerance, C. On a problem of oppenheim concerning "factorisatio numerorum". *J. Number Theory 17* (1983), 1–28.

[22] Cavallar, S., Dodson, B., Lenstra, A. K., Lioen, W. M., Montgomery, P. L., Murphy, B., te Riele, H., Aardal, K., Gilchrist, J., Guillerm, G., Leyland, P. C., Marchand, J., Morain, F., Muffett, A., Putnam, C., Putnam, C., and Zimmermann, P. Factorization of a 512-bit RSA modulus. In *Theory and Application of Cryptographic Techniques* (2000), pp. 1–18.

[23] Cohen, H. *A Course in Computational Algebraic Number Theory*. Springer-Verlag, 1993. ISBN: 978-3-540-55640-4.

[24] Commeine, A., and Semaev, I. An algorithm to solve the discrete logarithm problem with the number field sieve. In *Public Key Cryptography* (2006), M. Yung, Y. Dodis, A. Kiayias, and T. Malkin, Eds., vol. 3958 of *Lecture Notes in Computer Science*, Springer, pp. 174–190.

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

[26] Coppersmith, D. Modifications to the number field sieve. *Journal of Cryptology 6*, 3 (March 1993), 169–180.

[27] Coppersmith, D., Odlyzko, A. M., and Schroeppel, R. Discrete logarithms in $GF(p)$. *Algorithmica 1*, 1 (1986), 1–15.

[28] Crandall, R., and Pomerance, C. *Prime Numbers: A Computational Perspective*. Springer, 2001. ISBN 0387947779.

[29] de Bruijn, N. G. On the number of positive integers $\leq x$ and free of prime factors $> y$ ii. *Indag. Math., 38* (1966), 239247.

[30] Dickman, K. On the frequency of numbers containing primes of a certain relative magnitude. *Arkiv för Matematik, Astronomi och Fysik 22* (1930), 1–14.

[31] Diffie, W., and Hellman, M. New direction in cryptography. *IEEE Trans. Info. Theory. IT 22*, 1-2 (1976), 644–654.

[32] Dodson, B., and Lenstra, A. NFS with four large primes: An explosive experiment. In *Advances in Cryptology - CRYPTO '95: 15th Annual International Cryptology Conference* (1995), vol. 963 of *Lecture Notes In Computer Science*, Springer-Verlag, pp. 372–385. ISBN:3-540-60221-6.

[33] ElGamal, T. A public key cryptosystem and a signature scheme based on discrete logarithms. *IEEE Transactions on Information Theory 31*, 4 (July 1985), 469–472.

[34] Elkenbracht-Huizing, M. An implementation of the number field sieve. *Experimental Mathematics 5* (1996), 231–253.

[35] Elkenbracht-Huizing, R. M. A multiple polynomial general number field sieve. In *ANTS-II: Proceedings of the Second International Symposium on Algorithmic Number Theory* (London, UK, 1996), Springer-Verlag, pp. 99–114.

[36] Floyd, R. W. Nondeterministic algorithms. *J. ACM 14*, 4 (1967), 636–644.

[37] Franke, J. RSA-160. E-mail announcement, April 2003.

[38] Franke, J., and et al., T. K. RSA-576. E-mail announcement, December 2003.

[39] Franke, J., and Kleinjung, T. Continued fractions and lattice sieving. In *Special-Purpose Hardware for Attacking Cryptographic Systems SHARCS 2005, Paris* (2005).

[40] Franke, J., Kleinjung, T., Paar, C., Pelzl, J., Priplata, C., and Stahlke, C. Shark: A realizable special hardware sieving device for factoring 1024-bit integers. In *Cryptographic Hardware and Embedded Systems CHES 2005* (2005), vol. 3659 of *Lecture Notes in Computer Science*, Springer, pp. 119–130.

[41] Gao, S., and Howell, J. A general polynomial sieve. *Des. Codes Cryptography 18*, 1-3 (1999), 149–157.

[42] Geiselmann, W., Januszewski, F., Köpfer, H., Pelzl, J., and Steinwandt, R. A simpler sieving device: Combining ECM and TWIRL. In *Information Security and Cryptology ICISC 2006* (2006), vol. 4296 of *Lecture Notes in Computer Science*, Springer, pp. 118–135.

[43] Geiselmann, W., and Steinwandt, R. Non-wafer-scale sieving hardware for the NFS: Another attempt to cope with 1024-bit. In *Advances in Cryptology - EUROCRYPT 2007* (2007), vol. 4515 of *Lecture Notes in Computer Science*, Springer, pp. 466–481.

