INDIAN STATISTICAL INSTITUTE

Aspects of Index Calculus Algorithms for Discrete Logarithm and Class Group Computations

by

Madhurima Mukhopadhyay

under the supervision of
Prof. Palash Sarkar

A thesis submitted in partial fulfillment for the
degree of Doctor of Philosophy

Applied Statistics Unit
Indian Statistical Institute, Kolkata

September 2021

Acknowledgements

"***I can no other answer make but thanks,***
And thanks; and ever thanks;"
-William Shakespeare.

At the end of the beautiful educational journey of Doctorate studies, it would be my pleasure to thank everyone whose contribution has made my dream to see the light of the day. I am highly indebted to each one of them.

First and foremost, I would like to express my deep gratitude to my supervisor Prof. Palash Sarkar for providing me with the privilege to work under his direction. Undoubtedly, this thesis would not have come to reality without his active efforts. Within his busy schedule, he had always provided me with the opportunity to discuss any idea or improvement which came to my mind. The last year of my PhD tenure was the time when the entire world was gripped by the pandemic. Despite that, I did not face any difficulty academically due to his continuous support whenever needed. His optimistic attitude had been my inspiration during the tough times of research that had instilled within me a positive view of things. His wonderful ability to look into the core of the matter and provide a broader perspective to each of the underlying issues had given clarity to my thoughts related to the works that we undertook. Discussions of any form had been rewarding and no amount of thanks would be sufficient to express explicitly how much I owe him for all his thoughtful suggestions and endeavours.

I also wish to express my sincere acknowledgement to the co-authors of each of my papers. Prof. Sarkar is the co-author of all the research works included in this thesis. Apart from his tremendous effort in each of the papers, this thesis contains a paper (Chapter 5) co-authored by Emmanuel Thomé (INRIA senior research scientist) and Shashank Singh (IISER, Bhopal). I would like to appreciate the interest taken by Thomé to do the linear algebra part of the reported calculation in that paper. It was carried out on the French Grid'5000/SILECS experimental testbed, supported by a scientific interest group hosted by Inria and including CNRS, RENATER and several universities as well as other organizations. Some of the computational resources used on the SILECS/Grid'5000 testbed were also funded by the CPER cyberentreprises project. Regards to Shashank Singh for always helping me with suggestions about the research works and various computational aspects. I would also remain grateful to him for hosting the visit to IIT-Kanpur where we had some fruitful discussions regarding various aspects of the research work concerned with record discrete log computation. Also, I would wish to thank him for helping me with some technical aspects during the time of revision of the thesis.
I would also like to thank the anonymous reviewers of my works for providing their valuable

feedback which in some cases lead to the betterment of the work.

I would be much obliged to recognise the useful advice regarding working in Magma provided by the members of the Computational Algebra Group, University of Sydney. Particularly, I wish to mention the names of Geoff Bailey, John Cannon, Claus Fieker, Allan Steel, Don Taylor. Their timely suggestions via email helped me to learn Magma and attempt several intricate computations with it. I had received tips regarding various theoretical, as well as computational issues of class group computation from Alexandre Gélin. It was a series of discussions via email centered around his work related to class groups. I wish to thank him for that. I would also like to recognise the suggestions about some theoretical matters associated with class group computation provided by Jean-François Biasse via email.

I got admission to ISI, Kolkata with a background in Pure Mathematics. I would like to express my heartfelt thanks to all faculty members whose lessons during the course work period of research helped to gather some pre-requisites required to perform research in Cryptography. The placid and tranquil atmosphere at Indian Statistical Institute, Kolkata also provided a suitable work environment to focus on the work at hand. Along with that, I would also like to acknowledge the assistance delivered by the various offices of the institute, especially CSSC and ASU office when needed. Additionally, I wish to thank friends over here for offering their company.

The academic voyage through various phases of student life would not have been complete without the assistance of teachers from school (Carmel High School, Kolkata), college (St. Xavier's College, Kolkata) and university (Department of Pure Mathematics, University of Calcutta). I shall remain highly indebted to each one of them for all that they have imparted me with.

The greatest regards goes to my parents: Mr. Pradip Kumar Mukhopadhyay and Mrs. Kabita Mukhopadhyay. Their selfless love and lifelong sacrifices have played a key role to fulfil all my aspirations. I would have never made it this far without their love and support. No expression of my admiration for them would be enough to express the endless support that I received and continue to do so from them.
A big thanks to my dearest friend Anwoy for all efforts, suggestions and the words of constant encouragement.

I would specially like to express my heartfelt gratitude to Prof. Chandan Mazumdar, Jadavpur University for always motivating me. All his valuable advices were extremely helpful for me since my schooldays. I would like to gratefully acknowledge late Dr. Amit Chaudhuri for consistently encouraging me to do better.

Stimulation and inspiration play a pivotal role to continue the amazing expedition of learning and research. There were various articles, biographies online as well as books, which helped me to stay motivated. The simple act of understanding the gradual journey of science and the efforts undertaken by great men and women devoted to their subject, has helped to gather inspiration. Reading supplied the enthusiasm required to continue the scientific ride.
My maternal uncle, aunt and my paternal uncle could not live long to see this day. Indeed, they would have been quite happy to see me pursuing and completing my Doctorate studies. I wish to express my deep gratitude to them for all the warmth they had given me.

As a concluding note for this section, I wish to say thanks to each and everyone whose support, advice have helped me along the way.

Contents

Abbreviations

DLP	**D**iscrete **L**og **P**roblem
FFS	**F**unction **F**ield **S**ieve
QPA	**Q**uasi **P**olynomial **A**lgorithm
NFS	**N**umber **F**ield **S**ieve
ECDLP	**E**lliptic **C**urve **D**iscrete **L**og **P**roblem
HCDLP	**H**yperelliptic **C**urve **D**iscrete **L**og **P**roblem
BSGS	**B**aby **S**tep **G**iant **S**tep algorithm
CRT	**C**hinese **R**emainder **T**heorem
HNF	**H**ermite **N**ormal **F**orm
SNF	**S**mith **N**ormal **F**orm

Symbols

p	a prime number
Q	p^n, where n is a positive integer
$\mathbb{F}_{q^n}$	a finite field of order q^n
$\mathbb{F}_{q^n}^\star$	$\mathbb{F}_{q^n} \setminus \{0\}$
$\mathbb{F}_Q$	a finite field of order Q
G	a group
$\#G$	order of group G
$\langle \mathfrak{g} \rangle$	a subgroup generated by g
$L_Q(a,c)$	$\exp\left((c+o(1))(\ln Q)^a(\ln\ln Q)^{1-a}\right)$
$L_Q(a)$	$L_Q(a,c)$ for some c
$\log_g h$	discrete logarithm of h to base g
$\mathcal{K}$	a number field
$\mathcal{O}_\mathcal{K}$	ring of integers of the number field $\mathcal{K}$
$\Delta_\mathcal{K}$	discriminant of a number field $\mathcal{K}$
$Cl(\mathcal{O}_\mathcal{K})$	class group of $\mathcal{K}$ or $\mathcal{O}_\mathcal{K}$
$\mathcal{N}(\mathbf{a})$	norm of an ideal $\mathbf{a}$

Introduction

"***You cannot teach a man anything, you can only help him find it within himself.***"
- Galileo Galilei.

1.1 Aim of the Thesis

This thesis focuses on two problems: The discrete logarithm problem and the class group computation problem. Besides the inherent mathematical appeal, both of these problems have a strong cryptographical interest.
In particular, this thesis revolves around the study of index calculus algorithms for discrete logarithms in finite fields of small and medium characteristic and for performing the relation collection phase of class group computation. An improvement in the case when tag tracing is used in Pollard's Rho algorithm for computation of discrete logarithm for prime order fields is also noted.

1.2 The Problems

Problem 1. Discrete Logarithm Problem (DLP) : Let G be a finite cyclic group and g be a generator of G. The discrete logarithm problem (DLP) in G is the following. Given a non-zero element h of G, find i such that $g^i = h$. This i is called the discrete logarithm of h to base g which is written as $i = \log_g h$.
It may be noted that the description of the problem assumes that g is the generator of the group G. It is also implicitly assumed that the order of the group, along with an efficient algorithm for performing group operations is known.

Problem 2. Class Group Computation: Given a number field $\mathcal{K}$ with ring of integers $\mathcal{O}_\mathcal{K}$, compute the class group $Cl(\mathcal{O}_\mathcal{K})$ of $\mathcal{K}$ or $\mathcal{O}_\mathcal{K}$.

1.3 Complexity Analysis of the Problems

The running time of any algorithm to tackle the problems is measured in terms of the input size. Let Q be the cardinality of the considered group. By input size, we mean the total number of bits used to represent the input in ordinary binary notation. The $L_Q(a, c)$ notation is used to express the complexity of the algorithm concretely.

1.3.1 $L_Q(a, c)$ Notation

The L_Q function may be viewed as a way of interpolating between polynomial ($(\log Q)^c$) and exponential (Q^c) time. The L_Q function is defined as

$$L_Q(a, c) = \exp((c + o(1))(\log Q)^a(\log \log Q)^{(1-a)}) \tag{1.1}$$

where $0 \leq a \leq 1$ and c is a positive constant.
It may be seen that the double logarithm of the L_Q function is a linear combination of the double logarithms of the polynomial and exponential time. The value of the function at the two extreme values of a is a polynomial (at $a = 0$) and exponential (at $a = 1$) in $\log Q$. When $0 < a < 1$, then the algorithm is said to run in subexponential time. The aim would be to achieve lower values of a and c to get better complexities. Sometimes, the second constant is simply ignored and $L_Q(a)$ is used to mean $L_Q(a, c)$ for some c.

A problem is called *easy* in a particular context known there exists some algorithm to solve it in polynomial time. If no polynomial time method is found for a problem, it is said to be *hard*. Problems which are easy in one group can become hard in another. The degree of hardness is quantified according to the complexity of the algorithm for solving it.

1.4 The Discrete Logarithm Problem

The computation of discrete logarithms is known to be a hard problem in general. Gauss [68] mentioned the term *index* for discrete logarithms. The most naive method that may be used to solve this problem is to keep on raising g to different powers until the desired target h is reached. This method, although guaranteed to work, is not computationally practical when the group size is large.
The notion of a problem being easy or hard in a particular context has already been defined in section 1.3. Below we enlist some groups and the nature of the problem over that group.

1.4.1 Discrete Logarithms over Well-known Groups

1.4.1.1 Hard Groups

The discrete logarithm problem is computationally hard over many well-known groups. Some examples are: large-sized finite fields of the form $\mathbb{F}_{p^n}^*$, subgroups of the multiplicative group of a considerable-sized finite field, group of points on an elliptic curve over a finite field [64], divisor class group of fixed genus hyperelliptic curves.
Finite fields of medium to large characteristics are an example where DLP is computationally hard. Jacobians of algebraic curves, class groups of number fields provide other examples.
Depending on the situation over which DLP is considered, it is called discrete logarithm problem over finite field or elliptic curve discrete logarithm problem (ECDLP) or hyperelliptic curve discrete logarithm problem (HCDLP).
In HCDLP, computation of discrete log is attempted in the jacobian of the curve. Koblitz [112] proposed utilisation of hyperelliptic curves in cryptography. Koblitz [111] and Miller [128] recommended use of elliptic curves in cryptography. The complexity of implementing the group operation depends essentially on the product of the size of the base field and the genus. Implementing genus two over a given field is roughly equivalent to implementing elliptic curve cryptography over a field of twice the bitlength.

1.4.1.2 Easy Groups

The DLP is easy over additive groups $(\mathbb{Z}_N, +)$ for some integer N. The logarithm in case of such groups is easily found in polynomial time by applying the Euclidean algorithm. Some other examples of easy groups are elliptic curves of trace one [164, 152], hyperelliptic curves with trace one [148] etc.

1.4.2 Applications in Cryptography

The intractability of the discrete logarithm problem in certain groups leads to schemes that revolutionized cryptography. In the earlier half of twentieth-century, cryptography used was mostly symmetric key encryption techniques. Such methods had some security, but had practical drawbacks too. The key to encrypt would be the key to decrypt. Thus, it should be known by the parties before exchanging the message. Exchange of such keys over an insecure channel left it prone to the attack of any adversary observing the communications secretly.

A solution to all these problems was found by Diffie and Hellman [54] in 1976. They explained how two parties can agree on a secret number over an insecure channel without the possibility for a third party to recover this number. This seminal development, which led to asymmetric or

public-key cryptography was based on the hardness of the discrete log problem.

The difficulty of the discrete logarithm problem is widely used to perform secure communication over the Internet. Most of the present public key cryptosystems are based on the hardness of either discrete log problem or integer factorization problem. The Diffie-Hellman key agreement protocol is based on the difficulty of DLP. ElGamal [55] discovered that this hardness can also be used for encryption and signature. Schnorr [156] proposed an identification protocol based on a zero-knowledge proof of knowledge of a discrete logarithm, which can be converted into Schnorr's signature scheme using the Fiat-Shamir transform [58]. The Paillier's encryption scheme [139] is a cryptosystem based on DLP in a group of unknown order.
Pairing-based cryptography made the advent of new schemes whose security is dependent on the hardness of DLP. The primitives such as one-round tripartite key-agreement of Joux [95], identity-based non-interactive key distribution [150], identity-based encryption schemes [27], short signature schemes [28], the traitor tracing scheme of Mitsunari, Sakai and Kasahara [129], identity-based signature schemes [42], Boneh-Lynn-Shacham short signature scheme [29] have their security based on the pairing assumption which again relies on DLP.
The NIST standard Digital Signature Algorithm [106] is also based on modified zero-knowledge proof of knowledge of discrete logarithm.

1.4.3 Brief Introduction to Algorithms for Attacking DLP

The emphasis in this thesis has been mentioned before. In case of DLP, the works are under the broad category of index calculus techniques. An improvement in the tag tracing variant of Pollard's Rho applied to prime order fields is also proposed. In this section, a short exposure to the existing strategies is given considering the central theme into account. The previous algorithms for ECDLP, HCDLP and DLP in other high genus curves are noted briefly without delving into details. The central point of this section remains the application of index calculus methods to compute discrete logarithms in finite fields of small and medium characteristic and generic algorithms for solving DLP.

The known algorithms for DLP can be categorised into two major categories, namely, *generic algorithms* (presented in chapter 2) which work in any group and *index calculus algorithms* (as in chapter 4) which work in a special class of groups. The time complexity of the first class is exponential while for the latter is subexponential.

1.4.3.1 Generic Model and Best Complexity

Nechaev and Shoup [136, 162] introduced the *generic model* where for a group G given as a *black box*, algorithms can only perform basic operations like applying group law, calculating inverses and testing equality. The computational model of Nechaev permits the algorithm to do group operations and equality tests, but does not allow any other operations. No encoding of group elements was accepted by this model. The lower bound was on the number of group operations and was valid for groups of prime order. This model was deterministic. Shoup extended the result to a broader class of algorithms. Shoup's model allows randomised algorithms. Both the models propose that the lower bound on the complexity of generic algorithms is the square root of the group order [136, 162]

1.4.3.2 Generic Algorithms

The best complexity in terms of time and memory is about the square root of the group order with linear storage. *Pollard's Rho algorithm* seems to be the best generic algorithm as it satisfies both the conditions. *Tag tracing* under which group operations can be postponed and later done together in one step, and the concept of *distinguished points* accelerate Pollard's rho algorithm [43] further. In Chapter 3, we have suggested improvements in the case of prime order fields when tag tracing is used to perform Pollard's Rho algorithm.

1.4.3.3 Index Calculus Strategies

The index calculus method is the best tactic to compute discrete logarithms. The three main variants of index calculus apply to three different types of finite field with small, medium and large characteristic. A rough estimate of the complexities are $L_Q(\frac{1}{3})$ for number field sieve and function field sieve, and $\log(Q)^{o(\log(\log(Q)))}$ for quasi-polynomial algorithms where Q is the cardinality of the considered group. Each index calculus method again consists of four main steps, namely: precomputation of the factor basis, relation collection, linear algebra and individual logarithm phase. The function field sieve applies to both medium and small characteristic. In the medium prime case, Joux-Lercier variant [100] is most commonly used. Joux [99] introduced the important idea of pinpointing. Sarkar and Singh [151] developed a sieving method relying on divisibility and smoothness technique. This method combined with the pinpointing strategy lead to practical improvements. Solid accelerations happened in the relation collection and individual descent phases. The 2-1 descent, which is the most crucial step of the descent algorithm improved due to usage of the walk technique [151] along with the utilisation of additional degrees of freedom. All these methods were utilised by us [Chapter 5] to perform record discrete log computation. The strategy adopted in our work may be applied to arbitrary fields. Even for higher sized characteristic with moderate extension degrees, such a method may be adopted for relation collection and descent.

The presence of computational resources to perform linear algebra then suffices to perform discrete logarithm computations in larger fields too.
The last step of discrete logarithm computation is composed of individual logarithm and descent. For small characteristic fields with composite extension degrees, Waterloo algorithm [24, 25] was used to perform the initial splitting step. Guillevic [84] had proposed another algorithm for initial splitting which is better than Waterloo. We [Chapter 6] have proposed an algorithm that completes initial splitting with lesser cost.

There are various variants of Number field sieve (NFS), namely: the basic version of NFS, the multiple number field sieve (MNFS), the tower number field sieve [14], extension of tower number field sieve called exTNFS [107, 108] in case the extension degree of the finite field is composite, special number field sieve [104] for primes having special form. MNFS itself again has two variants namely, the asymmetric one [50] which is the one most generally used and the symmetric case [15].

Barring some weak cases, the attacks on ECDLP are always exponential. Due to this, it is quite appropriate to use elliptic curves in various cryptographic schemes.
Adleman, DeMarrais and Huang [7] suggested the first subexponential algorithm for hyperelliptic curves. For genus greater than the size of the base field, the time complexity was subexponential. Later they [6] proposed another variant. Gaudry [65] suggested an alternative variant of Adleman-Huang algorithm with the assumption that the group order is known. Nicolas Thériault [172] recommended an improvement to Gaudry's algorithm using the idea of large primes. The runtime was improved further by Gaudry *et al.* [66]. This led to DLP in some higher genus curves vulnerable to index calculus algorithms [66]. There are similar results for non-hyperelliptic curves [57, 53].

1.5 Class Group Computation

The ideal class group of an algebraic number field $\mathcal{K}$ is the group of fractional ideals of $\mathcal{K}$ modulo principal ideals. This ideal class group of the ring of integers (which is a maximal order of the number field) is a finite abelian group and can be decomposed as $Cl(\mathcal{O}_\mathcal{K}) = \oplus_i \mathbb{Z}/d_i\mathbb{Z}$ with $d_i | d_{i+1}$. The class group computation problem is quite important and relevant as it provides information about the structure of multiplication in the field. However, computationally it remains a difficult problem to date. The structure of the class group is obtained from the d_i's. A set of prime ideals whose norms are bounded by some integer B, helps in computing a matrix. A linear algebra phase on this matrix computes the d_i's. The detailed procedure to compute the class group is given in Section 7.3 of this thesis. Class group computation is one of the four major problems [143, 142] in computational algebraic number theory postulated by Zassenhaus (the others being computation

of unit group, ring of integers, Galois group).

1.5.1 Mathematical Perspective

Interestingly, ideal class groups were studied even before the formulation of the concept of ideal. It was first recognised by Gauss [67] while working with class numbers of quadratic fields. The formalisation of the idea of class group along with its connection with the non-uniqueness of factorization in the ring of integers of the associated number field was studied by Kummer ([144], Section 4.1) in the context of cyclotomic fields. Based on the work of Kummer in cyclotomic fields, Dedekind developed the theory of the ideal class group for quadratic field extensions and generalized it to all number fields [72].
Computation of class group provides numerical confirmation to unproven conjectures like heuristics of Cohen and Lenstra [47] on the ideal class group of a quadratic number field, Littlewood's bounds [123], Bach bound [11] which is the minimal bound such that all prime ideals having norm below it generates the ideal class group. Class groups play a significant role in computing the Mordell-Weil group of elliptic curves with the descent method or the Brauer group computations for representation theory [59].
Developments in the course of research in class groups led to further progress in other branches of algorithmic number theory as well. The baby-step-giant-step algorithm of Shanks' [159] was initially discovered for computing class groups of imaginary quadratic number fields. Factorization algorithms were also obtained from works regarding class group computation ([45], Sections 8.6 and 10.2). Also, Kummer's work [144] played a significant role in the completion of Fermat's last theorem.