[44] Golliver, R. A., Lenstra, A. K., and McCurley, K. S. Lattice sieving and trial division. In *ANTS-I: Proceedings of the First International Symposium on Algorithmic Number Theory* (London, UK, 1994), Springer-Verlag, pp. 18–27.

[45] Gong, G., and Harn, L. Public-key cryptosystems based on cubic finite field extensions. *IEEE Transactions on Information Theory 45*, 7 (November 1999), 2601–2605.

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

[47] Gordon, D. M., and McCurley, K. S. Massively parallel computation of discrete logarithms. *Lecture Notes in Computer Science 740* (1993), 312–323.

[48] Granger, R., and Vercauteren, F. On the discrete logarithm problem on algebraic tori. In *CRYPTO 2005* (2005), V. Shoup, Ed., vol. 3621 of *Lecture Notes in Computer Science*, Springer, pp. 66–85.

[49] Hestenes, M. R., and Stiefel, E. Methods of conjugate gradients for solving linear systems. *J. Res. Nat. Bureau of Standards 49* (1952), 409–436.

[50] Horsley, S. The sieve of eratosthenes. being an account of his method of finding all the prime numbers. *Philosophical Transactions (1683-1775) 62* (1772), 327–347.

[51] Joux, A., and Lercier, R. The function field sieve is quite special. In *ANTS-V: Proceedings of the 5th International Symposium on Algorithmic Number Theory* (London, UK, 2002), Springer-Verlag, pp. 431–445.

[52] Joux, A., and Lercier, R. Improvements to the general number field sieve for discrete logarithms in prime fields: a comparison with the gaussian integer method. *Mathematics of Computation 72* (2003), 953–967.

[53] Joux, A., and Lercier, R. Discrete logarithms in $GF(370801^{30})$ – 168 digits – 556 bits. Tech. rep., NMBRTHRY list, November 2005.

[54] Joux, A., and Lercier, R. The function field sieve in the medium prime case. In *Advances in Cryptology - EUROCRYPT 2006* (2006), vol. 4004 of *Lecture Notes in Computer Science*, Springer-Verlag, pp. 254–270.

[55] Joux, A., Lercier, R., Smart, N., and Vercauteren, F. The number field sieve in the medium prime case. In *Advances in Cryptology - CRYPTO 2006* (2006), vol. 4117 of *Lecture Notes in Computer Science*, Springer-Verlag, pp. 326–334.

[56] Keinjung, T. Discrete logarithms in $GF(p)$ — 160 digits. Tech. rep., NMBRTHRY list, February 2007.

[57] Kleinjung, T. On polynomial selection for the general number field sieve. *Math. Comp. 75* (2006), 2037–2047.

[58] LaMacchia, B. A., and Odlyzko, A. M. Computation of discrete logarithms in prime fields. *Designs, Codes and Cryptography 1*, 1 (1991), 47–62.

[59] LaMacchia, B. A., and Odlyzko, A. M. Solving large sparse linear systems over finite fields. In *CRYPTO '90: Proceedings of the 10th Annual International Cryptology Conference on Advances in Cryptology* (London, UK, 1991), Springer-Verlag, pp. 109–133.

[60] Lanczos, C. Solution of systems of linear equations by minimized iterations. *J. Res. Nat. Bureau of Standards 49* (1952), 33–53.

[61] Lenstra, A., Lenstra, H. J., Manasse, M., and Pollard, J. The number field sieve. In *Proceedings of the 22nd Annual ACM Symposium on Theory of Computing* (1990), pp. 564–572.

[62] Lenstra, A., Tromer, E., Shamir, A., Kortsmit, W., Dodson, B., Hughes, J., and Leyland, P. Factoring estimates for 1024-bit RSA modulus. In *Advances in Cryptology - ASIACRYPT 2003* (2003), vol. 2894 of *Lecture Notes in Computer Science*, Springer, pp. 55–74.

[63] Lenstra, A., and Verheul, E. An overview of the XTR public key system. In *Publickey cryptography and computational number theory (Warsaw, 2000)* (2001), de Gruyter, Berlin, pp. 151–180.

[64] Lenstra, A. K. SNFS versus (G)NFS and the feasibility of factoring a 1024-bit number with SNFS. EIDMA-CWI Workshop on Factoring Large Numbers, December 2003. PPT presentation.

[65] Lenstra, A. K., and Lenstra, H. W., Eds. *The Development of the Number Field Sieve*, vol. 1554 of *Lecture Notes in Mathematics*. Springer-Verlag, Berlin, 1993. ISBN: 978-3-540-57013-4.

[66] Lenstra, A. K., Lenstra, H. W., and Lovász, L. Factoring polynomials with rational coefficients. *Mathematische Annalen 261*, 4 (December 1982), 515–534.