1.5.2 Cryptographical Viewpoint

Cryptosystems were initially described in the context of imaginary quadratic number fields due to large class groups with size approximately about the square root of the discriminant of the field. It has already been remarked in Section 1.4.1.1 of this thesis that DLP is a hard problem in class groups. Using this intractability, Buchmann and Williams [39] suggested a variant of Diffie-Hellman key exchange protocol in class groups of imaginary quadratic number fields. RDSA [86], which is a variant of the digital signature algorithm [156] is also linked with this class group computation problem. NICE (New Ideal Coset Encryption) [141] utilises ideals in imaginary quadratic number fields.
Class group computations for real quadratic number fields was also done [22, 153]. Later, higher degree number fields were employed [23, 127]. Implementing this needed identification of families of number fields having large class groups. Computing these class groups was done for various degrees of number fields [167, 35, 118].

McCurley [126] proposed the first subexponential algorithm for computing the class number of a quadratic number field. The ideal class group of an imaginary quadratic number field offered appropriate settings for the Diffie and Hellman key distribution scheme. Not much later, Hafner and McCurley [85], suggested the first rigorous proof under the Extended Riemann Hypothesis of the subexponentiality of the ideal class group computation in imaginary quadratic number fields. Buchmann [34] generalized Hafner and McCurley's algorithm to arbitrary degree number fields. They computed the ideal class group and regulator in $L_{\Delta_\mathcal{K}}(1/2, O(1))$ where $\Delta_\mathcal{K}$ denotes the discriminant of the number field $\mathcal{K}$. Alongside, systems of fundamental units of the maximal order of number fields can be computed as well in subexponential time. This led to the solution of norm equations like the Pell equation [119]. Buchman's algorithm later led to Enge's [56] subexponential algorithm for computing discrete logarithms in Jacobians of high-genus hyperelliptic curves. Knowledge about the structure of the class group can also be used to solve DLP ([90], Chapter 13).

The computation of class group is also related with cryptosystems based on the hardness of finding short vectors in ideal lattices, such as the homomorphic encryption scheme of Vercauteren and Smart [165]. Isogeny computation between abelian varieties required ideal decomposition which in turn needed the knowledge of class groups. Jao and Soukharev [91] did such computations in the case of elliptic curves. Fast techniques to derive relations in class groups of small degree number fields can also be employed for evaluating isogenies between small genus curves using methods of complex multiplication. Evaluating isogenies between genus g curves requires relations in the class group of a degree $2g$ number field. This evaluation is done by expressing the isogeny as a composition of lower degree isogenies derived from the relations. This has quite some cryptographic applications in point counting and reducing DLP in Jacobian of a curve to another group. Thus relations in the class group have wide applications in the form of computation of class group, solving DLP in the class group of that order, computation of the generator of a principal ideal in the chosen order, along with relation search in polarized class group.

The class group of $\mathbb{Z}[\theta]$, for some suitable θ, can be used for factoring large numbers or solving DLP using number field sieve algorithm [120]. In fact, relations in an arbitrary order can be used to find a solution to the discrete log problem in the class group of that order. There is no reduction known between discrete logarithm in the Jacobian of a curve and in the class group of any order. This may again be used for the purpose of building cryptosystems.

1.5.3 Short Note on the Algorithms for Computing Class Group

Several attempts were undertaken to compute class groups. The earliest efforts were for quadratic number fields while presently efficient algorithms exists for all number fields. The initial costs were exponential which later improved to subexponential. The improvements were mostly done

either in relation collection phase or to produce a *good* defining polynomial.

In 1968, Shanks's [159, 160] proposed an exponential algorithm to compute the ideal class group of an imaginary quadratic number field. It was based on the baby-step giant-step algorithm and runs in time $O(|\Delta_{\mathcal{K}}|^{\frac{1}{4}+\epsilon})$ or $O(|\Delta_{\mathcal{K}}|^{\frac{1}{5}+\epsilon})$ under the extended Riemann hypothesis [122]. Attempts to remove the restriction on degree was made by Pohst and Zassenhaus [145].

Later subexponential strategies were proposed [85, 34, 56, 57]. Cohen and Diaz y Diaz [48] suggested an algorithm for reducing defining polynomials. Implementation of Hafner and McCurley's method was done [36, 37] along with some improvements to it. Further advancements were suggested in [37, 89, 37, 168, 46].

More recently certain improvements [19, 20, 21, 71, 69, 70] produced simplified algorithms. Exhaustive classification of number fields was introduced by Gélin and Joux [71]. The latest works in this arena [69, 70] provides suitable algorithms for each kind of number field under the classification in [71].

1.6 Thesis Lay-Out

1.6.1 Chapter Organization

The thesis is subdivided into three parts. The first part is devoted to the discrete logarithm problem. The second part deals with class group computation.
Chapter 2, 4 and 7 are background chapters which provide a gateway to the research works presented in the thesis. Naturally, more stress is given on the concepts which are needed for the chapters focusing on research works. A short study of the generic algorithms for discrete log is done in Chapter 2. This is followed by an improvement of Pollard's Rho algorithm in case of prime order fields in Chapter 3. Next, the index calculus techniques for finding discrete logarithm is presented in Chapter 4. Chapters 5 and 6 provide advances in index calculus strategies to compute discrete logarithms along with the corresponding implementations. Chapter 7 provides a note regarding class group computations. An index calculus advancement to compute class groups is illustrated in Chapter 8.
The research papers provide an implementation of the newly introduced techniques. Altogether, there are *four new* research works: Chapters 3, 5 and 6 and 8. In case of DLP, leaving aside the *improvement in the tag tracing variant of Pollard's Rho for prime order fields in chapter 3, all others are improvements to scenarios where index calculus algorithms are used. Chapter 8 also provides an index calculus technique to compute class groups.*

1.6.2 Research Works on Which the Thesis Is Based

1. [132] Madhurima Mukhopadhyay and Palash Sarkar. Combining Montgomery multiplication with tag tracing for the pollards rho algorithm in prime order fields. 2021. https://eprint.iacr.org/2021/043.

2. [135] Madhurima Mukhopadhyay, Palash Sarkar, Shashank Singh, and Emmanuel Thomé. New discrete logarithm computation for the medium prime case using the function field sieve. *Advances in Mathematics of Communications*, 2020. https://www.aimsciences.org/article/doi/10.3934/amc.2020119.

3. [133] Madhurima Mukhopadhyay and Palash Sarkar. Faster initial splitting for small characteristic composite extension degree fields. *Finite Fields and Their Applications*, 62:101629, 2020. https://www.sciencedirect.com/science/article/abs/pii/S1071579719301327.

4. [134] Madhurima Mukhopadhyay and Palash Sarkar. Pseudo-random walk on ideals: Practical speed-up in relation collection for class group computation. Cryptology ePrint Archive, Report 2021/792, 2021. https://eprint.iacr.org/2021/792.

1.6.3 Background

- The first part of this **introductory chapter** provides a brief overview of the importance of both the problems. Both mathematical, as well as cryptographical aspects are covered. The techniques that have been used previously to solve them is also described in a nutshell.

- **Chapter 2** is an overview of the generic algorithms for computing discrete logarithms. A short discussion of adding walks and tag tracing for Pollard rho algorithm is also done.

- **Chapter 4** is a brief exposure to the index calculus algorithms for DLP. The L notation for complexity is utilised to categorise the finite fields in terms of the suitability of various index calculus algorithms. A general introduction to the various steps of index calculus algorithms is given along with an example. Details about linear algebra and smoothness issues are next discussed. This is followed by a description of the function field sieve when the characteristic of the field is a medium or a small prime. Previous works in these two cases are also mentioned. The Waterloo algorithm for initial splitting is given here. The chapter concludes with a very brief note about number field sieve.

- **Chapter 7** provides a short summary on the existing strategies to compute the class group. The related problems along with some more cryptographic applications in addition to those mentioned in Section 1.5.2 is also given. Presently index calculus class of algorithms are used to tackle this problem. The detailed steps of the method are discussed in the context of class group computations. The verification strategy to ensure correctness is also explicitly given. It ends with the description of some previous works.

1.6.4 Brief Note on the Research Undertaken

- **Speeding up Pollard's Rho Algorithm in Prime Order Fields with Tag Tracing and Montgomery Multiplication:** This work [132] is described in **Chapter 3**. It leads to a speed up of Pollard's rho algorithm [147] in fields of prime order at no additional cost. It is known that Pollard's rho algorithm [147] is the best generic algorithm to compute discrete logarithm. The complete discrete logarithm in a finite field is found by applying Pollard's rho in sub-problems that may arise when index calculus algorithms are applied. The tag tracing variant [43] of Pollard's rho is considered. This is done as practical improvements are possible in certain groups using this. Concrete speed-ups were also demonstrated for prime order subgroups of multiplicative groups of finite fields. An integral part of performing this algorithm is field multiplication. In a finite field, $\mathbb{F}_p$, a field multiplication consist of an integer multiplication and a reduction modulo p. For arbitrary primes lacking structural speciality, modular reduction is quite costly. A large chunk of the time necessary to perform field multiplication is due to the cost of modular reduction. We have intertwined

the technique of Montgomery multiplication [131, 30] with tag tracing. As a result, the expensive modular reductions are completely replaced by divisions by a suitable power of two. The implementation of these divisions is easy as it is just right shift operations. In doing this, we do not compromise any advantage that tag tracing offered. The essential difference is instead of doing field multiplication after a certain number of steps we just do a Montgomery multiplication after the same number of steps. The net result of our work is that a speed-up is obtained without any additional costs of memory.

- **A Record Discrete Logarithm Computation and Analysis of Previous Strategies for the Medium Prime Case Using the Function Field Sieve:** This work [135] is described in **Chapter 5**. It successfully combats the challenge to perform a larger discrete logarithm computation for a medium prime case field than what had been reported earlier without losing generality. It inspects the mechanisms and computes a record discrete log for a field with 22-bit characteristic and 1051-bit size.
  This work, reports progress in discrete logarithm computation for the general medium prime case using the function field sieve algorithm. A new discrete logarithm computation over a 1051-bit field having a 22-bit characteristic was performed. This is a record computation https://dldb.loria.fr/?filter=all&sort=date. Fields of size 1175-bit and the 1425-bit have been considered earlier [99]. But it is noteworthy that these fields, enjoyed advantage offered by the Kummer extension property. This made the factor base size small and 2-1 descent easier. In particular, for the fields considered in [99], 20-bit factor bases suffice whereas in our case a 23-bit factor basis is required. Some previously known techniques of the earlier works [99, 151] are improved as well as implemented. An increase in the size of the field makes various steps of the function field sieve more complicated. Our study delves into these difficulties. The linear algebra step along with descent form the main stumbling blocks. The various techniques of descent [151] like alternating walk and branching were implemented efficiently. This scanning reveals that the same procedures are sufficient for doing relation collection and descent for even larger fields with 32-bit characteristic and moderate extension degrees. The linear algebra step remains the most time-consuming step. The manner in which the various methods are used in this paper, is equally applicable for any general medium prime field with or without special relation between various parameters.

- **Faster Initial Splitting for Small Characteristic Composite Extension Degree Fields:** This work [133] is given in **Chapter 6**. It proposes a new algorithm for initial splitting in small characteristic fields of composite extension degree. It is better than the other existing algorithms like Waterloo algorithm [24, 25] and Guillevic's [84] initial splitting algorithm. Implementation https://github.com/Madhurima11/faster-initial-splitting of the new algorithm has also been done. Our algorithm reduces the cost of generation

of polynomials which are to be tested for smoothness. The usage of this algorithm is appropriate when attempting record discrete logarithm computations over such fields.
We consider the problem of computing discrete logarithms with function field sieve in $\mathbb{F}_{p^n}$ for a small characteristic p and composite n. The last phase of the function field sieve, namely the individual logarithm step itself again consist of two sub-steps of initial splitting and descent. The focus in this work is on the initial splitting phase. The standard algorithm for initial splitting for such fields is the Waterloo algorithm [24, 25]. Guillevic [84] proposed another algorithm for initial splitting over these fields. It enjoyed the advantage of checking the smoothness of a single polynomial instead of two required by the Waterloo algorithm. Time required for generating a polynomial by our method is significantly lower than the time for generating a polynomial using the Guillevic splitting algorithm. The smoothness probabilities and time for testing smoothness is almost the same in both the algorithms. Additionally, our algorithm is completely parallelizable compared to some sort of semi-parallelism possible for Guillevic's algorithm.

- **Practical Speed-up in Relation Collection Phase of Class Group Computation of Large Degree Fields:** This work [134] is presented in Chapter 8. Our algorithm accelerates the relation collection phase. Theoretical prediction has been confirmed by practical implementations.
  The latest work in the series of improvements to the relation collection phase was suggested by Gélin [69] for *large degree fields* (This definition of large depends on some other parameters instead of the usual sense). Broadly, the steps of Gélin's algorithm are: random generation of ideals, lattice reduction techniques to obtain a small element of the generated ideal, constructing a new ideal utilising the principal ideal generated by the small element followed by obtaining a relation if the new ideal is smooth for some parameter. The random generation of ideals in the first step of Gélin's algorithm is done by multiplying some considerable number of ideals of the factor basis which are chosen randomly. Our algorithm performs a pseudo-random walk on ideals with the help of a precomputed table of ideals. The time to pre-compute a table in the offline phase is negligible. In the online phase, each step of our walk requires a single multiplication instead of many in the first step of Gélin's algorithm. The later steps remain the same as that of Gélin's strategy. In our algorithm, there is some savings of time in each step due to the lower number of multiplications. Experiments for degrees $10, 15, 20, 25$ were done to compare Gélin's method and our algorithm. It has been seen in all these cases that the time required by Gélin's technique was about 2.86 to 3.39 times more. This suggests that practically our algorithm is about three times faster than Gélin's method in these cases though the asymptotic costs are the same. Our algorithm could also successfully compute a large number of relations for the bigger fields.

1.6.5 Concluding Chapter

- **Chapter 9** is the chapter that concludes this thesis. Along with that, some future research directions have been shown.

Part I

Discrete Log Problem over Finite Fields

Chapter 2

Generic Algorithms for the Discrete Logarithm Problem

It has already been mentioned before that algorithms for solving discrete logarithm problems are classified into generic algorithms and non-generic algorithms.
Generic algorithms for the discrete logarithm problem are those algorithms that utilise only the group operation and no other algebraic structure or special properties of the group. The generic model has been detailed in Section 1.4.3.1 of this thesis. This is the reason they can work in any cyclic group. The running time of all the generic algorithms is exponential. As discussed in Section 1.4.3.1 of this thesis, the best complexity is about the square root of the group order with constant storage.
The generic algorithms are presented in the succeeding sections. It is assumed implicitly that the group order is known or the problem of computing it is easier than the discrete logarithm problem. Also, it is assumed that multiplications, exponentiations, inverses in G can be computed in time polynomially bounded by the input size. In the descriptions that follow, the order of the group G is represented as $\#G$.

2.1 Exhaustive Search

This is the most naive algorithm. It computes the powers of g until the element h is found. The worst-case running time is $O(\#G)$ group operations and so it is highly inefficient for groups of cryptographic interest.

2.2 Shanks' Baby-Step Giant-Step Method

Shanks' Baby-Step Giant-Step (BSGS) is a time-memory trade off method as it reduces the time of brute force search in exchange of extra storage.

Let $m = \lceil \sqrt{\#G} \rceil$, then, the required logarithm can be expressed as

$$\log_g h = qm + r \tag{2.1}$$

where $0 \leq q, r < m$ where the ordering between q and m arises due to the choice of m and the ordering of r and m is attributed to Euclid's division lemma.
From equation (2.1), $g^{\log_g h} = g^{qm} g^r$, which implies,

$$h(g^{-m})^q = g^r. \tag{2.2}$$

The essential idea of the algorithm is to find q and r by searching the match of equation (2.2) by computing powers g and g^{-m}. The target logarithm can then be computed from equation (2.1). The method is outlined in the following algorithm.

Input: The generator g of G and the target h.
Output: $\log_g h$.
$m \leftarrow \lceil \sqrt{\#G} \rceil$
/*Baby steps*/
Initialize an easily searchable structure T.
Insert $(0, 1)$ and $(1, g)$ in T
Set $t = g$
for $i \leftarrow 2$ ***to*** $(m - 1)$ **do**
    $t \leftarrow t \cdot g$
    Insert (i, t) in T
end
Sort T with respect to the second coordinate.
/*Giant steps*/
Set $t = h, w = g^{-m}$
for $j \leftarrow 0$ ***to*** $(m - 1)$ **do**
    if $(i, t) \in T$ *for some* i **then**
        return $i + jm$
    else
        $t \leftarrow tw$
    end
end

Algorithm 1: Baby-step giant-step algorithm.

The BSGS algorithm is deterministic. The overall time and space complexity turn out to be $O(\sqrt{\#G})$. The large storage requirement of this algorithm restricts its application. A demonstration of the algorithm is given in [Section 11.5.1, [113]].

2.3 Pohlig-Hellman Algorithm

The Pohlig-Hellman (PH) method assumes that the prime factorization of the order of the group is known. Let $\#G = \prod_{i=1}^{k} p_i^{\alpha_i}$. The main idea in the PH algorithm is to compute the logarithms modulo the smaller prime factors p_i and find out the entire logarithm by Chinese Remainder Theorem (CRT).

For any prime p_i present in factorization of $\#G$, both g and h is projected into subgroups of order ${p_i}^{\alpha_i}$ as $g_i = g^{\frac{\#G}{{p_i}^{\alpha_i}}} \in G$ and $h_i = h^{\frac{\#G}{{p_i}^{\alpha_i}}} \in G$. Both g_i, h_i are elements of G. The order of g_i is ${p_i}^{\alpha_i}$. We next describe the strategy to obtain l_i such that $g_i^{l_i} = h_i$ holds where $l_i \equiv \log_g h \mod {p_i}^{\alpha_i}$. All such $l_i, \forall i \in [1, k]$ are utilised to find the entire logarithm by applying CRT.

The explanation given below shows how the PH method computes the discrete log modulo a single p_i with exponent α_i in the factorization of the group order. This is called the Hensel Step.

Let $p = p_i, \alpha = \alpha_i, l = l_i$. Also, let the p-ary representation of the target l be $l = d_0 + d_1 p + \ldots + d_{\alpha-1} p^{\alpha-1}$. The note given below describes the method to compute a single d_j, $0 \leq j \leq (\alpha - 1)$. Setting $g_0 = {g_i}^{p^{\alpha-1}}$ and $h_0 = {h_i}^{p^{\alpha-1}}$, the discrete logarithm of h_0 with base g_0 in the subgroup of order p is d_0.

Using the process of induction for known values of $d_0, d_1, \ldots, d_{i-1}$ we have the equality,

$$h_i {g_i}^{-(d_0 + d_1 p + \ldots + d_{i-1} p^{i-1})} = {g_i}^{p^i (d_i + d_{i+1} p + \ldots + d_{\alpha-1} p^{\alpha-1-i})}. \tag{2.3}$$

Both the elements on the left and right are elements of G. Multiplying both sides by $p^{\alpha-i-1}$, the right side is a power of g having one of the factors of the exponents as d_i. The occurrence of $d_{i+1}, \ldots, d_{\alpha-1}$ vanish simply as the order of g_i is p^{α}. After substituting the known values d_i may be obtained as the discrete logarithm in a subgroup of order p. Any generic algorithm can be utilised to compute it.

The above process may be done for each of the primes in the factorization after which an application of the Chinese Remainder Theorem gives the required logarithm for the group. Thus, this method is a divide and conquer strategy which first computes the smaller discrete logarithms in the subgroups of order ${p_i}^{\alpha_i}$. The run-time of the algorithm turns out to be $O(\sum_{i=1}^{k} \alpha_i (\log \#G + \sqrt{p_i}))$ which makes a worst-case running time of $O(\sqrt{\#G})$. The average run-time is taken as $O(\sqrt{p_l})$ where p_l is the largest prime present in the factorization of $\#G$.

It may be noted that the Pohlig-Hellman algorithm is effective if the group order is known beforehand and is smooth with respect to some integer, *i.e.*, each prime factor is small enough. In situations where any of these conditions fail, the application of the algorithm is not possible.

An example describing the algorithm is available in Section 13.2 of [63].

Next, the description of Pollard's rho algorithm is given. This algorithm is narrated in a detailed way stressing especially on tag tracing which is required later as a background of the improvement that we have suggested in Chapter 3.

2.4 Pollard's Rho Algorithm for Computing Logarithms

Pollard's rho algorithm for computing discrete logarithms is a probabilistic algorithm based on birthday paradox ([63], Section 14.1).
We assume that G is a cyclic group of prime order. Originally, Pollard [147] suggested this algorithm for application in $\mathbf{F}_p^*$, but the iteration function used by him can be used for any cyclic group. We assume that this group G can be partitioned into at most three equal sized subsets based on some easily identifiable compatible property. Let $G = G_0 \cup G_1 \cup G_2$ so that G_1 does not contain the identity element of the group G. The pseudo-random walk is defined as follows:

$$g_{i+1} = F(g_i) = \begin{cases} g \cdot g_i & \text{if } g_i \in G_0, \\ g_i^2 & \text{if } g_i \in G_1, \\ h \cdot g_i & \text{if } g_i \in G_2. \end{cases} \tag{2.4}$$

The element g_0 can be fixed as any arbitrary element of the group. In general, it is fixed as the identity element which is changed if some unsuccessful collision somehow occurs within the walk. The sequence in 2.4 may be defined in another fashion as $g_i = g^{a_i} h^{b_i}$ so that

$$a_{i+1} = \begin{cases} (a_i + 1) \bmod \#G & \text{if } g_i \in G_0, \\ 2a_i \bmod \#G & \text{if } g_i \in G_1, \\ a_i & \text{if } g_i \in G_2, \end{cases} \tag{2.5}$$

$$b_{i+1} = \begin{cases} b_i & \text{if } g_i \in G_0, \\ 2b_i \bmod \#G & \text{if } g_i \in G_1, \\ (b_i + 1) \bmod \#G & \text{if } g_i \in G_2. \end{cases} \tag{2.6}$$

The function F is said to be exponent traceable when the output after operating F can be expressed as powers of g and h if the input of F is so, *i.e.*, there exists easily computable functions F_g and F_h so that

$$F(g^a h^b) = g^{F_g(a,b)} h^{F_h(a,b)}. \tag{2.7}$$

It may be observed that for $g_i = g^a h^b \in G_0$, $F(g_i) = g^{a+1} h^b$. Then F turns out to be exponent traceable for G_0 with $F_g(a, b) = a + 1$ and $F_h(a, b) = b$. The same nature of exponent traceability

of F holds with just different functions F_g and F_h if any other subset of G is considered.
In such a pseudo-random walk, a collision is expected after roughly $O(\sqrt{\#G})$ steps according to the birthday paradox. The walk is composed of an initial tail and a cyclic part. This looks like the Greek letter rho which signifies the name of the algorithm. After a collision $g_i = g_j$, $j > i$ has been found,

$$g^{a_i} h^{b_i} = g^{a_j} h^{b_j}. \tag{2.8}$$

Taking logarithm with respect to g,

$$\log_g h = (a_i - a_j)(b_j - b_i)^{-1} \bmod (\#G). \tag{2.9}$$

This logarithm is computable only when the inverse exists, *i.e.*, $\gcd(b_j - b_i, \#G) = 1$. If this fails to hold, a fresh walk is started with some other starting point g_0.
It is here noteworthy that the recovery of logarithm from equation (2.8) is possible because the sequence g_i is a sequence in a smaller set compared to (a_i, b_i). Due to this difference in size of the domain set, collision in the former sequence in most cases is not a collision in the later. Various collision detection techniques, which will be discussed later, are used for detecting such a collision. The expected running time is about $O(\sqrt{\#G})$ which is same as the baby step giant step method. Finding collision in a naive way by using the birthday paradox would lead to storage complexity same as the BSGS algorithm. The cycle finding algorithms utilise the rho shape of the walk to find a collision using constant amount of storage. The memory requirement is thus $O(1)$.

2.5 Collision Detection Methods

Collision detection methods aim to look for a collision at the cost of some additional iterations. We discuss some collision detection methods.

Floyd's Algorithm: Knuth [110] stated the idea that for a sequence x_i, there exists i, such that $\mu \leq i \leq (\mu + \lambda)$ and $x_i = x_{2i}$ holds. The integers μ, λ are the pre-period and the period of the sequence x_i respectively. When the sequence is created by iterating a random function, the value of $(\mu + \lambda)$ is close to $\sqrt{\frac{\pi \#G}{2}}$ so $3\sqrt{\frac{\pi \#G}{2}}$ iterations are sufficient to get a collision. The storage required is negligible.

Brent's Method: Teske [171] applied this method to store the eight most recent g_k for which k is a power of three. Collision is searched among the i-th iteration element g_i and the stored elements. For a random iteration function it requires about $1.229\sqrt{\frac{\pi \#G}{2}}$ iterations.

Nivasch Stack-Based Cycle Detection: The central idea of this algorithm was suggested by Nivasch [137]. The algorithm starts with an empty stack. As the process continues, a stack of pairs is maintained, where each pair consist of the element at the i-th iteration g_i along with the integer i. An essential property of the sequence comsisting of pairs (g_i, i) is that both g_i and i are strictly increasing sequences. One starts with an empty stack and at each step i, all entries (g_j, j) such that $g_j > g_i$ are removed. If some element with value equal to that of g_i is found, then the process is stopped as a collision is found. Otherwise, g_i is pushed to the top of the stack and the process is continued. This algorithm has linear run-time with logarithmic memory. By using the technique of partitioning, the time required to find a collision can be lessened.

Distinguished Point Method: A distinguished point is a point in G that satisfies some easily testable property. One begins with an empty table of distinguished points. Distinguished points encountered in the random walk are added to the table. A collision is searched only among these points. As a drawback, some extra iterations, have to be performed after the initial collision. The number of extra iterations is equal to the inverse of the probability of a point being distinguished. To make extra iterations minimum, the distinguishing property is chosen so that sufficient elements satisfy such property. The table size also has to be easily manageable. The most common example of distinguishing property is some most or least significant bits being zero. This method can also be parallelized.

2.6 Parallelization of Pollard's Rho Algorithm

The Pollard's rho algorithm can be parallelized with each process performing the pseudo-random walk with the same algorithm. The initial points differ from one another. The entire system stops when a collision is encountered (between two different processes or the same process). In such a scenario where the processes do communicate with each other, the time complexity decreases by a factor of m, where m is the number of processes. However, there must exist a central server that stores all the computed elements leading to a storage complexity of $O(\sqrt{\frac{\pi \#G}{2}})$. This storage may be reduced at the cost of some extra iterations by using the idea of distinguished points [175]. This reduction of memory requirement can be huge as it is obtained by dividing the previous storage by the number of iterations needed to get a distinguished point.

2.7 Adding Walks and Tag Tracing for Pollard's Rho

We next discuss Pollard's rho with adding walks and tag tracing applied to prime order fields. Let the prime be p. Several variants of this algorithm have been studied. We briefly mention the variant introduced in [158].

Let r be a small positive integer. For $i = 1, \ldots, r$, random integers $\alpha_i, \beta_i \in \{0, \ldots, p-2\}$ are chosen such that both α_i and β_i are not zeros. Some integers m_i are defined as $m_i = g^{\alpha_i} h^{\beta_i}$, $i = 0, \ldots, r-1$. A pre-computed table T stores the entries $(i, m_i, (\alpha_i, \beta_i))$ for $i = 0, \ldots, r-1$. An indexing function s is defined as: $s : G \to \{0, \ldots, r-1\}$. Using s, a sequence of elements of G is defined as follows. Integers $a_0,\ b_0$ are randomly chosen and fixed so that, $a_0, b_0 \in \{0, \ldots, \#G-1\}$ and an initial point for the walk is set as $g_0 = g^{a_0} h^{b_0}$. For $j \geq 0$, the next point of the walk is defined from the previous point as $g_{j+1} = g_j m_{s(g_j)}$. The sequence $g_0, g_1, g_2, \ldots$ is heuristically assumed to be a pseudo-random walk on G for suitable value of r and proper choice of the function s.

Writing $g_j = g^{a_j} h^{b_j}$ for $j \geq 0$, we have $a_{j+1} = a_j + \alpha_{s(g_j)}$ and $b_{j+1} = b_j + \beta_{s(g_j)}$. So, it is easy to obtain a_{j+1} and b_{j+1} from a_j and b_j. Since G is finite, there must exist some j and k, with $j < k$ such that $g_j = g_k$, *i.e.*, the pseudo-random walk must lead to a collision. Denoting $\log_g h$ by $\mathfrak{d}$, the condition $g_j = g_k$ leads to the relation $a_j + \mathfrak{d} b_j = a_k + \mathfrak{d} b_k$. Under the condition that $b_j - b_k$ is invertible modulo $\#G$ (which holds with high probability for large p and appropriate group G), we have $\mathfrak{d} = (a_j - a_k)(b_k - b_j)^{-1} \bmod \#G$.

It may be noted that each iteration of the r-adding walk requires a negligible amount of information for processing. This paves the way for performing the group operations later in a single step, thus saving some per-step costs.

Tag Tracing: The concept of tag tracing is a time-memory trade-off introduced by Cheon et. al. [43]. In the pseudo-random walk defining Pollard's rho algorithm, the computation of g_{j+1} from g_j is done by multiplying g_j and $m_{s(g_j)}$. So, each step requires a multiplication of group elements. The tag tracing method was introduced in [43]. The essential idea is to increase the size of the pre-computed table so that a group multiplication is required after every ℓ steps for a suitable choice of the parameter ℓ. The computation done in the intermediate steps between two group multiplications is significantly faster than a group multiplication.

The set of multipliers $\mathcal{M} = \{m_i : m_i = g^{\alpha_i} h^{\beta_i}, i = 0, \ldots, r-1\}$ is defined as in the case of the original Pollard's rho algorithm. A parameter ℓ is chosen and $\mathcal{M}_\ell$ is denoted as the set of all possible products of at most ℓ elements from $\mathcal{M}$. The elements of $\mathcal{M}_\ell$ can be indexed by vectors of the form $(i_1, \ldots, i_k)$ where $i_1, \ldots, i_k \in \{0, \ldots, r-1\}$ and $0 \leq k \leq \ell$. Given $\mathbf{x} = (i_1, \ldots, i_k)$, the element of $\mathcal{M}$ indexed by $\mathbf{x}$ is $m_{\mathbf{x}} = m_{i_1} \cdots m_{i_k}$. Note that if $\mathbf{x}'$ is obtained by permuting the components of $\mathbf{x}$, then $m_{\mathbf{x}'} = m_{\mathbf{x}}$. So, we will assume that the vector $\mathbf{x}$ satisfies $i_1 \leq i_2 \leq \cdots \leq i_k$. A pre-computed table Tab is created. The rows of Tab are as follows.

$$(\mathbf{x}, m_{\mathbf{x}}, (a.b), (\hat{m}_0, \ldots, \hat{m}_{d-1}))$$

where

- $\mathbf{x} = (i_1, \ldots, i_k)$, with $0 \leq k \leq \ell$, $i_1, \ldots, i_k \in \{0, \ldots, r-1\}$, $i_1 \leq i_2 \leq \cdots \leq i_k$

- $m_{\mathbf{x}} = m_{i_1} \cdots m_{i_k} \bmod p$,
- (a, b) is such that $m_{\mathbf{x}} = g^a h^b$.

We explain the component $(\hat{m}_0, \ldots, \hat{m}_{d-1})$ later. The table Tab is stored as a hash table (or, some other suitable data structure), so that given an appropriate vector $\mathbf{x}$, it is easy to locate the corresponding row of Tab.

The indexing function $s : G \to \{0, \ldots, r-1\}$ defines the pseudo-random walk. Tag tracing requires an auxiliary indexing function $\overline{s} : G \times \mathcal{M}_\ell \to \{0, \ldots, r-1\} \cup \{\mathsf{fail}\}$, such that

$$\text{if } \overline{s}(y, m) \neq \mathsf{fail}, \text{ then } \overline{s}(y, m) = s(ym).$$

Suppose the element at the j-th step of the pseudo-random walk is g_j. The elements in the next ℓ steps are $g_{j+1}, \ldots, g_{j+\ell}$. For $1 \leq i < \ell$, recall that in the Pollard's rho algorithm $g_{j+i} = g_{j+i-1} m_{s(g_{j+i-1})}$. Iterating leads to the following.

$$\begin{aligned} g_{j+i} &= g_{j+i-1} m_{s(g_{j+i-1})} \\ &= g_{j+i-2} m_{s(g_{j+i-2})} m_{s(g_{j+i-1})} \\ &= \cdots \\ &= g_j m_{s(g_j)} m_{s(g_{j+1})} \cdots m_{s(g_{j+i-1})}. \end{aligned}$$

The goal of the tag tracing method is to avoid computing the intermediate elements $g_{j+1}, \ldots, g_{j+\ell-1}$ and instead jump directly from g_j to $g_{j+\ell}$. This requires obtaining the element $m_{s(g_j)} m_{s(g_{j+1})} \cdots m_{s(g_{j+\ell-1})}$ and so in particular, the index values $s(g_j), s(g_{j+1}), \ldots, s(g_{j+\ell-1})$. Since g_j is available, $s(g_j)$ can be directly obtained. For $i > 1$, the value of $s(g_{j+i})$ is obtained using the auxiliary tag function $\overline{s}$ as

$$\begin{aligned} s(g_{j+i}) &= s(g_j m_{s(g_j)} m_{s(g_{j+1})} \cdots m_{s(g_{j+i-1})}) \\ &= \overline{s}(g_j, m_{s(g_j)} m_{s(g_{j+1})} \cdots m_{s(g_{j+i-1})}). \end{aligned}$$

The elements $m_{s(g_j)} m_{s(g_{j+1})} \cdots m_{s(g_{j+i-1})}$ for $i = 0, \ldots, \ell - 1$ are elements of $\mathcal{M}_\ell$ and are part of the pre-computed table.

In the tag tracing method, a tag set $\mathcal{T}$ is identified. The index function s is defined as the composition of a tag function $\tau : G \to \mathcal{T}$ and a projection function $\sigma : \mathcal{T} \to \{0, \ldots, r-1\}$, *i.e.*, for $y \in G$, $s(y) = \sigma(\tau(y))$. Similarly, the auxiliary index function $\overline{s}$ is defined as the composition of an auxiliary tag function $\overline{\tau} : G \times \mathcal{M}_\ell \to \mathcal{T}$ and a projection function $\overline{\sigma} : \mathcal{T} \to \{0, \ldots, r-1\} \cup \{\mathsf{fail}\}$, *i.e.*, for $y \in G$ and $m \in \mathcal{M}_\ell$, $\overline{s}(y, m) = \overline{\sigma}(\overline{\tau}(y, m))$.

The definitions of $\tau, \sigma, \overline{\tau}$ and $\overline{\sigma}$ depend on a number of parameters. The two basic parameters are the prime p and the size of the index set r. The tag set is $\mathcal{T} = \{0, \ldots, t-1\}$ which also defines the parameter t. The parameter u is taken to be a suitable word size and d is defined to be

$d = \lceil \log_u(p-1) \rceil$. An integer $\bar{t}$ is chosen such that $\bar{t} > d(u-1)$ and $t\bar{t} < p^{1/3}$. The parameter w is defined to be $w = t\bar{t}$. Finally, the parameter $\overline{r}$ is defined so that $r\overline{r} = t$. As shown in [43], it is possible to choose all the parameters (other than p) to be a power of 2. Based on these parameters, the functions τ and σ are defined as follows.

$$\tau(y) = \left\lfloor \frac{y \bmod p}{\bar{t}\overline{w}} \right\rfloor; \qquad \sigma(x) = \lfloor x/\overline{r} \rfloor. \tag{2.10}$$

where $\overline{w} = \lceil \frac{p}{w} \rceil$. To define the function $\overline{\tau}$, elements of $y \in \mathbb{F}_p^\star$ are represented in base u as $y \bmod p = y_0 + y_1 u + \cdots + y_{d-1}u^{d-1}$. Given $m \in \mathcal{M}_\ell$, for $i = 0, \ldots, d-1$, define $\hat{m}_i = \lfloor (u^i m \bmod p)/\overline{w} \rfloor$. Since u is fixed, for each $m \in \mathcal{M}_\ell$, the values $\hat{m}_0, \ldots, \hat{m}_{d-1}$ are pre-computed and stored in the table Tab along with m (as mentioned earlier).

Given $y \in G$ and $m \in \mathcal{M}_\ell$, the value of $\overline{\tau}(y, m)$ is defined to be the following.

$$\overline{\tau}(y,m) \quad = \quad \left\lfloor \frac{\left(\sum_{i=0}^{d-1} y_i \hat{m}_i\right) \bmod w}{\bar{t}} \right\rfloor.$$

Given $x \in \mathcal{T}$, the function $\overline{\sigma}$ is defined as follows.

$$\overline{\sigma}(x) \quad = \quad \begin{cases} \text{fail} & \text{if } x \equiv -1 \bmod \overline{r}, \\ \lfloor x/\overline{r} \rfloor & \text{otherwise.} \end{cases}$$

The proof of correctness of the tag tracing procedure based on the above definitions of s and $\overline{s}$ is complex. We refer to [43] for details. The use of tag tracing for Pollard's rho requires a suitable definition of distinguished point. Again, we refer to [43] for details.

The computation of $\overline{s}$ has a chance of failure. In case of failure, a field multiplication is required. Otherwise, a field multiplication is required after every ℓ steps. The computation of $\overline{s}$ requires the computations of $\overline{\tau}$ and $\overline{\sigma}$. The quantities $\hat{m}_0, \ldots, \hat{m}_{d-1}$ are part of the pre-computed table. So, for the computation of $\overline{\tau}$, the d multiplications $y_i\hat{m}_i$, $i = 0, \ldots, d-1$ are required. Apart from these, all other computations are divisions by w, $\bar{t}$ and $\overline{r}$. Since these are chosen to be powers of 2, such computations are very fast. Overall, the computation of $\overline{s}$ is significantly faster than a field multiplication.

Our description of tag tracing has been in the context of DLP computation in a multiplicative subgroup of $\mathbb{F}_p^\star$ as given in [43]. The method can be applied to any finite cyclic group for which suitable tag and projection functions can be defined [43].

The net effect of tag tracing along with r adding walk and the idea of distinguished points lead to much faster discrete log computations. It has been reported by Cheon et. al. [43] that implementations in specific fields lead to at least 10 times faster results than the previous algorithms. This improvement is attributed to the fact that computing a smaller number of bits of a

random element and a preset element of the group is time-saving when compared to computing a full product. Computation of each iteration quicker with the aid of precomputed tables outgrows the disadvantage of a slight increase in the number of iterations. Tag Tracing applied to elliptic curves also leads to a small speed-up [31].

2.8 Pollard's Kangaroo Algorithm

This is another generic algorithm discovered by Pollard [147]. It is called the kangaroo or lambda method. It is especially useful when it is known that the target logarithm lies in some smaller interval $[a, b]$ with $(b - a) < \#G$.

In the Rho method, a single random walk is used in the shape of the Greek letter ρ while the Lambda method uses two walks in the shape of the Greek letter λ. It is described by employing two different random walks performed by two kangaroos. Kangaroo in this context means a sequence of elements in the group whose logarithm increases by successive jumps.

Two kangaroos are defined, a tame one with starting point g^a and a wild one with starting point h. The logarithm of each of the elements of the tame kangaroo sequence with base g is always known which is not the case of wild kangaroos. A jump function associates to each element of G a positive nunber upper bounded by $\sqrt{b-a}$.

The classical method is to partition G into almost equal sized subsets $G_i, i = 0, 1, \ldots, k$ with $k = \lfloor \log_2 (\frac{b-a}{2}) \rfloor$ and use a jump function $j(x) = 2^i$ if $x \in G_i$. The iteration function is set as $F(x) = x \cdot g^{j(x)}$.

The tame kangaroo sequence T_i and the wild kangaroo sequence W_i is defined respectively by

$$\begin{cases} T_0 = g^a, \\ T_{i+1} = F(T_i), \end{cases}$$

$$\begin{cases} W_0 = h, \\ W_{i+1} = F(W_i). \end{cases}$$

The known logarithm of each T_i is $a + \sum_{t=0}^{i-1} j(T_t)$. This is noted at each step and whenever this value exceeds b, the algorithm is aborted with $T_n = g^{l_n}$. Once such a point T_n has been encountered, we begin the wild sequence and keep a track of the jumps taken. This sequence is halted when either T_n is found again or when the sum of jumps exceeds the length $(b-a)$ of the interval. We denote the former case by success and the later by failure. After successful completion with a match of T_n and $W_i = hg^{w_i}$ the logarithm can be obtained as $\log_g h = (l_n - w_i)$. In case of failure, a new wild kangaroo may be started with some point $W_0 = hg^{w_0}$ for some small integer w_0.

It may be noted that, unlike BSGS, storage of every element is not required. For comparison,

only the last element of the tame kangaroo is to be stored. More details regarding the walk are present in the original work [147] and other works [146, 170]. This algorithm has good success probability [130]. The expected runtime is $O(\sqrt{b-a})$ with constant memory requirement. It may be also used in the entire interval but in that case, though the asymptotic complexity is the same as the rho method, the rho method is better due to the associated constants.
Van Oorschot and Weiner [175] presented an improved version that uses distinguished points. As this Kangaroo method uses a special type of r-adding walk as well as distinguished points, so tag tracing can be applied.