[67] Lenstra, A. K., Lenstra, H. W., Manasse, M. S., and Pollard, J. M. The number field sieve. In Lenstra and Lenstra [**65**], pp. 11–42. ISBN: 978-3-540-57013-4.

[68] Lenstra, A. K., and Shamir, A. Analysis and optimization of the twinkle factoring device. In *EUROCRYPT* (2000), pp. 35–52.

[69] Lenstra, A. K., Shamir, A., Tomlinson, J., and Tromer, E. Analysis of bernstein's factorization circuit. In *ASIACRYPT '02: Proceedings of the 8th International Conference on the Theory and Application of Cryptology and Information Security* (London, UK, 2002), Springer-Verlag, pp. 1–26.

[70] Lenstra, A. K., and Verheul, E. R. The XTR public key system. *Lecture Notes in Computer Science 1880* (2000), 1+.

[71] Lenstra, H. W. J. Factoring integers with elliptic curves. *Annals of Mathematics 126*, 2 (1987), 649–673.

[72] Leyland, P. C., Lenstra, A. K., Dodson, B., Muffett, A., and Wagstaff, S. Mpqs with three large primes. In *ANTS-V: Proceedings of the 5th International Symposium on Algorithmic Number Theory* (London, UK, 2002), Springer-Verlag, pp. 446–460.

[73] Lidl, R., and Niederreiter, H. *Introduction to finite fields and their applications.* Cambridge University Press, 1994.

[74] Lim, S., Kim, S., Yie, I., Kim, J., and Lee, H. XTR extended to $GF(p^{6m})$. In *Selected Areas in Cryptography (SAC 2001)* (2001), vol. 2259 of *Lecture Notes in Computer Science*, Springer, pp. 301–312.

[75] Magliveras, S., Stinson, D., and van Trung, T. New approaches to designing public key cryptosystems using one-way functions and trapdoors in finite groups. *Journal of Cryptology 15*, 4 (September 2007), 285–297.

[76] Matyukhin, D. V. On the asymptotic complexity of computing discrete logarithms in the field $GF(p)$. *Diskr. Mat. 15*, 1 (2003), 2849. translation in Discrete Math. Appl. 13 (2003), no. 1, 27–50.

[77] Menezes, A., van Oorschot, P. C., and Vanstone, S. *Handbook of Applied Cryptography.* CRC Press, Boca Raton, 1996.

[78] Menezes, A., Vanstone, S., and Okamoto, T. Reducing elliptic curve logarithms to logarithms in a finite field. In *STOC '91: Proceedings of the twenty-third annual ACM symposium on Theory of computing* (New York, NY, USA, 1991), ACM, pp. 80–89.

[79] Murphy, B., and Brent, R. P. On quadratic polynomials for the number field sieve. Tech. Rep. TR-CS-97-17, CS Lab, ANU, Canberra 0200 ACT, Australia, 1997.

[80] NIST. Data Encryption Standard (DES). FIPS PUB 46-2, January 1988.

[81] NIST. Digital Signature Standard (DSS). FIPS PUB 186-2, January 2000.

[82] NIST. Recommendation for pair-wise key establishment schemes using discrete logarithm cryptography. Special Publication 800-56A, March 2007.

[83] Odlyzko, A. Discrete logarithms: The past and the future. *Designs, Codes, and Cryptography 19*, 2–3 (2000), 129–145.

[84] Odlyzko, A. M. Discrete logarithms in finite fields and their cryptographic significance. In *Proc. of the EUROCRYPT 84 workshop on Advances in cryptology: theory and application of cryptographic techniques* (New York, NY, USA, 1985), Springer-Verlag New York, Inc., pp. 224–314.

[85] Pohlig, S. C., and Hellman, M. E. An improved algorithm for computing logarithms over $GF(p)$ and its cryptographic significance. *IEEE Transactions on Information Theory 24*, 1 (1978), 106–110.

[86] Pollard, J. The lattice sieve. In Lenstra and Lenstra [**65**], pp. 43–49. ISBN: 978-3-540-57013-4.

[87] Pollard, J. M. Monte Carlo methods for index computation mod p. *Mathematics of Computation 32* (1978), 918–924.

[88] Pomerance, C. A tale of two sieves. *The Notices of the Amer. Math. Soc. 43* (1996), 1473–1485.

[89] Pritchard, P. Fast compact prime number sieves (among others). *J. Algorithms 4*, 4 (1983), 332–344.

[90] Rivest, R., Shamir, A., and Adleman, L. A method for obtaining digital signatures and public-key cryptosystems. *Communications of the ACM 21*, 2 (1978), 120–126.

[91] Rubin, K., and Silverberg, A. Torus-based cryptography. In *Advances in Cryptology (CRYPTO 2003)* (2003), vol. 2729 of *Lecture Notes in Computer Science*, Springer, pp. 349–365.

[92] Schirokauer, O. Discrete logarithms and local units. *Phil. Trans. R. Soc. Lond. A 345* (1993), 409–423.

[93] Schirokauer, O. Virtual logarithms. *Journal of Algorithms 57* (2005), 140–147.

[94] Schirokauer, O. The number field sieve for integers of low weight. Tech. Rep. 107, Cryptology ePrint Archive, 2006.

[95] Schirokauer, O., Weber, D., and Denny, T. F. Discrete logarithms: The effectiveness of the index calculus method. In *Algorithmic Number Theory: Second Intern. Symp.* (1996), H. Cohen, Ed., vol. 1122 of *Lecture Notes in Math.*, Springer, pp. 337–362.

[96] Schnorr, C. P. Efficient signature generation by smart cards. *Journal of Cryptology 4*, 3 (1991), 161–174.

[97] Schroeder, M. *Number Theory in Science and Communication*. Springer-Verlag, 1984. ISBN 3-540-12164-1.

[98] Semaev, I. Special prime numbers and discrete logs in finite prime fields. *Mathematics of Computation 71*, 237 (2000), 363–377.

[99] Shamir, A. Factoring large numbers with the twinkle device (extended abstract). In *CHES* (1999), Çetin Kaya Koç and C. Paar, Eds., vol. 1717 of *Lecture Notes in Computer Science*, Springer, pp. 2–12.

[100] Shamir, A., and Tromer, E. Factoring large numbers with the TWIRL device. In *Proceedings of Crypto 2003* (2003), vol. 2729 of *Lecture Notes in Computer Science*, Springer-Verlag.

[101] Shamir, A., and Tromer, E. On the cost of factoring RSA-1024. *RSA CryptoBytes 6*, 2 (2003), 10–19.

[102] Shanks, D. Class number, a theory of factorization, and genera. In *Proc. Symp. Pure Math.* (1971), D. J. Lewis, Ed., vol. 20, Amer. Math. Soc., pp. 415–440. MR 47:4932.

[103] Shor, P. W. Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer. *SIAM J.SCI.STATIST.COMPUT. 26* (1997), 14–84.

[104] Silverman, R. Optimal parameterization of SNFS. *Journal of Mathematical Cryptology 1*, 2 (2007), 105–124.

[105] Sýs, M., and Novák, V. Selected part of solving sparse system over Zn2 via block lanczos apgorithm. In *International Workshop on Grid Computing for Complex Problems GCCP 2005* (December 2006), VEDA., pp. 145–151. ISBN 80-969202-1-9.

[106] Sýs, M., and Zajac, P. Discrete logarithm problem and its applications in cryptography. *Begabtenförderung im MINT Bereich 12* (2005), 129–146.

[107] Thomé, E. Computation of discrete logarithms in f2607. In *ASIACRYPT '01: Proceedings of the 7th International Conference on the Theory and Application of Cryptology and Information Security* (London, UK, 2001), Springer-Verlag, pp. 107–124.

[108] Wambach, G., and Wettig, H. Block sieving algorithms. Tech. Rep. 190, University of Cologne, 1995.

[109] Weber, D. Computing discrete logarithms with the general number field sieve. In *ANTS-II: Proceedings of the Second International Symposium on Algorithmic Number Theory* (London, UK, 1996), Springer-Verlag, pp. 391–403.

[110] Weber, D., and Denny, T. F. The solution of McCurley's discrete log challenge. In *CRYPTO '98: Proceedings of the 18th Annual International Cryptology Conference on Advances in Cryptology* (London, UK, 1998), Springer-Verlag, pp. 458–471.

[111] Weisstein, E. W. RSA-200 factored. MathWorld Headline News., May 10 2005.

[112] Wiedemann, D. H. Solving sparse linear equations over finite fields. *IEEE Trans. Inf. Theor. 32*, 1 (1986), 54–62.

[113] Zajac, P. Generalized line sieve algorithm. In *Proceedings of ELITECH '07* (2007).

[114] Zajac, P. How to solve XTR-DL using NFS. In *Mikulášska Kryptobesídka* (2007).

[115] Zajac, P. Remarks on the NFS complexity. Submited to TATRA MOUNTAINS Mathematical Publications, 2007.

[116] Zajac, P. Smoothness probability in degree six number fields. *Journal of Electrical Engineering 58*, 7/s (2007), 14–16.