It is seen that except for the baby-step-giant-step algorithm, all other generic algorithms presented here have linear memory requirement. The BSGS algorithm and its variants are deterministic while Pollard's Rho and its variants are probabilistic. All of these processes may be distributed to obtain gain in time by the number of processors. The baby step giant step, as well as Pollard's Rho algorithm have $O(\sqrt{\#G})$ time complexity. The exhaustive search has runtime $O(\#G)$, while Pollard Kangaroo and Pohlig-Hellman have time complexity $O(\sqrt{b-a})$ and $O(\sqrt{p_l})$ respectively where $[a, b]$ is the interval-length for kangaroo method and p_l is the largest prime in the factorization of the group order for Pohlig-Hellman.
It thus follows from the discussion in the previous paragraph that Pollard's Rho algorithm offers the best time and space complexity in the case of generic algorithms. This paves the path to the next chapter where we suggest improvements in the case of prime order fields having large characteristic.

Chapter 3

Combining Montgomery Multiplication with Tag Tracing for Pollard's Rho Algorithm in Prime Order Fields

3.1 Introduction

In this chapter, we show how to apply Montgomery multiplication to the tag tracing variant of the Pollard's rho algorithm applied to prime order fields. This combines the advantages of tag tracing with those of Montgomery multiplication. In particular, compared to the previous version of tag tracing, the use of Montgomery multiplication entirely eliminates costly modular reductions and replaces these with much more efficient divisions by a suitable power of two.
It has already been stated in chapter 2 that the best known generic algorithm for solving DLP is Pollard's rho algorithm [147]. The resources required by the algorithm is $\sqrt{\#G}$ time and negligible space. For the DLP on finite fields, faster algorithms, namely the function field sieve and the number field sieve, are known. There are situations, however, where Pollard's rho algorithm is used for solving some of the sub-problems that arise while solving the DLP over finite fields. We refer to [99, 151, 135] for examples of such usage.

Since its introduction, several variants of Pollard's rho algorithm have been proposed. In particular, the tag tracing variant, as mentioned in Section 2.7 of this thesis showed the possibility of obtaining practical speed-up of Pollard's rho algorithm for certain groups. Concrete speed-ups were demonstrated for prime order subgroups of multiplicative groups of finite fields. Two kinds of fields were considered in [43], namely, prime order fields and small characteristic, large extension degree fields. We focus on the application of tag tracing to prime order fields.

Let p be a prime, $\mathbb{F}_p$ be the finite field of p elements. The group G where DLP is considered is typically a prime order subgroup of $\mathbb{F}_p^\star$.

Pollard's rho algorithm performs a pseudo-random walk. For solving DLP in $\mathbb{F}_p$, each step of the walk requires performing a multiplication in $\mathbb{F}_p$. The improvement achieved by the tag tracing method is to ensure that a field multiplication is required after every ℓ steps for a suitable choice of the parameter ℓ. In the intermediate steps between two field multiplication steps, a special computation is performed by the tag tracing method. This computation is significantly faster than a field multiplication. So, tag tracing speeds up Pollard's rho algorithm by a factor of about ℓ.

A field multiplication in $\mathbb{F}_p$ consists of two phases. The first phase is an integer multiplication while the second phase is a reduction modulo p operation. For primes p not having a special structure, the reduction operation can require a substantial portion of the overall time for a field multiplication. The technique of Montgomery multiplication [131, 30] works with Montgomery representation of elements and replaces a field multiplication by a Montgomery multiplication. The advantage of Montgomery multiplication is that all divisions are by certain powers of two and so can be implemented using right shift operations. The expensive modulo p operation is no longer required.

In this chapter, we show how the Montgomery multiplication can be combined with the tag tracing method. The goal is to retain the advantages achieved by tag tracing and also simultaneously replace the field multiplications required after every ℓ steps by a Montgomery multiplication. All the time consuming modulo p operations are completely eliminated. Consequently, the Montgomery multiplication version of tag tracing achieves further speed-up compared to the usual tag tracing algorithm. The combination of Montgomery multiplication and tag tracing is achieved without any trade-offs. In particular, the storage space required remains the same in both cases.

3.2 Montgomery Multiplication

Let x and y be two elements of $\mathbb{F}_p$ and the requirement is to compute the product $xy \in \mathbb{F}_p$. Typically, this is a two-stage process, where in the first stage the integer multiplication of x and y is carried out and then the result is reduced modulo p. The reduction operation can take a substantial fraction of the total time to perform the field multiplication. This is especially true if p does not have a special form. Montgomery multiplication was introduced [131] to replace the costly reduction operation modulo p by much cheaper divisions by powers of two. Below we provide a brief description of Montgomery multiplication based on [30].

Following the notation used in the context of tag tracing, let u be a power of two representing a word size and d be such that the elements of $\mathbb{F}_p^\star$ have a d-digit representation to base u. Choose

$R = u^d$ such that $u^{d-1} \leq p < u^d$. Since p is odd and u is a power of two, there exists μ satisfying $\mu = -p^{-1} \bmod u$.

The core of Montgomery multiplication is a procedure called Montgomery reduction. Given an integer x having a d-digit representation to base u, Montgomery reduction computes $xR^{-1} \bmod p$. The Montgomery multiplication is a generalisation which given two integers x and y computes $xyR^{-1} \bmod p$. Suppose x and y satisfy $0 \leq x, y < R$ and x is written as $x = \sum_{i=0}^{d-1} x_i u^i$ with $0 \leq x_i < u$ for $i = 0, \ldots, d-1$. From [30], the Montgomery multiplication procedure is the following.

$z \leftarrow 0$
for $i = 0$ to $d - 1$ do
  $z \leftarrow z + x_i y$
  $q \leftarrow \mu z \bmod u$
  $z \leftarrow (z + pq)/u$
end for
if $z \geq p$ then $z \leftarrow z - p$
output z.

It can be shown that the output z satisfies $z \equiv xyR^{-1} \bmod p$. For a proof of this statement, we refer to [30]. The point to be noted here is that the only divisions in the above procedure are by u which is a power of two. So, these divisions are simply right shift operations and are very fast.

Given two field elements x and y, one way to multiply them is to first convert them to Montgomery representation by computing $\tilde{x} = xR \bmod p$ and $\tilde{y} = yR \bmod p$, then performing a Montgomery multiplication of $\tilde{x}$ and $\tilde{y}$ to obtain $\tilde{z} = \tilde{x}\tilde{y}R^{-1} = xyR \bmod p$ and then performing a Montgomery reduction (or, performing Montgomery multiplication of $\tilde{z}$ and 1) on $\tilde{z}$ to obtain $\tilde{z}R^{-1} \bmod p = xy \bmod p$. This procedure has the overhead of converting x and y to Montgomery representation and at the end applying a Montgomery reduction to $\tilde{z}$. So, for performing a single multiplication, this procedure is not very useful. Instead, Montgomery multiplication turns out to be effective when a sequence of multiplications can be done in the Montgomery representation.

3.3 Combining Montgomery Multiplication with Tag Tracing

Pollard's rho algorithm in G consists of a sequence of multiplications modulo p. So, it is an ideal application case for Montgomery multiplication. Let us first consider how this can be done.

As described earlier, the pseudo-random walk of the Pollard's rho algorithm starts with g_0 and continues by computing $g_1, g_2, \ldots$, where for $j \geq 0$, $g_{j+1} = g_j m_{s(g_j)}$. Recall that for each $i \in \{0, \ldots, r-1\}$, the values α_i and β_i are known such that $m_i = g^{\alpha_i} h^{\beta_i}$. As before, a pre-computed table T stores $(i, m_i, (\alpha_i, \beta_i))$ for $i = 0, \ldots, r-1$.

To perform Pollard's rho algorithm using Montgomery multiplication, the multipliers are converted to Montgomery representation. This requires a change in the pre-computed table T. Denote the modified table by modT. Then the rows of modT are $(i, \tilde{m}_i, (\alpha_i, \beta_i))$ for $i = 0, \ldots, r-1$, where $\tilde{m}_i = m_i R \bmod p$.

As in the Pollard's rho algorithm described above, randomly choose a_0 and b_0 and define $z_0 = g^{a_0} h^{b_0}$. Let $\tilde{z}_0 = z_0 R \bmod p$ be the Montgomery representation of z_0. For $j \geq 0$, we define $z_{j+1} = z_j m_{s(\tilde{z}_j)} \bmod p$. Note that in this case, the indexing function s is applied to $\tilde{z}_j$ instead of being applied to z_j. This is because the element computed at the $(j+1)$-th step of the walk is $\tilde{z}_{j+1}$. The quantity $\tilde{z}_{j+1}$ is computed by applying Montgomery multiplication to $\tilde{z}_j$ and $\tilde{m}_{s(\tilde{z}_j)}$, *i.e.*, $\tilde{z}_{j+1} = \tilde{z}_j \tilde{m}_{s(\tilde{z}_j)} R^{-1} \bmod p = z_j m_{s(\tilde{z}_j)} R \bmod p = z_{j+1} R \bmod p$.

With the above modification, all the multiplications required in the pseudo-random walk are Montgomery multiplications. So, at no stage the reduction operation modulo p is required.

The exponent information can be obtained from the walk. For $j \geq 0$, let $z_j = g^{a_j} h^{b_j}$. Note that a_0 and b_0 are known. Let $i = s(\tilde{z}_j)$. Then from the pre-computed table, it is possible to obtain (m_i, α_i, β_i). By definition, we have $z_{j+1} = z_j m_i$ and so, $a_{j+1} = a_j + \alpha_i$ and $b_{j+1} = b_j + \beta_i$.

Now, suppose there is a collision in the pseudo-random walk, i.e., there are j and k with $j < k$ such that $\tilde{z}_j = \tilde{z}_k$. Using the definition of $\tilde{z}_j$ and $\tilde{z}_k$, we have $z_j R = z_k R \bmod p$ implying $z_j = z_k \bmod p$ since R is co-prime to p. Using $z_j = z_k \bmod p$, we obtain $a_j + \mathfrak{d} b_j = a_k + \mathfrak{d} b_k$, where $\mathfrak{d} = \log_g h$. From this relation, it is possible to obtain $\mathfrak{d}$ as mentioned earlier.

The distinguished point method for detecting collisions can be applied to this modified pseudo-random walk by defining distinguished points based on $\tilde{z}_j$ for $j \geq 0$.

The above description shows that using Montgomery multiplication to define the pseudo-random walk for the Pollard's rho algorithm results in replacing all the relatively expensive modulo p operations with divisions by powers of two. We next consider, how the tag tracing method can be applied to this version of the Pollard's rho algorithm.

Let us first consider the difficulties in applying Montgomery multiplication to the setting of tag tracing. Suppose the pseudo-random walk is at an element $\tilde{z}_j$ for some $j \geq 0$. The goal of tag tracing is to perform a single field multiplication to move to the element $\tilde{z}_{j+\ell}$. For the intermediate points of the walk, the index values $s(\tilde{z}_j), s(\tilde{z}_{j+1}), \ldots, s(\tilde{z}_{j+\ell-1})$ are required.

The goal is to replace the usual field multiplication with a Montgomery multiplication. On the other hand, recall that the function s is obtained from the auxiliary function $\overline{s}$, such that for $y \in G$ and $x \in \mathcal{M}_\ell$, if $\overline{s}(y, m) \neq$ fail, then $s(ym) = \overline{s}(y, m)$. The product ym in the argument of s is the usual field multiplication. So, there are two apparently conflicting requirements. For the movement from $\tilde{z}_j$ to $\tilde{z}_{j+\ell}$, a Montgomery multiplication is to be applied, while the indexing function s is defined with respect to the usual field multiplication.

We show a simple resolution of this problem. The first thing to note is that the product in the argument of s is not actually performed. Instead, $s(ym)$ is computed as $\overline{s}(y, m)$. For

$1 \leq i \leq \ell$, we have

$$\begin{aligned}
s\left(\tilde{z}_{j+i}\right) &= s\left(\tilde{z}_{j+i-1}\tilde{m}_{s(\tilde{z}_{j+i-1})}R^{-1} \bmod p\right) \\
&= s\left(\tilde{z}_{j+i-1}m_{s(\tilde{z}_{j+i-1})}RR^{-1} \bmod p\right) \\
&= s\left(\tilde{z}_{j+i-1}m_{s(\tilde{z}_{j+i-1})} \bmod p\right) \\
&= s\left(\tilde{z}_{j+i-2}\tilde{m}_{s(\tilde{z}_{j+i-2})}R^{-1}m_{s(\tilde{z}_{j+i-1})} \bmod p\right) \\
&= s\left(\tilde{z}_{j+i-2}m_{s(\tilde{z}_{j+i-2})}m_{s(\tilde{z}_{j+i-1})} \bmod p\right) \\
&= \cdots \\
&= s\left(\tilde{z}_j m_{s(\tilde{z}_j)} \cdots m_{s(\tilde{z}_{j+i-2})}m_{s(\tilde{z}_{j+i-1})} \bmod p\right) \\
&= \overline{s}\left(\tilde{z}_j, m_{s(\tilde{z}_j)} \cdots m_{s(\tilde{z}_{j+i-2})}m_{s(\tilde{z}_{j+i-1})} \bmod p\right).
\end{aligned}$$

Let $m = m_{s(\tilde{z}_j)} \cdots m_{s(\tilde{z}_{j+i-2})}m_{s(\tilde{z}_{j+i-1})} \bmod p$. The element m is in the set $\mathcal{M}_\ell$. For computing $\overline{\tau}$, the quantities $\hat{m}_0, \ldots, \hat{m}_{d-1}$ derived from m are required, but, the actual value of m is not required. The fourth component of the pre-computed table Tab corresponding to the entry for m has the values $\hat{m}_0, \ldots, \hat{m}_{d-1}$. So, using the entries in Tab, it is possible to compute $\overline{\tau}(\tilde{z}_j, m)$ and hence $\overline{s}(\tilde{z}_j, m)$ which provides the value for $s\left(\tilde{z}_{j+i}\right)$.

Now let us consider the computation of $\tilde{z}_{j+\ell}$ from $\tilde{z}_j$.

$$\begin{aligned}
\tilde{z}_{j+\ell} &= \tilde{z}_{j+\ell-1}\tilde{m}_{s(\tilde{z}_{j+\ell-1})}R^{-1} \bmod p \\
&= \cdots \\
&= \tilde{z}_j m_{s(\tilde{z}_j)} \cdots m_{s(\tilde{z}_{j+\ell-2})}m_{s(\tilde{z}_{j+\ell-1})} \bmod p \\
&= \tilde{z}_j m_{s(\tilde{z}_j)} \cdots m_{s(\tilde{z}_{j+\ell-2})}m_{s(\tilde{z}_{j+\ell-1})}RR^{-1} \bmod p \\
&= \tilde{z}_j \mathfrak{m}RR^{-1} \bmod p \\
&= \tilde{z}_j \tilde{\mathfrak{m}}R^{-1} \bmod p
\end{aligned}$$

where $\mathfrak{m} = m_{s(\tilde{z}_j)} \cdots m_{s(\tilde{z}_{j+\ell-2})}m_{s(\tilde{z}_{j+\ell-1})}$. So, $\tilde{z}_{j+\ell}$ is obtained by applying Montgomery multiplication to $\tilde{z}_j$ and $\tilde{\mathfrak{m}}$. The element $\mathfrak{m}$ is in the set $\mathcal{M}_\ell$ and so is in the pre-computed table Tab. Note however, the value of $\tilde{\mathfrak{m}}$ is required which is not present in Tab. One may, of course, obtain $\tilde{\mathfrak{m}}$ from $\mathfrak{m}$ by performing the product $\mathfrak{m}R \bmod p$. This would be costly and would defeat the whole purpose of utilising Montgomery multiplication. So, a better option would be to include the element $\tilde{\mathfrak{m}}$ in the table Tab as part of the entry corresponding to the row for $\mathfrak{m}$. This would increase the size of the table Tab. Instead, we propose that in the table Tab, the entry $\tilde{\mathfrak{m}}$ is to be stored in place of $\mathfrak{m}$.

Let us denote the modified table by modTab. Based on the above discussion, the rows of the table modTab are as follows.

$$(\mathbf{x}, \tilde{m}_{\mathbf{x}}, (a.b), (\hat{m}_0, \ldots, \hat{m}_{d-1}))$$

where

- $\mathbf{x} = (i_1, \ldots, i_k)$, with $0 \leq k \leq \ell$, $i_1, \ldots, i_k \in \{0, \ldots, r-1\}$,
- $m_{\mathbf{x}} = m_{i_1} \cdots m_{i_k} \bmod p$ and $\tilde{m}_{\mathbf{x}} = m_{\mathbf{x}} R \bmod p$,
- (a, b) is such that $m = g^a h^b$,
- $\hat{m}_i = \lfloor (u^i m \bmod p)/\overline{w} \rfloor$ for $i = 0, \ldots, d-1$.

where $\overline{w}$ is defined in equation 2.10 of Section 2.7 of this thesis. So, modTab stores $\tilde{m}$ instead of m while the quantities $\hat{m}_0, \ldots, \hat{m}_{d-1}$ in modTab are derived from m and not from $\tilde{m}$. In particular, the only difference between Tab and modTab is that Tab stores m whereas modTab stores $\tilde{m}$. All other entries of Tab and modTab are identical. So, the storage requirements of both Tab and modTab are also the same.

Using modTab, tag tracing can proceed as follows. For the jump from $\tilde{z}_j$ to $\tilde{z}_{j+\ell}$, the entry $\tilde{m}$ is to be used, whereas for the computations of the outputs of the function s, the entries $\hat{m}_0, \ldots, \hat{m}_{d-1}$ are to be used. Consequently, the advantage of tag tracing is retained, *i.e.*, all computations required for computing the output of s are divisions by powers of two. Additionally, there is an efficiency gain where the field multiplication required in tag tracing for the jump from the j-th step of the walk to the $(j+\ell)$-th step of the walk is replaced by a Montgomery multiplication. As explained earlier, this replaces the costly reduction operations modulo p by inexpensive divisions by powers of two.

3.4 Conclusion

In this chapter, we have shown how to incorporate Montgomery multiplication to the tag tracing variant of Pollard's rho algorithm for solving DLP in $\mathbb{F}_p$. This results in replacing costly modulo p operations with divisions by a power of two. It leads to practical speed-ups in actual implementations. A very rough estimate of the speed-up is the acceleration that happens when schoolbook multiplication is replaced by Montgomery multiplication in the prime field.

Chapter 4

Index Calculus Algorithms for Finite Field Discrete Log Problem

The index calculus technique is presently the best method to compute discrete logarithms on finite fields. A suitable representation of the group elements which are *smooth* in certain sense is exploited in such class of algorithms. The focus in this chapter will be on the application of *index calculus* strategies in the context of DLP in finite fields. The best index calculus strategy to compute discrete logarithm in a given finite field is chosen depending upon the size of the characteristic and extension degree. This method uses linear algebra to compute the discrete logarithms of some *small elements*. The details of the steps explicitly are discussed later. We shall shortly discuss the complexity of the algorithms proposed. *Complexity* means the heuristic running time in the case when runtime complexity is discussed. It means storage complexity when the theme of discussion is memory issues.

The core idea of the algorithm appeared in 1922 in the work of Maurice Kraitchik [114]. Andrew Odlyzko [138] first assigned the name *index calculus* algorithm. Adleman [4] discussed the runtime in the case of prime fields. Hellman and Reyneri [88] explored the idea in case of small characteristic and large extension degree fields. The runtime is expressed in terms of the L-notation. In both the algorithms of Adleman [4] as well as Hellman and Reyneri [88] the core idea has same intuition as the Morrison-Brillhart factorization method. The intuition comes from the probability of a natural number to be factored in terms of small primes. In the same way when the elements of the finite field are conceived as polynomials of some fixed degree, the probability of a random polynomial to be smooth with respect to some fixed degree bound may be considered. Adleman and Western-Miller [4, 176] first proposed algorithms for discrete log computations in prime fields. The complexity in these algorithms was $L(\frac{1}{2})$. Coppersmith [49] improved the complexity to $L(\frac{1}{3})$ in binary fields of the same size. Sometime later, the number field sieve (NFS) which was originally devised for the integer factoring problem [120] was adapted to DLP for prime fields [154, 76]. The complexity in this case was again $L(\frac{1}{3})$. The function field sieve [5, 8, 101,

100] (FFS) on the other hand, later came into the scenario and targeted small characteristic fields. NFS was generalised in [105] where the complexity of $L(\frac{1}{3})$ held for all classes of finite fields. There has been some noteworthy improvements in both NFS and FFS and other algorithms in small characteristic cases.

The extra information of the group structure is harnessed in case of index calculus algorithms to obtain faster subexponential algorithms compared to exponential generic ones. The best complexity is obtained by balancing the parameters. The details of the algorithm will depend on the concrete representation of the elements of the group.

4.1 Classification of Finite Fields and Broad Overview of Suitable Algorithms

The present algorithms for discrete logarithm can be classified on the basis of the characteristic and extension degree of the target field. Given a finite field $\mathbb{F}_{p^n}$, the characteristic p is written as $p = L_Q(a_p, c_p)$ for some suitable constants a_p, c_p and cardinality Q of the group. Based on the value of the first parameter corresponding to the characteristic, the finite fields are classified into three groups [18] as given below. Henceforth, by *small*, *medium* and *large* characteristic the following meanings will be used.

- Finite fields with *small* characteristic when $a_p < \frac{1}{3}$.
- Finite fields with *medium* characteristic when $\frac{1}{3} < a_p < \frac{2}{3}$.
- Finite fields with *large* characteristic when $\frac{2}{3} < a_p$.
- The boundary cases are at $a_p = \frac{1}{3}, \frac{2}{3}$ in which crossover between algorithms happen.

Till 2013, the following results were obtained. Function field sieve applied to small characteristic finite fields resulted in a complexity of $L_Q(\frac{1}{3}, (\frac{32}{9})^{\frac{1}{3}})$ [8]. In case of medium characteristic finite fields, applying number field sieve-high degree variant a complexity of $L_Q(\frac{1}{3}, (\frac{128}{9})^{\frac{1}{3}})$ [105] was obtained. The complexity was $L_Q(\frac{1}{3}, (\frac{64}{9})^{\frac{1}{3}})$ [105, 154, 125] for large characteristic finite fields when number field sieve algorithm was used. Joux [94] lowered the complexity to $L_Q(\frac{1}{4})$ for small characteristic fields. Later in 2014, Barbulescu, Gaudry, Joux, and Thomé [16] suggested algorithms for fields of lower characteristic. For small characteristic fields with characteristic $p = L_Q(a_p, c_p)$, their complexity $L_Q(a_p)$ which is slower than quasipolynomial unless $a_p = o(1)$. Granger, Kleinjung, and Zumbrägel [78] proposed quasi-polynomial algorithms (QPA) to attack DLP in binary fields. Thus given the present state-of-the-art algorithms, function field sieve and quasi-polynomial algorithms are preferred for small characteristic fields while number field sieve offers better complexity when the field has large characteristic, which includes prime fields as well.

It may be applied in case of medium characteristic too. The boundary cases are a bit more complex in the sense that many algorithms are available in those and pinpointing to a particular one does not seem easy. The quasi-polynomial algorithm has been obtained for small characteristic while the function field sieve is still the best for a corner case of medium characteristic. A rough estimate of the complexities are $L_Q(\frac{1}{3})$ for number field sieve and function field sieve, and $\log(Q)^{o(\log(\log(Q)))}$ for quasi-polynomial algorithms.

4.2 General Description of Index Calculus Algorithms

The index calculus algorithms consists of three main steps along with two other auxiliary phases, namely the preparatory phase and the final phase. In the preparatory step, a convenient representation of the field is chosen. In the final phase, after the target element is obtained completely in terms of factor base elements, the substitution of the logarithms of the factor base elements provides the logarithm of the target element. The steps are detailed as follows:

4.2.1 Precomputation of the Factor Basis

A set of elements which are *small* in certain sense is chosen as the factor basis. To define this, a pre-defined integral bound B is fixed beforehand. For example, when G is represented as a set of elements modulo some prime, p, the factor basis may consist of all prime numbers smaller than the bound B. In those cases, where the group G is represented as a set of polynomials modulo some irreducible polynomial, the factor basis may be selected as the set of all irreducible polynomials having degree smaller than B. At present, let the factor basis $\mathcal{F}$ be taken as $\mathcal{F} = \{p_1, p_2, \ldots, p_k\}$ where each of element $p_i, 1 \leq i \leq t$ is *smaller* than B in some sense depending on the scenario. The cardinality of the factor base, *i.e.*, $\#\mathcal{F} = \frac{B}{\log B}$. This follows from the Prime number theorem when G is represented as the set of elements modulo some integer and from Landau's Prime Ideal Theorem [116] in the case of a number field. It is stated in Theorem 7.16 of this thesis.

4.2.2 Relation Collection Phase

This phase aims to find *multiplicative relations* among elements of $\mathcal{F}$. Logarithm with base g taken on both sides of such a multiplicative relation results in a linear equation between the logarithms $\log_g p_i, 1 \leq i \leq k$. This linear system is formed modulo $(p-1)$. This phase uses trial division, sieving or pinpointing. The target should be to collect slightly more than k equations so that k independent equations are obtained.

4.2.3 Linear Algebra

The linear system of equations is solved modulo $(p-1)$ to get the discrete logarithm of the factor base elements. It may be observed that the system of equations that are produced is quite sparse.

Hence, special algorithms to be discussed later on, are used to solve these set of equations which are much faster than the general algorithms. This is detailed in Section 4.3.1.

4.2.4 Individual Logarithm Phase

This phase computes the logarithm of a random element of the group. This may be done as a two-step process. Let us call the target arbitrarily chosen element as z. Initially, z is decomposed as a product of other elements which are smaller than z in some sense depending upon the situation. This process is continued until z is obtained as a product of elements belonging to the factor basis. The logarithm of z is obtained by plugging the values of the discrete logarithms obtained in the earlier linear algebra phase.

4.3 Some Details Regarding the Phases of Index Calculus Algorithms

4.3.1 Linear Algebra over Finite Fields

After completion of the relation collection step a $k^{'} \times k$ matrix M is obtained where $k^{'}$ denotes the total number of equations collected which is slightly bigger than k. The linear algebra step aims to find a vector in the kernel of M. It is solved modulo $(p-1)$. Until 1980, this step was a major bottleneck.
Initially, the only algorithm to tackle this phase was the Gaussian elimination method which has cubic complexity in the number of rows (or columns) of the matrix.
When the Gaussian elimination method is applied to a sparse system of linear equations, each pivoting increases the number of non-zero elements per row. So, after a few iterations, the matrix loses its property of sparseness. For large dimensional matrix, Gaussian elimination quickly overflows the available memory. For small matrices, it may be sufficient enough for linear algebra. But entries of matrices in DL computation can belong to some large characteristic finite field compared to boolean matrices of integer factorization.

Some special families of algorithms are available which are much better than Gaussian elimination when dealing with sparse matrices. The first family, structured Gaussian elimination [138, 115], is composed of variants of Gaussian elimination so that fill-in is minimized during pivoting. The aim is to reduce the dimension of the matrix and at the same time, ensure that the matrix remains sparse. The general term for this minimization of fill-in is *filtering* and it was optimized recently [33].
To solve the reduced system the iterative methods like Lanczos and Wiedemann [177] and its generalization for parallel processing, Block Wiedemann [173] can be used. The core idea of both these methods is to compute *Krylov subspaces*, vectors $Vect\langle M^i \cdot b\rangle$ for the given matrix M and

some fixed vector b. The algorithms in this family find a solution of a linear system by computing the minimal polynomial of the considered matrix. For a square matrix of order N, with λ entries per row, the complexity is $O(\lambda N^2)$.
It has already been said that using sparse linear algebra techniques like Lanczos and Block Wiedemann instead of Gaussian elimination optimizes memory requirement. Furthermore, these techniques for the sparse case are also better in terms of time complexity. Roughly speaking, for a matrix of order N, Gaussian elimination takes $O(N^3)$ field operations while these methods take $O(N^2)$ field operations. The linear algebra for solving discrete log, may be needed for some finite field $\mathbb{F}_l$ for some large integer l. In such a case, this savings in time complexity would be huge practically.
It is noteworthy that Lanczos's method needs $O(k)$ divisions in the base field. The CADO-NFS package [169, 173] can be employed to compute the kernel of the matrix obtained after relation collection step when attempting record DL computations with several million rows and columns.

4.3.2 Smoothness

In index calculus algorithms it is heuristically assumed that the associated entities, which may be integers, ideals or polynomials depending on the context, behave like random entities of the same size. *Smooth* elements are those that split into elements of a much smaller set. Any relation generation technique that produces more of such smooth elements lead to an improvement of the index calculus technique in general. As the complexity of the relation collection phase plays a significant role in the overall complexity of the index calculus algorithm, it is important to keep note of the smoothness of the associated quantities.
The definitions and the theorems listed below try to provide an estimate of the probabilities of smoothness that are essential to analyse index calculus techniques.

Definition 4.1. A positive number is said to be *B-smooth* for some positive B if all its prime factors are at most B.

Definition 4.2. A polynomial over a finite field is said to be *b -smooth* for some positive integer b if all its irreducible factors have degree at most b.

Canfield, Erdös and Pomerance [41] expressed the asymptotic density of smooth numbers amongst the integers as follows.

Theorem 4.3. *Let $\psi(A, B)$ be the number of natural numbers smaller or equal to A that are B-smooth. If $A \geq 10$ and $B \geq \log A$, then*

$$\psi(A, B) = Au^{(-u+o(1))}$$

where, $u = \frac{\log A}{\log B}$ and the limit implicit in the $o(1)$ is for $A \to \infty$.

An analogous result in case of polynomials is presented by Odlyzko [138] and Lovorn [124].

Theorem 4.4. *A uniformly random polynomial $f \in \mathbb{F}_q[x]$ of degree m is b-smooth with probability*

$$u^{(-u+o(1))}$$

where $u = \frac{m}{b}$ provided that $m^{\frac{1}{100}} \leq b \leq m^{\frac{99}{100}}$.

The complexity of index calculus algorithms depends on the relation collection phase. In this step, there is a trade-off between the size of the factor base and the number of relations. This trade-off may be optimised by choosing proper bounds for the smoothness parameter for the elements under consideration. Corollary 4.5 of Theorem 4.3 is used in suitably choosing values of the associated quantities as follows.

Corollary 4.5. *Given an integer A which is at most $L_Q(\alpha_A, \sigma)$ and a smoothness bound $B = L_Q(\alpha_B, \beta)$ with $\alpha_B < \alpha_A$, the probability that A is B-smooth is*

$$\Pr[A \text{ is } B\text{-smooth}] = L_Q(\alpha_A - \alpha_B, -(\alpha_A - \alpha_B)\tfrac{\sigma}{\beta}).$$

The run-time of any index calculus algorithm is dependent on the smoothness of the associated elements. The complexity is expressed in terms of the subexponential notation. This is done after fixing optimal values of the elements.

To balance the parameters and ensure optimal cost, in case of $\mathbb{F}_p^\star$, the bound of the factor base is set in terms of subexponential notation as $B = L_p(\alpha_B, \beta)$. We already have $p = L_p(1, 1)$. In order to calculate the probability of smoothness of integers less than p, the values of α_A, σ are then taken as $\alpha_A = 1, \sigma = 1$. From corollary 4.5, the probablity of B-smoothness of any integer less than p is,

$$\Pr[(A \leq p) \text{ is } B\text{-smooth}] = L_p(1 - \alpha_B, -(1 - \alpha_B)\tfrac{1}{\beta}).$$

As we need about $\#\mathcal{F} = \frac{B}{\log B} \leq B$ relations so

$$\text{Number of tests} \approx \frac{B}{\Pr[(A \leq p) \text{ is } B\text{–smooth}]} = L_p(\alpha_B, \beta) L_p(1 - \alpha_B, \tfrac{(1-\alpha_B)}{\beta})$$

It may be seen that $\max(\alpha_B, 1 - \alpha_B)$ is minimal for $\alpha_B = \frac{1}{2}$. The relation collection cost is then $L_p(\frac{1}{2}, \beta + \frac{1}{2\beta})$.

The linear algebra phase takes time $B^2 = L(\frac{1}{2}, 2\beta)$. From the cost of these first two steps of relation collection and linear algebra, $\min(\beta + \frac{1}{2\beta})$ is at $\beta = \frac{1}{\sqrt{2}}$. The total cost of the first two steps is thus $L_p(\frac{1}{2}, \sqrt{2})$ from the optimality analysis. The individual logarithm step requires a single relation and so is of low complexity.

For finite fields in general, the running time of relation collection is also $L(\frac{1}{2}, \sqrt{2})$, which is based on the smoothness probabilities for the polynomials.

4.3.3 Example to Demonstrate Index Calculus Strategy

This example describes the steps of index calculus algorithm on a small field.
Let $p = 101$ and $g = 3$. The aim is to compute the discrete logarithm of $z = 53$ with respect to g. The factor base is taken as $\mathcal{F} = \{2, 3, 5, 7\}$.
Some random powers of g are taken to compute the relations among the logarithms of the factor basis elements. The following six relations modulo the prime p may be considered for use.

$$\begin{aligned} g^{19} &\equiv 2^2 \times 7 \pmod{p} \\ g^{30} &\equiv 2 \times 3 \pmod{p} \\ g^{45} &\equiv 2^5 \pmod{p} \\ g^{54} &\equiv 2^2 \times 5 \pmod{p} \\ g^{61} &\equiv 7 \pmod{p} \\ g^{86} &\equiv 2 \times 5 \times 7 \pmod{p}. \end{aligned}$$

This can be written in matrix notation as follows:

$$\underbrace{\begin{bmatrix} 2 & 0 & 0 & 1 \\ 1 & 1 & 0 & 0 \\ 5 & 0 & 0 & 0 \\ 2 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 1 & 0 & 1 & 1 \end{bmatrix}}_{} \times \begin{bmatrix} \log_g 2 \\ \log_g 3 \\ \log_g 5 \\ \log_g 7 \end{bmatrix} = \begin{bmatrix} 19 \\ 30 \\ 45 \\ 54 \\ 61 \\ 86 \end{bmatrix} \mod (p-1).$$

Using linear algebraic methods to solve this system leads to the following logarithm of the factor base elements.

$$\begin{bmatrix} \log_g 2 \\ \log_g 3 \\ \log_g 5 \\ \log_g 7 \end{bmatrix} = \begin{bmatrix} 29 \\ 1 \\ 96 \\ 61 \end{bmatrix} \mod (p-1).$$

To find the individual logarithm, it is observed that

$$zg^{53} \equiv 2 \times 5 \pmod{p}.$$

This leads to

$$\log_g z \equiv \log_g 2 + \log_g 5 - 53 \pmod{(p-1)}.$$

Then the logarithm turns out to be

$$\log_{\mathfrak{g}} z = 72$$

in the chosen subgroup.

4.4 Function Field Sieve

It has already been discussed in Section 4.1 that the function field sieve (FFS) is suitable for solving the discrete logarithm problem in finite fields of medium and small characteristic. The name comes from the fact that the initial versions of the algorithm used multiplicative relations between ideals in two function fields.

4.4.1 Previous Works

The function field sieve was initially suggested by Adleman [5] in 1994 and later by Adleman and Huang [8]. This can also be viewed as a generalisation of Coppersmith's [49] algorithm. Later, Joux and Lercier [101], provided practical improvements to this algorithm. The asymptotic complexity is the same as that of Adleman-Huang FFS, but it is more practical than that. Further improvement in the case of medium-sized fields was suggested by Joux and Lercier [100]. This variant is quite simple in which use of function fields were removed. These are the algorithms which serve as a benchmark for comparing recent developments.

The FFS in cases of medium and small characteristic fields is described next.

4.4.2 The Function Field Sieve for Medium Characteristic Prime

Joux and Lercier [100] proposed the following variant of the function field sieve in 2006. It is simple in the sense that the variant no longer uses ideals. The intricacies of the function field sieve subtly remain embedded in the background. The variant is designed for the fields $\mathbb{F}_{p^n}$ having medium-small characteristic *i.e.*, where $p = L_{p^n}(\alpha)$ with $\alpha \leq \frac{1}{3}$. It may also be applied to fields $\mathbb{F}_{q^m}$ for some prime power q. This description of the algorithm is based on [100, 99, 151].
The main phases of the algorithm are relation collection, linear algebra and the individual discrete log phase. Two other auxiliary phases are the preparatory phase and the final phase.

Representation of $\mathbb{F}_{p^n}$: The first task is to choose a convenient representation for $\mathbb{F}_{p^n}$.

Choose n_1, n_2 and k to be positive integers less than n. This needs to be done after a careful inspection as these values play a role in deciding the complexity of the algorithm. Let $g_1(X) = X^{-n_1}$ and $g_2(X)$ be such that $X^k - g_2(g_1(X)) = f_1(X)/X^{n_1 n_2}$, where $f_1(X)$ is a

polynomial of degree $n_1n_2 + k$. The idea is to choose $g_2(X)$ to be a polynomial over $\mathbb{F}_p$ of degree n_2 such that $f_1(X)$ has an irreducible factor $f(X)$ of degree n. Since the degree of $f_1(X)$ is $n_1n_2 + k$, $n \leq n_1n_2 + k$. If $n = n_1n_2 + k$, then experiments show that it is possible to choose $g_2(X)$ such that $f_1(X)$ itself is irreducible. In this case, we take $f(X)$ to be equal to $f_1(X)$.

The field $\mathbb{F}_{p^n}$ is represented as $\mathbb{F}_p[x]/(f(x))$. Let $y = g_1(x) = x^{-n_1}$. Then $x^k - g_2(y) = f_1(x)/x^{n_1n_2}$. Since $f(x) \mid f_1(x)$, the relation $x^k = g_2(y)$ holds in $\mathbb{F}_{p^n}$. For $k = 1$, this method was described by Joux [99]. In the following, we will assume $k = 1$, $y = g_1(x) = x^{-n_1}$ and $x = g_2(y)$ since these are the choices we use in our discrete logarithm computation. These two relations play a central role in framing the algorithm. For other variants of choosing the field representation, we refer to [99, 151].

Note that representation of the finite field does not involve objects from function fields or number fields. Elements are polynomials of degree at most $(n-1)$ over the finite field $\mathbb{F}_p$. This makes it possible to implement the algorithm entirely using finite field concepts.

Choice of Generator: The non-zero elements of $\mathbb{F}_{p^n}$ form a cyclic group under multiplication. Discrete logarithms are computed with respect to some generator. The actual choice of a generator is not important. It usually turns out that one of the polynomials $x + a$ is a generator for some $a \in \mathbb{F}_p$. This is something practically convenient as then this element can be included in the factor base where the logarithm is known beforehand to be equal to one.

Factor Basis: The factor basis is $\mathbb{B} = \{(x + a_i), (y + b_j) : a_i, b_j \in \mathbb{F}_p\}$. There are $2p$ elements in $\mathbb{B}$.

The more general option is to consider the factor base to be consisting of all irreducible polynomials in x, y of degree at most D. But for medium-sized prime, if $D > 1$, the size of the factor base which is $2p^D$ becomes too large. This increases the complexity of the bottleneck linear algebra phase.

For certain extension fields, by using the action of Frobenius it is possible to reduce the size of the factor basis by a factor of n. Joux [99] showed this for Kummer extension, *i.e.*, for fields $\mathbb{F}_{p^n}$ such that $n \mid (p-1)$. An advantage of Kummer extension is that the size of the factor basis reduces by a factor of n, making it possible to perform discrete logarithm computations on larger fields. The condition $n \mid (p-1)$, however, is very special and is unlikely to hold for general p and n.

Modulus of Discrete Logarithms: The requirement is to compute the discrete logarithm modulo $p^n - 1$. For some generator α of $\mathbb{F}_{p^n}$, an element α^i is in $\mathbb{F}_p$ is equivalent to the condition that $\alpha^{ip} = \alpha^i$. This is again equivalent to $(p^n - 1) | i(p-1)$.

Let $M = \frac{p^n-1}{p-1}$. Then the necessary and sufficient condition for α^i to be in $\mathbb{F}_p$ is M divides i. In practice, $p^n - 1$ is factored and the FFS algorithm is used to compute the discrete logarithm modulo the large prime factors. The discrete logarithms of elements of $\mathbb{F}_p$ are equal to 0 modulo M and so in the computation of discrete logarithm modulo a large prime factor of M, one may ignore the discrete logarithms of the elements of $\mathbb{F}_p$.

4.4.2.1 Relation Collection

In this phase, relations among factor basis elements are to be gathered which will lead to a system of equations. For arbitrary elements $a, b, c \in \mathbb{F}_p$, consider the expression $(x + a)y + (bx + c) = xy + ay + bx + c$. Using $y = g_1(x) = x^{-n_1}$ and $x = g_2(y)$, this expression can be written in two different ways as follows.

$$\begin{aligned} xg_1(x) + ag_1(x) + bx + c &= (x + a + bx^{n_1+1} + cx^{n_1})/x^{n_1} = h_1(x)/x^{n_1}; & (4.1)\\ yg_2(y) + ay + bg_2(y) + c &= h_2(y). & (4.2) \end{aligned}$$

Note that $h_1(x)$ is a polynomial of degree $n_1 + 1$ and $h_2(y)$ is a polynomial of degree $n_2 + 1$. Over $\mathbb{F}_{p^n}$, we have $h_1(x)/x^{n_1} = h_2(y)$. Suppose that both $h_1(x)$ and $h_2(y)$ are smooth polynomials, *i.e.*, $h_1(x) = b\,\Pi_{\alpha_i}(x + \alpha_i)$ and $h_2(y) = d_1\Pi_{\beta_j}(y + \beta_j)$ for some $d_1 \in \mathbb{F}_p$. Then, over $\mathbb{F}_{p^n}$, we have the relation

$$h_1(x)/x^{n_1} = b\,\Pi_{\alpha_i}(x + \alpha_i)/x^{n_1} = d_1\Pi_{\beta_j}(y + \beta_j) = h_2(y). \quad (4.3)$$

This gives the following linear equation among the discrete logarithms of the elements of the factor basis.

$$-n_1 \log x + \sum_{\alpha_i} \log(x + \alpha_i) = \sum_{\beta_j} \log(y + \beta_j) \bmod M.$$

Each such linear equation involves $n_1 + n_2 + 1$ terms.

The factor basis contains $2p$ elements. To be able to solve the system of linear equations arising from linear equations of the above type, a little more than $2p$ relations are required. The free parameters are a, b and c giving rise to p^3 expressions of the type $xy + ay + bx + c$. Heuristically, we may assume that the p^3 expressions give rise to $p^3/((n_1 + 1)!(n_2 + 1)!)$ linear equations. This quantity come from the fact that heuristically for a degree m polynomial, the probability of it factoring into linear factors is $\frac{1}{m!}$ and the degrees of the polynomial h_1, h_2 are $(n_1 + 1)$ and $(n_2 + 1)$ respectively. This quantity reaches a maximum for almost equal n_1, n_2. So,

for the relation collection phase to succeed, the following condition has to hold.

$$\frac{p^3}{(n_1+1)!(n_2+1)!} \quad > \quad 2p. \tag{4.4}$$

Pinpointing: The idea of pinpointing was introduced by Joux [99] to speed up relation collection. Suppose that for some choice of a, b and c, the polynomial $h_1(x)$ turns out to be smooth, *i.e.*,

$$x + a + bx^{n_1+1} + cx^{n_1} \quad = \quad b \cdot \Pi_{\alpha_i}(x + \alpha_i). \tag{4.5}$$

Using the transformation $x \mapsto tx$, for $t \in \mathbb{F}_p \setminus \{0, 1\}$, the right side of (4.5) remains smooth, while the left hand side corresponds to the expression obtained from $a' = a$, $b' = bt^{n_1+1}$ and $c' = ct^{n_1}$. So, once a smooth $h_1(x)$ is obtained, by varying t over all elements of $\mathbb{F}_p$, it is possible to obtain $p - 2$ smooth $h_1(x)$'s without any further smoothness checking.

In [99], it was shown that the number of trials required for obtaining a single relation which has both sides smooth is

$$\frac{(n_1+1)!(n_2+1)!}{p-1} + \min\left((n_1+1)!, (n_2+1)!\right). \tag{4.6}$$

It was shown in [151] that the idea of smoothness checking can be implemented with a sieving procedure which entirely avoids smoothness checking.

4.4.2.2 Linear Algebra

The relation collection phase produces a little more than $2p$ linear equations involving the discrete logarithms of the elements of the factor basis. Each equation has $n_1 + n_2 + 1$ terms. Additionally, we include the linear equation $\log y = -n_1 \log x$ to account for the relation $y = x^{-n_1}$ between x and y.

The obtained system of linear equations is sparse. From the discussions in Section 4.3.1, Block Wiedemann algorithm is used for solving it.
The system of inhomogeneous linear equations is given by a matrix $\mathbf{M}$ and a coefficient vector $\mathbf{b}$. Before attempting to solve the system, a filtering step is applied. The goal of the filtering step is to reduce the size of the matrix and/or make it more sparse. Basic filtering can be applied to remove duplicate rows and empty columns of the matrix $\mathbf{M}$. Sophisticated filtering methods are available in the CADO-NFS software [169]. The completion of the linear algebra step provides discrete logarithms of some elements.

4.4.2.3 Individual Descent

Let $\Pi(x)$ be the target element whose discrete logarithm is to be computed. Typically, $\Pi(x)$ will be a polynomial of degree $n-1$. After the linear algebra step, assume that we have computed the discrete logarithms of all linear polynomials of the form $x+\alpha_i$ and $y+\beta_j$. So, the goal is to be able to express $\Pi(x)$ as a rational function where both the numerator and the denominator are products of linear polynomials. This procedure is called descent.

The entire descent is not done in a single step. The target polynomial is successively descended to lower degree polynomials until finally descent to linear polynomials become possible. For the initial descent, a simple randomisation strategy usually works. Suppose we wish to descend from an irreducible polynomial $\phi(x)$. Choose a random polynomial $D(x)$ whose factors are of lower degree than that of $\phi(x)$. Let $N(x) = \phi(x)D(x) \bmod f(x)$. If the factors of $N(x)$ are also of degrees lower than that of $\phi(x)$, then since $\phi(x) = N(x)/D(x)$, we have a descent from $\phi(x)$ to lower degree factors of $N(x)$ and $D(x)$. If these factors are not linear, then they would require to be further descended. More systematic techniques for descent are known. A method based on computing the kernel of a matrix has been described in [100, 96]. Another method has been used in [151].

The descent becomes more difficult as the degrees of the polynomials become close to one. The 2-1 descent from quadratic to linear polynomials is the most difficult one. A heuristic argument has been used to show that the probability of a successful 2-1 descent in a single trial is $1/((n_1-1)!(n_2+1)!)$ [99, 151]. The work [151] provides the probability of a successful d-$(d-1)$ descent (*i.e.*, descent from a degree d polynomial to polynomials of degrees at most $d-1$) for $d \geq 2$. In an asymptotic setting, the effect of d-$(d-1)$ descent on the overall time for solving discrete logarithm has been analysed in [151]. It has been shown that for $d > 2$, the asymptotic cost of d-$(d-1)$ descent is always lower than the asymptotic cost of relation collection. On the other hand, for $d = 2$, there are situations where the asymptotic cost of 2-1 descent is more than the asymptotic costs of the other two phases.

Following the methods of [100, 99, 151], using a single degree of freedom, the heuristic probability of success for a 2-1 descent is $1/((n_1-1)!(n_2+1)!)$. So, the number of trials required for a single 2-1 descent is about

$$(n_1-1)!(n_2+1)!. \tag{4.7}$$

With a single degree of freedom, the number of trials that can be made is p. Let

$$\Lambda \;=\; \frac{p}{(n_1-1)!(n_2+1)!}. \tag{4.8}$$

It has been suggested in [100, 99, 151] that for a 2-1 descent to be possible, $\Lambda \geq 1$ has to hold. Experiments show that while $\Lambda \geq 1$ makes the descent easy, it may be possible to perform a 2-1

descent even when $\Lambda < 1$. The parameter Λ does, however, play a role in determining the ease of descent. The higher the value of Λ, the easier is a 2-1 descent, while for lower values of Λ, a direct 2-1 descent may not be possible and one would have to use walk and/or branching techniques.

4.4.2.4 Final Discrete Logarithm Computation

The linear algebra step provides the discrete logarithm of the elements of the factor basis elements modulo the large prime divisors of $p^n - 1$. So, once the descent step is completed, it is possible to compute the discrete logarithm of the target element modulo the large prime divisors of $p^n - 1$. The discrete logarithm of the target element modulo the smaller factors of $p^n - 1$ are computed using the Pollard rho and the Pohlig-Hellman algorithms. Finally, all the discrete logarithms are combined using the Chinese remainder theorem to obtain the discrete logarithm of the target element modulo $p^n - 1$.

4.5 Algorithms for Small Characteristic Fields

The function field sieve along with Joux's algorithm [94] and the quasi polynomial algorithms provide optimized methods to solve discrete log problem in case of small characteristic finite fields.

The three major steps of function field sieve have been already discussed as relation collection, linear algebra and the individual logarithm step. The aim of the last individual log step is to express the logarithm of the target element as a linear combination of logs of elements which are present in the factor base or the logarithms are known. It can be further subdivided as initial splitting and descent step. The goal of the initial splitting step is to obtain the target logarithm as linear combination of logarithms of elements which are smooth for some suitable smoothness parameter B already chosen before. The descent step then obtains the expression of each of the irreducible factors of the B-smooth elements in terms of elements of the factor basis.

The relation collection and the linear algebra steps are the dominant steps when seen in the light of complexity. Attempts to make them less costlier result in a smaller factor base. This does not reduce the difficulty of the last individual logarithm step. In fact, reduction of the factor base increases the cost of individual logarithm step even more.

Speeding up the individual logarithm step may be done by reducing the cost of initial splitting. Subfields of the finite field can be utilised to split the target into smaller degree or smaller coefficient elements.

The standard algorithm for initial splitting in fields of composite extension degree is Waterloo algorithm [24, 25].

4.5.1 The Waterloo Algorithm

The basic idea behind the Waterloo algorithm is that the probability of smoothness increases as the degree decreases. A formal description is as follows.
Let $\mathbb{F}_{p^n}$ be the finite field of p^n elements where p is a small prime and $n = n_1 n_2 > 1$ is a composite integer. For practical scenarios, $n_1 \ll n_2$. The Waterloo algorithm iteratively generates a pair of polynomials of degrees at most $n_2/2$ and tests both of them for B-smoothness for an appropriate choice of B. The cost of generating the pair of polynomials is $O(n_2^2)$ multiplications over $\mathbb{F}_{p^{n_1}}$. Considering a multiplication in $\mathbb{F}_{p^{n_1}}$ to require $O(n_1^2)$ multiplications over $\mathbb{F}_p$, the cost of generating the pair of polynomials is $O(n^2)$ multiplications over $\mathbb{F}_p$.
This initial splitting idea was used in the individual logarithm step. Guillevic [84] proposed a different algorithm for initial splitting over such fields. It will be discussed in detail in Chapter 6.

4.5.2 Works in Small Characteristic

In small characteristic fields, several attempts have been made after Coppersmith [49]. It was followed by the works [77, 174, 101, 98, 87, 73, 97, 74, 52, 94, 75, 2, 93, 161, 16, 78, 81, 17, 1, 78, 79, 102, 109, 103, 79, 80, 82]. These include discrete log computations both in characteristic two as well as characteristic three fields. Most of these have been taken from the number theory mailing list (https://listserv.nodak.edu/cgi-bin/wa.exe?A0=NMBRTHRY).

There has been quite some improvements in small characteristic fields, specially from 2012. Due to these, fields of characteristic two, three, composite extension degree which were target groups of pairing-friendly hyper-elliptic curves are prone to attacks.
Göloğlu, Granger, McGuire, and Zumbrägel [74] showed that relation collection was achievable in polynomial time. Joux[94] reported a low complexity $L(\frac{1}{4} + o(1))$. Granger, Kleinjung, and Zumbrägel [81] suggested an alternative descent method. More recently, in July 2019, a new record discrete logarithm computation in the field $\mathbb{F}_{2^{30750}}$ was published.
These works mostly differ in the individual logarithm phase.
Some works in characteristic three fields which include record discrete log computations as well as concrete analysis for certain fields are [102, 2, 1, 3]. An improved descent was proposed in 2013 [62] which was a QPA. Two variants [16, 78] of this algorithm were further published.

4.6 Number Field Sieve

Number Field Sieve (NFS) is an index calculus algorithm with three main phases relation collection, linear algebra and descent. Prior to these, is the set-up phase. There is a common structure at the core of FFS, NFS and its various variants. Figure 4.1 describes the structure. The ring

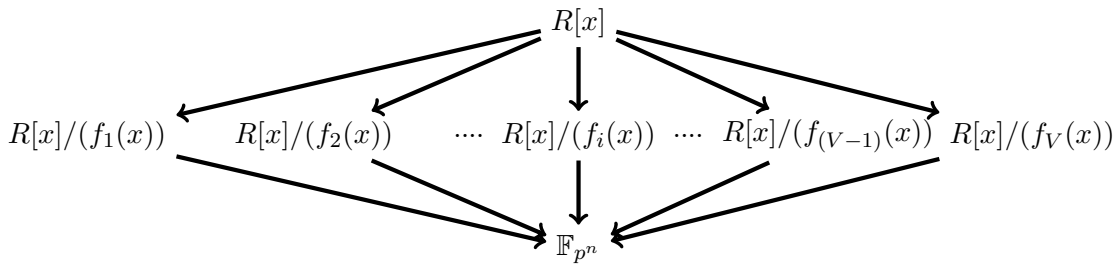

FIGURE 4.1: A work-flow of NFS.

R is the integer ring $\mathbb{Z}$ in case of the basic version of NFS, $\mathbf{F}_p[y]$ in case of FFS, a number ring $\mathbb{Z}[y]/(h(y))$ in case of Tower NFS, where a tower of number fields are considered. In the general case, V irreducible polynomials $f_i(x)$ are selected from $R[x]$ such that the maps from $R[x]/(f_i(x))$ to the field $\mathbb{F}_{p^n}$ is commutative.

It has already been discussed in Section 4.1 that NFS is well-suited for large characteristic fields with heuristic complexity around $L_Q(\frac{1}{3})$. It may be applied to medium characteristic fields too. There are various variants of NFS, namely, the basic version of NFS, the multiple number field sieve (MNFS), the tower number field sieve [14], extension of tower number field sieve called exTNFS [107, 108] in case the degree of extension of the finite field is composite, the special number field sieve [104] for primes having special form. MNFS itself again has two variants namely, the asymmetric one [50] which is the one most generally used and the symmetric case [15].
In the most basic version of NFS, polynomial selection phase outputs two polynomials which act as the defining polynomials for the number fields $\mathbb{Z}[x]/(f_i(x))$, $i = 1, 2$ respectively. The defining polynomial of the finite field is given by the gcd of these polynomials modulo the prime p. In the next phase for sieving, polynomials with bounded coefficients and fixed degree are considered. A relation is found when the norms of the image of the considered polynomial in $R[x]/(f_i(x))$ is B-smooth for B fixed beforehand. In order to compute the logarithm of the target element a two step procedure is followed. The target element is subjected to smoothing until all the constituent elements in one of the number field have some smoothness generally larger than B. An iterative smoothing, is applied to each constituent factor which allows to recursively write the target in terms of factors having smoothness smaller than B. A descent tree is obtained where the root is the ideal with a slightly greater smoothness bound and the nodes are the ideals with reduced smoothness bound at each step. The logarithm of the target can be expressed as the logarithm of smaller ideals which are known beforehand. The papers [105, 61] provide further details. Guillevic [84] proposed faster initial splitting algorithms for NFS and Tower variants.

The papers included in this thesis do not contain any new improvement regarding NFS. Due to this, details concerning NFS are not considered over here. One may refer to the theses [13,

163].

Chapter 5

New Discrete Logarithm Computation for the Medium Prime Case Using the Function Field Sieve

5.1 Introduction

Let p be a prime and n be a positive integer. Let $\mathbb{F}_{p^n}$ be the finite field consisting of p^n elements. Let $\mathfrak{g}$ be a generator of the cyclic group of all non-zero elements of $\mathbb{F}_{p^n}$.
In this chapter, we have considered discrete logarithm computation for a medium characteristic prime using FFS. This progress, in this arena, has not been as remarkable as in the small characteristic case. Important simplification of the FFS was made by Joux and Lercier [100]. The next work was by Joux [99] who introduced the important idea of pinpointing. Later work by Sarkar and Singh [151] performed a detailed asymptotic analysis. [151, 51] presents a corrected version of the analysis.

All three of the works [100, 99, 151] reported discrete logarithm computations. These are summarised in Tables 5.1 and 5.2. Table 5.1 compares the various discrete logarithm computations in the medium prime case using the FFS algorithm. Comparison of previously performed discrete logarithm computations for Kummer extensions are shown in Table 5.2. For Kummer extensions, the condition $n \mid (p-1)$ holds. This does not arise in most cryptographic applications. So, they do not have real cryptographic significance.

In Tables 5.1 and 5.2, $\#\mathbb{B}$ is the size of the factor basis. The parameter Λ is a measure of the feasibility of 2-1 descent. The lower the value of Λ, the more difficult it is to carry out a 2-1 descent. We provide the definition of Λ and explain its connection to the difficulty of 2-1 descent later.

The present chapter represents progress in the discrete logarithm computation for the medium prime case using the FFS algorithm. The challenge was to perform a larger discrete logarithm

Ref	$\lceil \log_2 p \rceil$	n	$\lceil \log_2 p^n \rceil$	$\lceil \log_2 \#\mathbb{B} \rceil$	Λ
JL [100]	17	25	401	18	3.79
SS [151]	16	37	592	17	0.11
SS [151]	19	40	728	20	0.08
This work	22	50	1051	23	0.07

TABLE 5.1: A comparison of the difficulty of computing discrete logarithms for the medium prime case using the function field sieve algorithm.

Ref	$\lceil \log_2 p \rceil$	n	$\lceil \log_2 p^n \rceil$	$\lceil \log_2 \#\mathbb{B} \rceil$	Λ
JL [100]	19	30	556	18	4.29
Joux [99]	25	47	1175	20	0.77
Joux [99]	25	57	1425	20	0.13

TABLE 5.2: A comparison of the difficulty of computing discrete logarithms for the medium prime case using the function field sieve algorithm for Kummer extensions, i.e., for fields $\mathbb{F}_{p^n}$ satisfying $n \mid (p-1)$.

computation for a medium prime case field than what has been reported earlier. To keep the problem general, we decided not to work with Kummer extensions. We chose a 1051-bit field having a 22-bit characteristic and extension degree 50 as our target. While the size of this field is smaller than the 1175-bit and the 1425-bit fields considered by Joux [99], the Kummer extension property of the latter two fields make the discrete logarithm computation much easier than the field considered in this chapter. In particular, for the fields considered in [99], 20-bit factor bases suffice whereas in our case a 23-bit factor basis is required. Also, the 2-1 descent for the field considered in this chapter is more difficult than those considered in [99]. This is indicated by the value of Λ in Tables 5.1 and 5.2. More details on the 2-1 descent are provided later. Our record computation is chronologically present in the website `https://dldb.loria.fr/?filter=all&sort=date`.

For our computation, the main techniques that were used are from [99, 151]. Applying these techniques to a larger field created complications, especially in the descent step. This required building on and implementing the alternating walk and branching technique. We considered the feasibility of using the FFS algorithm to solve a discrete logarithm challenge for a field having a 32-bit characteristic and extension degree 17. For this challenge, the relation collection and the descent phases were well within reach. The linear algebra step, on the other hand, required much more time. The ability to solve the discrete logarithm challenge depended on the feasibility of the linear algebra step.

5.2 Sieving Using Partial Smoothness-Divisibility

The technique of smoothness-divisibility and a sieving method based on it was introduced in [151]. Here we provide a brief account of this method based on the relations $y = g_1(x) = x^{-n_1}$ and $x = g_2(y)$. (This description is somewhat different from the one in [151] which was based on $y = g_1(x) = x^{n_1}$ and $x = g_2(y)$.)

Let $\phi(x)$ be a polynomial of degree $d \geq 0$. Let $T(x, y)$ be a bivariate polynomial and let $F(x)$ and $C(y)$ be such that $T(x, g_1(x)) = F(x)/x^{n_1}$ and $C(y) = T(g_2(y), y)$. The polynomial $T(x, y)$ is said to be *good for* $\phi(x)$ if $\phi(x)$ divides $F(x)$ and both $G(x) = F(x)/\phi(x)$ and $C(y)$ are smooth, i.e., can be factored into linear polynomials. Note that $F(x) = \mathrm{Res}_y(T(x, y), x^{n_1}y - 1)$, and $C(y) = \mathrm{Res}_x(T(x, y), x - g_2(y))$.

Suppose $T(x, y)$ is a monic polynomial which has a total of $\rho + 1$ monomials. We assume $d < \rho$. Let the degrees of $F(x)$ and $C(y)$ be ρ_1 and ρ_2 respectively. The degree of $G(x)$ is $\rho_1 - d$. Let $e = \rho - d$ which represents the degree of freedom. Let us write

$$F(x) \quad = \quad \phi(x)(x - a_1) \cdots (x - a_e)H(x) \tag{5.1}$$

for some polynomial $H(x)$ of degree $h = \rho_1 - d - e$. So, $G(x) = (x - a_1) \cdots (x - a_e)H(x)$. If we can find $a_1, \ldots, a_e \in \mathbb{F}_p$ such that $F(x)$ can be written as in (5.1), then we are able to ensure that $F(x)$ is divisible by $\phi(x)$ and partial smoothness of $G(x)$. By trying various values of $a_1, \ldots, a_e$, the smoothness of $H(x)$ and the corresponding $C(y)$ has to be ensured. A sieving based method for implementing this idea has been described in [151].

The partial smoothness-divisibility technique is useful for both relation collection and the descent step. In the context of relation collection, we set $T(x, y) = xy + ay + bx + c$ and $\phi(x) = 1$ so that $\rho = 3$ and $d = 0$ leading to $e = \rho - d = 3$. The resulting sieving technique can be combined with pinpointing. We refer to [151] for further details.

For application to the descent step, the 2-1 descent is described in detail in [151] since it is for such descent that the partial smoothness-divisibility technique was applied. Here we describe how the technique can be used for d-$(d - 1)$ descent for $d \geq 2$. Let $\phi(x)$ be a polynomial of degree d and the goal is to descend to polynomials of degrees less than d. Let $\rho = d + 1$ so that $e = \rho - d = 1$, providing a single degree of freedom. With $e = 1$, we have

$$F(x) \quad = \quad \phi(x)(x - \alpha)H(x) \tag{5.2}$$

where degree of $H(x)$ is h. The ρ undetermined coefficients of $T(x, y)$ appears in $F(x)$. As before, let $G(x) = (x - \alpha)H(x)$.

Consider α to be a symbolic variable. From (5.2) and using a method described in [151], it is possible to symbolically solve for the coefficients of $H(x)$ and $F(x)$ in terms of α. The symbolic

computation is a one-time task.

We provide an example of the symbolic computation. Suppose $n = 49 = n_1 n_2$ with $n_1 = n_2 = 7$, $T(x, y) = xy + ax + by + c$ where a, b and c are undetermined elements of $\mathbb{F}_p$. Then $F(x) = T(x, x^{n_1})$ is a polynomial of degree $\rho_1 = 8$. Let $\phi(x) = q_0 + q_1 x + q_2 x^2$, where $q_0, q_1, q_2 \in \mathbb{F}_p$. So, $d = 2$, $e = 1$ and from (5.2), the degree h of $H(x)$ is 5. Symbolic computation using SAGE [149] provides $H(x) = h_0 + h_1 x + \cdots + h_4 x^4 + x^5$ where $h_0, \ldots, h_4$ are as follows.

$$\begin{aligned}
h_0 &= -(\alpha^5 q_1^5 - \alpha^4 q_0 q_1^4 + \alpha^3 q_0^2 q_1^3 - \alpha^2 q_0^3 q_1^2 + \alpha q_0^4 q_1 - q_0^5 + (3\alpha^5 q_0^2 q_1 - \alpha^4 q_0^3) q_2^2 \\
&\quad -(4\alpha^5 q_0 q_1^3 - 3\alpha^4 q_0^2 q_1^2 + 2\alpha^3 q_0^3 q_1 - \alpha^2 q_0^4) q_2)/\Delta(\alpha); \\
h_1 &= q_0^2 q_2^3 - \alpha^4 q_1^5 + \alpha^3 q_0 q_1^4 - \alpha^2 q_0^2 q_1^3 + \alpha q_0^3 q_1^2 - q_0^4 q_1 \\
&\quad -(3\alpha^5 q_0 q_1^2 + \alpha^4 q_0^2 q_1) q_2^2 + (\alpha^5 q_1^4 + 3\alpha^4 q_0 q_1^3 - 2\alpha^3 q_0^2 q_1^2 + \alpha^2 q_0^3 q_1) q_2)/\Delta(\alpha); \\
h_2 &= (2\alpha^5 q_0 q_1 q_2^3 - \alpha^3 q_1^5 + \alpha^2 q_0 q_1^4 - \alpha q_0^2 q_1^3 + q_0^3 q_1^2 - (\alpha^5 q_1^3 + 2\alpha^4 q_0 q_1^2 + 2\alpha^3 q_0^2 q_1 - \alpha^2 q_0^3) q_2^2 \\
&\quad +(\alpha^4 q_1^4 + 3\alpha^3 q_0 q_1^3 - 2\alpha^2 q_0^2 q_1^2 + \alpha q_0^3 q_1 - q_0^4) q_2)/\Delta(\alpha); \\
h_3 &= -(\alpha^5 q_0 q_2^4 + \alpha^2 q_1^5 - \alpha q_0 q_1^4 + q_0^2 q_1^3 - (\alpha^5 q_1^2 + \alpha^4 q_0 q_1 + \alpha^3 q_0^2) q_2^3 \\
&\quad +(\alpha^4 q_1^3 + 2\alpha^3 q_0 q_1^2 + 2\alpha^2 q_0^2 q_1) q_2^2 - (\alpha^3 q_1^4 + 3\alpha^2 q_0 q_1^3 - 2\alpha q_0^2 q_1^2 + 2 q_0^3 q_1) q_2)/\Delta(\alpha); \\
h_4 &= -(\alpha^5 q_1 q_2^4 + \alpha q_1^5 - q_0 q_1^4 - (\alpha^4 q_1^2 + \alpha^3 q_0 q_1) q_2^3 + (\alpha^3 q_1^3 + 2\alpha^2 q_0 q_1^2 + \alpha q_0^2 q_1 - q_0^3) q_2^2 \\
&\quad -(\alpha^2 q_1^4 + 3\alpha q_0 q_1^3 - 3 q_0^2 q_1^2) q_2)/\Delta(\alpha),
\end{aligned}$$

where

$$\begin{aligned}
\Delta(\alpha) &= \alpha^5 q_2^5 - q_1^5 - (\alpha^4 q_1 + \alpha^3 q_0) q_2^4 + (\alpha^3 q_1^2 + 2\alpha^2 q_0 q_1 + \alpha q_0^2) q_2^3 \\
&\quad -(\alpha^2 q_1^3 + 3\alpha q_0 q_1^2 + 3 q_0^2 q_1) q_2^2 + (\alpha q_1^4 + 4 q_0 q_1^3) q_2.
\end{aligned}$$

Once the polynomial $\phi(x)$ in (5.2) is fixed, the coefficients of $H(x)$ and $G(x)$ are functions of α. For each possible value of α, denote the corresponding $H(x)$ and $G(x)$ as $H_\alpha(x)$ and $G_\alpha(x)$ respectively. Next, for each possible value of $\alpha \in \mathbb{F}_p$, compute the coefficients of $H_\alpha(x)$ and hence obtain $G_\alpha(x) = (x - \alpha) H_\alpha(x)$. Store all the $G_\alpha(x)$'s in a list $\mathcal{L}$. After $G_\alpha(x)$ has been added to $\mathcal{L}$ for all $\alpha \in \mathbb{F}_p$, sort $\mathcal{L}$. If a $G(x)$ occurs $h - d + 2$ or more times in the list, then at least $h - d + 2$ roots of $G(x)$ have been encountered in the sieving process. The remaining factor of $G(x)$ has degree at most $d - 1$ and so $G(x)$ is $(d - 1)$-smooth. For such a $G(x)$, construct the corresponding $F(x)$ as $\phi(x)G(x)$. Using Proposition 1 of [151], obtain $C(y)$ and check whether $C(y)$ is also $(d - 1)$ smooth. If it turns out that $C(y)$ is indeed $(d - 1)$ smooth, then we have

$$\phi(x) = \frac{x^{n_1} C(y)}{G(x)}$$

where both $C(y)$ and $G(x)$ are $(d - 1)$-smooth. So, it has been possible to descend from the polynomial $\phi(x)$ of degree d to the polynomials $C(y)$ and $G(x)$ which are $(d - 1)$-smooth. Note that for $d > 2$, while aiming for d-$(d - 1)$ descent, it might be possible to reach smaller degrees if in the sieving process, a $G(x)$ appears more than $h + d - 2$ times. The above description is for descending from a polynomial $\phi(x)$. A similar method works for descending from a polynomial

$\psi(y)$.

Suppose the above method is not successful, i.e., the sieving procedure does not result in a suitable $G(x)$ and $C(y)$. At this point, there are several ways to proceed.

The x-x Walk: Suppose the sieving procedure results in a $G(x)$ which has $h - d + 1$ linear factors. Then the other factor of $G(x)$ is of degree d. Let this factor be $\phi_1(x)$. Further, suppose that $C(y)$ turns out to be $(d-1)$-smooth. Then, in effect, we have moved from a $\phi(x)$ of degree d to the polynomial $\phi_1(x)$ also of degree d. Descent may now be attempted from $\phi_1(x)$. Similarly, one may need to move from a polynomial $\psi(y)$ of degree d to a polynomial $\psi_1(y)$ also of degree d and try to descend from $\psi_1(y)$. Such a method is called the walk technique.

The x-y Walk: Suppose the sieving procedure results in a desirable $G(x)$, i.e., one that has at least $h - d + 2$ linear factors. On the other hand, suppose that the corresponding $C(y)$ turns out to be d-smooth, instead of being $(d-1)$-smooth. For each factor $\psi(y)$ of $C(y)$ of degree d, one may try to descend to lower degree polynomials.

Similarly, one may define the y-y and the y-x walks. It is possible that neither x-x nor x-y walks succeed for a polynomial $\phi(x)$ of degree d. Then the following strategies can be tried.

1. Move from a single degree d polynomial in x to two degree d polynomials in x.
2. Move from a single degree d polynomial in x to one degree d polynomial in x and one degree d polynomial in y.

Analogous strategies hold for moving from a degree d polynomial $\psi(y)$ in y. Since this strategy moves from a single degree d polynomial to two degree d polynomials, it is called a branching strategy.

The walk and branching strategies were briefly mentioned in [99]. Detailed discussion of these strategies in the context of 2-1 descent is given in [151]. The computations in [151] used these techniques only for 2-1 descent. For the present computation, we needed the walk technique for both 3-2 and 2-1 descent.

For the x-y walk, an important implementation issue is to avoid cycling. Suppose the x-y walk starts from $\phi(x)$. It is possible that after a number of steps, the walk again enters $\phi(x)$. This is called cycling. In the presence of cycling, the descent fails. By suitably using randomisation, it is usually possible to avoid such cycling. Alternatively, cycle detection algorithms may be used to detect the presence of cycling and abort.

5.3 A Concrete Discrete Logarithm Computation

In this section, we present the details of an actual discrete logarithm computation. The preparatory phase, relation collection and the descent steps were done using Magma V2.21-10 on four

servers. Out of the four servers, three have the same configuration with each of these three servers consisting of Intel(R) Xeon(R) E7-4890 @ 2.80 GHz (60 physical cores and 120 logical cores) and the fourth server consists of Intel Xeon E7-8890 @ 2.50 GHz (72 physical cores and 144 logical cores). These servers are shared resources and were simultaneously utilised by other users to run heavy simulation programs. We were never able to obtain exclusive access to the servers. The linear algebra phase was run on a cluster of 16 dual-socket Intel(R) Xeon(R) Gold 6130 CPU @ 2.10GHz connected with Intel 100 Gbps Omni-Path.

In our computation, we chose $p = 2111023$ and $n = 50$. Note that $\lceil \log_2(p) \rceil = 22$ and $\lceil \log_2(p^n) \rceil = 1051$. So, the discrete logarithm computation is over a 1051-bit field having a 22-bit characteristic.

Preparatory Phase: We chose $n_1 = n_2 = 7$. Experimentally we obtained $g_1(x) = x^{-7}$ and $g_2(x) = x^7 + 1224488$ such that $x - g_2(g_1(x)) = f(x)/x^{49}$ where $f(x) = x^{50} + 886535x^{49} + 2111022$. The polynomial $f(x)$ is irreducible over $\mathbb{F}_p$ and we represented $\mathbb{F}_{p^n}$ as $\mathbb{F}_p[x]/(f(x))$. Under this representation, $x + 11$ turned out to be a primitive element and was taken as the base of our discrete logarithm computation.

The factorization of $(p^n - 1)/(p - 1)$ is the following.

$$\begin{aligned}\frac{p^n-1}{p-1} &= 2^5 \cdot 3^4 \cdot 11 \cdot 31 \cdot 83 \cdot 101 \cdot 131 \cdot 157 \cdot 251 \cdot 6361 \cdot \\ &\quad 12241 \cdot 131939 \cdot 839532251 \cdot 896407381 \cdot \\ &\quad 3943088101 \cdot 164534375651 \cdot 3062950366849991 \cdot 36244934276573651 \cdot \\ &\quad 752902385776306150901 \cdot p_1 \cdot p_2 \cdot p_3\end{aligned}$$

where

$$\begin{aligned}p_1 &= 204692161033930730108568803278296327232200 1; \\ p_2 &= 553059810014751323913181174167985322787847067 51; \\ p_3 &= 923983173059841394501412330899341649381951887 56\backslash \\ &\quad 2818470664758881417032041985737711765745606292778672234895 1.\end{aligned}$$

Note $\lceil \log_2(p_1) \rceil = 141$, $\lceil \log_2(p_1) \rceil = 156$ and $\lceil \log_2(p_1) \rceil = 353$.

Based on the choices of $g_1(x)$ and $g_2(x)$, we have $y = g_1(x) = x^{-7}$ and $x = g_2(y) = y^7 + 1224488$. The factor basis was set to be $\mathbb{B} = \{(x + a_i), (y + b_j) : a_i, b_j \in \mathbb{F}_p\}$.

Relation Collection: The relation collection was done using sieving based on partial smoothness-divisibility technique combined with pinpointing. The computation is highly parallelisable. It was distributed on four servers with 90 processes per server. The total time required for the relation collection phase was about 25 hours. Assuming that our jobs were allocated about 75% of the server time, a rough estimate of the number of core-years required for relation collection is 0.53 core-years.

A total of $2p + 100 = 4222146$ relations were generated among the elements of the factor basis which includes the relation $y = x^{-7}$. Except for $(y + 1849709)$, all elements of the factor basis were involved in at least one relation.

The fact that none of the $2p+100$ relations involved $(y+1849709)$ seemed peculiar to us. So, we decided to investigate this further. For this we considered applying the partial smoothness-divisibility technique for relation collection from the y-side. The starting point of this technique is to write

$$C(y) \quad = \quad (y - \alpha_1)(y - \alpha_2)(y - \alpha_3)H_1(y).$$

We set $(y - \alpha_1) = (y + 1849709)$. This is to ensure that any relation obtained from $C(y)$ and the corresponding $F(x)$ will necessarily involve $(y + 1849709)$. There are two degrees of freedom given by α_1 and α_2. This allows trying p^2 options. We were surprised to find that no relation could be obtained. It was possible to ensure that $C(y)$ is smooth. However, in each such case, it turned out that the corresponding $F(x)$ is not smooth. This suggests that it may indeed be the case that there is no relation among the factor basis elements which involves $(y + 1849709)$.

Since we were unable to obtain any relation involving $(y + 1849709)$, we decided to proceed without this element. The resulting matrix for the linear algebra stage consisted of $2p + 100$ relations involving $2p - 1$ unknowns.

Linear Algebra: The linear algebra step was performed for the three primes p_1, p_2 and p_3. The block Wiedemann algorithm implemented in the CADO-NFS software was used to complete the linear algebra step.

Let M denote the matrix obtained as input for the linear algebra step. Also, let N be the number of rows of M which is the number of relations collected. Given a prime l, block Wiedemann algorithm consists of three steps namely: *Krylov*, *Lingen*, *Mksol*. Let m, n be integers decided beforehand. These are called *blocking parameters* and may be chosen freely. For efficency purpose of lingen, m may be chosen as $2n$. After choosing m, n a block of m vectors denoted by x and another block of n vectors denoted by y is chosen. The aim of block Wiedemann algorithm is to compute a sequence of $m \times n$ matrices $(x^T M^i y)_{i \geq 0}$.

Looking into the above matrices column wise, the *Krylov* step computes n sequences. It can be distributed into n parallel tasks, each computing $(x^T M^i y^{(j)})_{i \geq 0}$. Tasks can be synchronized at the end when results are combined. The *Lingen* computes a linear generator for the previous sequence. It outputs n generators $F^{(0)}, F^{(1)}, \ldots, F^{(n-1)}$ which are polynomials in $\mathbb{Z}/l\mathbb{Z}$ of degree less than $\lceil (\frac{N}{n}) \rceil$. The *Mksol* step computes $w = \sum_{j=1}^{n} F^{(j)}(M) y^{(j)}$ which belongs to the kernel of M. It can be distributed into n independent computations.

For the largest prime p_3, the Krylov step took about 1.6 core years, Lingen required negligible time and Mksol required about 0.25 core years. The time requirements for the two smaller sized primes were smaller. In terms of space, for p_1 and p_2 about 29GB each was required and for p_3 about 53GB was required. We used $n = 4$ distinct sequences for the block Wiedemann algorithm, so that we were able to simultaneously use 16 nodes as 4 groups of 4 nodes, each group working on one sequence.

After the linear algebra step, the discrete logarithms of all the elements in the factor basis other than $(y + 1849709)$ were obtained modulo p_1, p_2 and p_3.

Individual Logarithm - Descent to Factor Basis Elements: As the target for the individual discrete logarithm computation we chose the following element derived from the digits of the real number π. The function 'Normalize' mentioned below makes the input polynomial monic by multiplying with the inverse of the leading coefficient.

$$\Pi(x) \quad = \quad \text{Normalize}\left(\sum_{0}^{(n-1)} \lfloor \pi \cdot p^{i+1} \mod p \rfloor x^i\right).$$

Explicitly $\Pi(x)$ is given by the following degree 49 polynomial.

$$\begin{aligned}\Pi(x) \quad = \quad & x^{49} + 308380x^{48} + 467398x^{47} + 934029x^{46} + 37835x^{45} + 2003442x^{44} + 174801x^{43} \\ & +1414683x^{42} + 733114x^{41} + 1077558x^{40} + 1049867x^{39} + 1848765x^{38} + 1653554x^{37} \\ & +949244x^{36} + 1627181x^{35} + 1592837x^{34} + 652981x^{33} + 1601022x^{32} + 635134x^{31} \\ & +900855x^{30} + 413911x^{29} + 74385x^{28} + 2057944x^{27} + 930210x^{26} + 310181x^{25} + \\ & +118528x^{24} + 1515849x^{23} + 93830x^{22} + 393848x^{21} + 644073x^{20} + 1018627x^{19} \\ & +1654544x^{18} + 611872x^{17} + 1491385x^{16} + 1797395x^{15} + 1833421x^{14} \\ & +1711611x^{13} + 406154x^{12} + 1588768x^{11} + 530413x^{10} + 1458736x^{9} \\ & +696502x^{8} + 496320x^{7} + 196737x^{6} + 535254x^{5} + 194167x^{4} \\ & +977109x^{3} + 1911333x^{2} + 1037166x + 1347394.\end{aligned}$$

For the initial descent a simple randomisation strategy was utilised which led to

$$\Pi(x) \quad = \quad \frac{N(x)}{D(x)}$$

where $N(x)$ and $D(x)$ are as follows.

$$
\begin{aligned}
N(x) \quad = \quad & (x+1424244)(x^3+237998x^2+42029x+734901)(x^3+299760x^2+1210894x+1086517) \\
& (x^4+1182727x^3+563430x^2+1055902x+1247639) \\
& (x^4+1251838x^3+723661x^2+1707546x+110202) \\
& (x^5+221654x^4+445454x^3+650438x^2+1275751x+124811) \\
& (x^5+665157x^4+337641x^3+1409401x^2+1379166x+322114) \\
& (x^5+927040x^4+199439x^3+342445x^2+1316050x+1494757) \\
& (x^6+61134x^5+1695168x^4+2017581x^3+293438x^2+766784x+1054073) \\
& (x^6+1565656x^5+129255x^4+419731x^3+1556013x^2+2087232x+207329) \\
& (x^7+1746884x^6+469847x^5+382378x^4+425150x^3+944772x^2+530084x+1756060), \\
D(x) \quad = \quad & (x+3541)(x+87748)(x+110850)(x+119667)(x+241035)(x+305058) \\
& (x+395128)(x+399638)(x+422176)(x+578119)(x+582549)(x+586316) \\
& (x+662109)(x+770637)(x+772129)(x+775849)(x+800910)(x+865556) \\
& (x+902073)(x+971438)(x+1011431)(x+1052833)(x+1060253)(x+1062580) \\
& (x+1103078)(x+1132166)(x+1147933)(x+1174406)(x+1176644)(x+1189750) \\
& (x+1231997)(x+1248106)(x+1289845)(x+1297742)(x+1347100)(x+1418494) \\
& (x+1433230)(x+1528574)(x+1579870)(x+1596791)(x+1660898)(x+1741103) \\
& (x+1805530)(x+1849061)(x+1912058)(x+2041324).
\end{aligned}
$$

Note that $N(x)$ is 7-smooth while $D(x)$ is smooth. This decomposition required about 30 minutes using 50 processes.

In the next step, the goal was to reduce to quadratic polynomials using successive d-$(d-1)$ descent for polynomials of degree $d > 2$. This strategy mostly succeeded, except for three cubic polynomials. For these three polynomials, we had to resort to alternating walk, i.e., move from degree 3 polynomials in x to degree 3 polynomials in y, as explained earlier. Using such a walk, we were able to descend to quadratic and linear polynomials. The total number of quadratic polynomials that were generated was 1212, of which 673 were quadratic polynomials in x and 539 were quadratic polynomials in y. The total time required for descending to quadratic polynomials was about 1481 minutes using 50 processes.

Finally we performed the 2-1 descent on all the quadratic polynomials. This was the most time consuming of all the descent steps. We used 50 processes on each of the four servers to perform either direct 2-1 descent or to apply alternating walk and/or branching. The entire 2-1 descent step was automated. The total time for all the 2-1 descents required about 10 days. Assuming that our jobs were allocated about 75% of the server time, a rough estimate of the number of core-years required for all the 2-1 descents is about 4.1 core-years.

Remark: The descent to quadratic polynomials resulted in a total of 1212 quadratic polynomials. For comparison, we mention the number of quadratic polynomials obtained in previous computations. In [151], the numbers of quadratic polynomials were 92 and 59 for the 592-bit and the 728-bit cases respectively. In [99], the number of quadratic polynomials for the 1125-bit case was 278; the number of quadratic polynomials for the 1425-bit case was not reported.

We note that the descent is a random procedure. So, different runs of the descent procedure

may lead to different numbers of quadratic polynomials. Since the 2-1 descent is the most time consuming of all the descent steps, it would be worthwhile to try and minimise the number of quadratic polynomials that arise from the upper levels of the descent. There is, however, no known method for such minimisation.

Individual Logarithm - Final Discrete Logarithm Computation: After the completion of the 2-1 descent, the target polynomial $\Pi(x)$ was expressed as a ratio $N_1(x)/D_1(x)$, where $N_1(x)$ is a product of 147126 linear polynomials in x and 127149 linear polynomials in y, and $D_1(x)$ is a product of 147164 linear polynomials in x and 126001 linear polynomials in y.

Recall that the relation collection and linear algebra steps were not able to compute the discrete logarithm of $(y + 1849709)$. Fortunately, this element does not appear among the linear factors of $N_1(x)$ and $D_1(x)$. In case it occurred as one of the factors of $N_1(x)$ or $D_1(x)$, alternating walk and/or branching had to be done.

The discrete logarithms modulo p_1, p_2 and p_3 of the linear factors of $N_1(x)$ and $D_1(x)$ had already been obtained after the completion of the linear algebra step. Consequently, after the descent step we were able to obtain the discrete logarithm of $\Pi(x)$ modulo p_1, p_2 and p_3.

We used Pollard rho and Pohlig-Hellman to compute the discrete logarithm of $\Pi(x)$ modulo the smaller factors of $p^n - 1$. The final discrete logarithm of $\Pi(x)$ to base $(x + 11)$ was computed using the Chinese Remainder Theorem. This value is given below.

$$
\begin{aligned}
&\log_{x+11}(\Pi(x)) \\
&= 1323496538911863968895271989039865754003499138979788669347646690861304065811174\backslash \\
&\quad 1258084796458565453948561234741578427747721119813849133097458001127822232655615\backslash \\
&\quad 0735099096613330104434651232074005278625612674879570628049934937631130006839219\backslash \\
&\quad 5452506485478263044561377117997258194255748683503064110129248778733465564209650 1.
\end{aligned}
$$

A short Magma program to verify the discrete logarithm is given in the Section 5.4.

5.3.1 Experiments with Filtering

The filtering step can reduce the size of the matrix to which the linear algebra step is to be applied. The basic filtering step that we applied ensured that there are no duplicates and no empty column.

Following a suggestion from a reviewer, we carried out experiments to determine the effectiveness of the filtering step. To this end, we utilised the filtering algorithm implemented in the CADO-NFS software which has been used in the recent 240-bit DLP computation [32]. For the filtering algorithm, an important parameter is the distribution of the weights of the columns. In the relations that we collected, the variable x is present in all the relations and the corresponding column has weight equal to the number of rows. While mentioning the maximum weight of columns, we will ignore this particular column. For our experiments, we converted the relation matrix obtained using Magma to a format suitable to CADO-filtering and used the subroutines

purge, merge-dl and replay-dl of CADO-NFS in the given order. Two sets of experiments were performed.

The first experiment was on the relation matrix used for the DLP computation over $GF(p^{50})$ with $p = 2111023$. This matrix has 4222146 (i.e., $2p+100$) rows and 4222045 (i.e., $2p-1$) columns; each row has at most 23 non-zero entries; the maximum weight of a column is 40 and the minimum weight is 1 and if we ignore a few thousand columns, the minimum weight is 6; the overall weight of the matrix is approximately $(2p + 100)23 = 97109358$. On applying filtering, we obtained a matrix with dimension 3234915×3234905 and weight 248322355. We note that the reduction in the dimension of the matrix is not much, but the overall weight of the matrix increases 2.5 fold.

The second experiment was for a smaller prime where the number of relations that we collected was about 50 times more than what is needed. The following parameters were considered: $n = 37$, $p = 64373$, $n_1 = n_2 = 6$, $g_1(X) = X^{-6}$, $g_2(X) = X^6 + 14833X^5 + 50952X^4 + 62125X^3 + 6269X^2 + 35223X + 53172$, and $f(X) = X^{37} + 11201X^{36} + 29150X^{30} + 58104X^{24} + 2248X^{18} + 13421X^{12} + 49540X^6 + 64372$. With these parameters, we collected $56(2p)$ relations leading to a matrix of dimension $112p \times (2p - 1)$. Each row has at most 20 non-zero entries; the maximum weight of a column is 923 and the minimum weight is 669; the overall weight of the matrix is about $20(112p)$. On applying filtering, we obtained a matrix with dimension 94731×93731 with weight 6679667. If we had instead collected only $2p + 100$ relations, then the dimension of the matrix would have been $(2p + 100) \times 2p$, with $2p = 128746$ and the total weight of the matrix would have been $20(2p) = 2574920$. So, the filtering did not reduce the dimension of the matrix substantially, but increased the overall weight of the matrix by a factor of 4.

Both the above experiments indicate that the CADO-filtering step is not very effective for this variant of the FFS algorithm. On the other hand, the filtering step leads to a substantial reduction in the dimension of the matrix for the NFS algorithm as can be noted from the recent 240-bit DLP computation in [32]. A possible explanation for this difference in behaviour is the following. In the context of NFS, filtering works well because it exploits the fact that typical NFS matrices are extremely far from having uniform density. There are dense "small-prime" columns as well as extremely sparse "large-prime" columns. Most of the story in NFS filtering is connected to the idea of getting rid of that immense near-vacuum formed by the sparsest columns, without making the resulting matrix too dense. In the present context, all columns (apart from the single column corresponding to the variable x) have more or less uniform weights. So, it is perhaps not completely surprising that the heuristics that work with typical NFS matrices do not play out well in the present context.

5.4 Magma Script to Verify the Computation

```
n := 50;
p := 2111023;
```

```
assert(IsPrime(p));
Fp := GF(p);
FpX<X> := PolynomialRing(Fp);
fX := X^50 + 886535*X^49 + 2111022;
assert IsIrreducible(fX);
Fpnx<x> := ext<Fp|fX>;

RR:=RealField();
pi := Normalize(&+[(Floor(Pi(RR)*p^(i+1)) mod p)*X^i : i in [0..n-1]]);

log := 13234965389118639688952719890398657540034991389797886693476466 90\
86130406581117412580847964585654539485612347415784277472 1119813\
84913309745800112782223265561507350990966133301044346512320 74005\
27862561267487957062804993493763113000683921954525064854782 63044\
56137711799725819425574868350306411012924877873346556420965 01;
base := x+11;
target := Fpnx!Eltseq(pi);

printf "base := %o\n",base;
printf "target := %o\n",target;
printf "log := %o\n", log;
printf "base^log eq target = %o\n",base^log eq target;
```

5.5 Unsolved DLP Challenge for the Medium Prime Case

In [117], the following has been stated.

> "For powers of very small primes and for large prime fields the function-field sieve and the number-field sieve are highly optimized; for intermediate fields algorithms with the same asymptotic behaviour exist but the actual running times are slower."

To encourage research for intermediate size fields, the following challenge has been proposed in [117]. Solve DLP in $\mathbb{F}_{p^{17}} = \mathbb{F}_p[x]/(x^{17} - 2)$ where $p = 2^{32} - 27897$.

This problem can be tackled using the FFS for the medium prime case. For this, we set $n_1 = n_2 = 4$ so that $n = n_1 n_2 + 1 = 17$. We estimate the costs of the relation collection, linear algebra and the descent steps.

- *Cost of linear algebra:* For this problem, $n \mid (p - 1)$ which will allow the factor basis to be reduced by a factor of 17. So, the size of the factor basis will be about 2^{29} and hence, linear algebra step will require about 2^{58} operations in $\mathbb{F}_p$.

- *Cost of relation collection:* Using $n_1 = n_2 = 4$, from (4.6), the number of trials required to obtain a single relation is about $2^{6.9}$. So, the total number of trials required to obtain about 2^{29} relations is about $2^{35.9}$. Further, the feasibility condition given by (4.4) holds.
- *Cost of 2-1 descent:* Using $n_1 = n_2 = 4$, from (4.7), the number of trials required to obtain a single 2-1 descent is about $2^{9.5}$. We would expect a few thousand quadratic polynomials would be required to be descended. The feasibility condition given by (4.8) holds.

So, based on the above analysis, we see that while the relation collection and the descent steps are well within reach, the major cost of performing the computation required for solving the challenge lies in the linear algebra computation. In case, a fraction of all degree-one polynomials are included in the factor base, it will decrease the cost of linear algebra phase but will increase the cost of relation collection. The effectiveness of this cannot be determined without actual implementation.

Remark: The above estimates are rough. For one thing, we have estimated that the time required for the linear algebra step on a factor basis of size N is about N^2. Secondly, the time for individual operations for the linear algebra step and the times for individual trials of the relation collection step and the 2-1 descent step have been assumed to be equal. To obtain more precise estimates, these issues would be required to be taken into consideration. Nevertheless, even with the rough estimates, the main conclusion that for this particular DLP computation it is the linear algebra step which will be the main bottleneck, remains valid.

Larger Extension Degree: The extension degree suggested in the above problem is 17 which is quite low. Let us consider a higher value of n. Suppose $n_1 = n_2 = 8$ and $n = n_1 n_2 + 1 = 65$. As in the above problem, assume that p is a 32-bit prime. Also, let us not make the assumption that $n \mid (p-1)$ holds. So, the factor basis will have about $2p$ elements. From (4.6), the number of trials required to obtain a single relation is about 2^{18} and to obtain about 2^{33} relations, about 2^{51} trials would be required. From (4.7), the number of trials required to obtain a single 2-1 descent is about 2^{31}. The feasibility condition (4.4) and (4.8) both hold. Since the size of the factor basis is about 2^{33}, the linear algebra will require about 2^{66} $\mathbb{F}_p$-operations. So, it is the linear algebra step which will be the major bottleneck in any such discrete logarithm computation.

5.6 Conclusion

In this chapter, we have reported the computation of discrete logarithm in a 1051-bit field having a 22-bit characteristic. For the general medium prime case (i.e., fields for which the condition $n \mid (p-1)$ does not hold), the computation reported here is the current discrete logarithm record computation. The techniques used in this chapter can be extended to solve relation collection and descent phases for 32-bit primes and moderate extension degrees. It is the linear algebra phase

that will require the maximum time for performing any record discrete logarithm computation for such fields.

Chapter 6

Faster Initial Splitting for Small Characteristic Composite Extension Degree Fields

6.1 Introduction

In this chapter, our focus will be on the task of initial splitting of the individual logarithm step of FFS when applied to small characteristic, composite extension degree fields.

Let $\mathbb{F}_{p^n}$ be the finite field of p^n elements where p is a small prime and $n = n_1 n_2 > 1$ is a composite integer. For practical scenarios, $n_1 \ll n_2$. The standard algorithm for initial splitting for such fields is the Waterloo algorithm [24, 25] which has already been described in Section 4.5.2.

Guillevic [84] proposed a different algorithm for initial splitting over such fields. This algorithm iteratively generates a polynomial of degree $n_2 - d/n_1$ and tests it for B-smoothness, where d is the largest non-trivial divisor of n. The key insight utilised in [84] is that multiplying the target by an element of a proper subfield does not change the discrete logarithm modulo $\Phi_n(p)$, where $\Phi_n(x)$ is the n-th cyclotomic polynomial. This insight was earlier stated in [83]. The amortised cost of generating a polynomial in the Guillevic splitting algorithm is the same as that of the Waterloo algorithm which as mentioned earlier is $O(n^2)$ multiplications over $\mathbb{F}_p$. The advantage of Guillevic splitting is that only a single polynomial is required to be B-smooth while for the Waterloo algorithm two polynomials are required to be B-smooth.

In this chapter, we present a new algorithm for initial splitting for FFS applied to small characteristic, composite extension degree fields. We also utilise the insight that multiplying the target by elements of a proper subfield does not change the logarithm modulo $\Phi_n(p)$. Our utilisation of this insight, however, is different from that in [84]. The main improvement that we obtain over Guillevic splitting is that the cost of generating a polynomial to be tested for B-smoothness is $O(n \log_p(1/\pi_2))$ operations over $\mathbb{F}_p$ where π_2 is the probability that a polynomial

of degree $n_2 - s - 1$ over $\mathbb{F}_{p^{n_1}}$ is B-smooth. The parameter s is new in our idea and can be chosen such that the degrees of the polynomials generated by the new method is only slightly larger than those generated by the Guillevic splitting algorithm. Consequently, the smoothness probabilities and the times for smoothness testing of both the Guillevic splitting and the new algorithm are almost the same. Since $\log_p(1/\pi_2) \ll n$, the time for generating a polynomial by the new method is significantly lower than the time for generating a polynomial using the Guillevic splitting algorithm.

6.2 Preliminaries

Let p be a prime and $n > 1$ be a composite integer. Let $\mathbb{F}_{p^n}$ be the finite field of p^n elements. Write $n = n_1 n_2$. Let $h(y)$ be an irreducible polynomial of degree n_1 over $\mathbb{F}_p$. The finite field $\mathbb{F}_{p^{n_1}}$ is represented as $\mathbb{F}_p[y]/\langle h(y)\rangle$. Let $I(x)$ be an irreducible polynomial of degree n_2 over $\mathbb{F}_{p^{n_1}}$. The field $\mathbb{F}_{p^n}$ is represented as $\mathbb{F}_{p^{n_1}}[x]/\langle I(x)\rangle$. We will denote this as the $(I(x), h(y))$-representation of the field $\mathbb{F}_{p^n}$. We will further assume that a generator α of $\mathbb{F}_{p^n}^*$ is available.

Using the $(I(x), h(y))$-representation, any element $T \in \mathbb{F}_{p^n}$ can be written as follows.

$$T(x) \quad = \quad t_0(y) + t_1(y)x + \cdots + t_{n_2-1}(y)x^{n_2-1}$$

where $t_0(y), t_1(y), \ldots, t_{n_2-1}(y)$ are polynomials of degree at most $n_1 - 1$ over $\mathbb{F}_p$.

Let $\Phi_n(x)$ be the n-th cyclotomic polynomial and ℓ be a non-trivial prime divisor of $\Phi_n(p)$. Given a target $T_0 \in \mathbb{F}_{p^n}^*$, the goal is to find its discrete logarithm modulo ℓ.

The function field sieve algorithm has three broad computational steps. In the first step, a suitable factor basis is identified and linear relations among the discrete logarithms of the factor basis elements are obtained. The second step applies sparse linear algebra computation to obtain the discrete logarithms of the factor basis elements. The logarithm of the target element $T_0(x)$ is obtained in the third step which is called the individual logarithm step.

The goal of the individual logarithm step is to express the logarithm of T_0 as an $\mathbb{F}_\ell$-linear combination of logarithms of elements which are either in the factor basis or whose logarithms are known. For FFS over small characteristic and composite extension degree fields, the individual logarithm step is carried out in two parts. An *initial splitting* step followed by descent to factor basis elements.

Let B be a positive integer. A polynomial is said to be B-smooth, if all its irreducible factors have degrees less than or equal to B. The initial splitting step expresses the logarithm of T_0 in terms of logarithms of one or more polynomials each of which is B-smooth for a suitable value of B. The descent step attempts to descend the irreducible factors of the B-smooth polynomial(s) to the factor basis. In this chapter, we will only be concerned with the initial splitting step.

A simple and important result proved by Guillevic [83] and extensively used in [84] is the following.

Lemma 6.1. *Let p be a prime and $n > 1$ be an integer. Let $T \in \mathbb{F}_{p^n}$ and u be an element in a proper subfield of $\mathbb{F}_{p^n}$. Then $\log T \equiv \log uT \bmod \Phi_n(p)$ and hence $\log T \equiv \log uT \bmod \ell$ for any divisor ℓ of $\Phi_n(p)$.*

The importance of Lemma 6.1 stems from the fact that one may try to multiply the target element T_0 with a proper subfield element u to obtain $W = uT_0$ such that the degree of W is substantially less than that of T_0. Then for any integer B, the chance of W being B-smooth is significantly higher than the chance of T_0 being B-smooth.

Let g be the generator of the order ℓ subgroup of $\mathbb{F}_{p^n}$ which is of interest. Given the target T_0 and an integer $t \in \{0, \ldots, \ell - 1\}$, let $T_t = g^t T_0$ so that $\log T_t = t + \log T_0 \bmod \ell$. Since t is known, it is sufficient to find the logarithm of T_t. A well known method for initial splitting is the Waterloo algorithm [24, 25]. This algorithm expresses T_t as $T_t(x) = N(x)/D(x) \bmod I(x)$ using the extended Euclidean algorithm such that the degrees of $N(x)$ and $D(x)$ are much smaller than that of $T_t(x)$. The procedure is repeated for random choices of t until both $N(x)$ and $D(x)$ are obtained to be B-smooth for a pre-defined choice of B. Once this is obtained, initial splitting is said to have been achieved.

Improvement to the Waterloo algorithm based on Lemma 6.1 has been described by Guillevic [84]. We call this the Guillevic splitting (GS) algorithm and the complete description is shown in Algorithm-2.

Input: An $(I(x), h(y))$-representation of $\mathbb{F}_{p^n}$; generator g of the order ℓ (where $\ell | \Phi_n(p)$) subgroup over which logarithms are to be computed; a target element $T_0(x)$; and a smoothness bound B.
Output: A B-smooth polynomial $P(x)$ such that $\log P(x) \equiv \log T_0(x) \bmod \ell$.

1.1 Let d be the largest non-trivial divisor of n
1.2 Set $\mathfrak{d} = \gcd(d, n_1)$ and $d' = d/\mathfrak{d}$
1.3 Obtain $U(x) \in \mathbb{F}_{p^n}$ such that $\{1, U, \ldots, U^{d'-1}\}$ is a basis for $\mathbb{F}_{p^{d'}}$
1.4 **repeat**
1.5 Choose t randomly from $\{1, 2, \ldots, \ell - 1\}$
1.6 $T \leftarrow g^t T_0$
1.7 Define $L = \begin{bmatrix} T \\ UT \\ \vdots \\ U^{d'-1}T \end{bmatrix}$ a $d' \times n_2$ matrix over $\mathbb{F}_{p^{n_1}}$
1.8 $M \leftarrow \text{RowEchelonForm}(L)$ (with $\mathbb{F}_{p^{\mathfrak{d}}}$-linear combinations)
1.9 Set $P(x)$ as the polynomial obtained from the first row of M
1.10 **until** *P is B-smooth*;
1.11 **return** $(t, P(x))$.

Algorithm 2: Guillevic splitting for small characteristic composite order fields.

Guillevic [84] proved the following properties of Algorithm 2.

1. $P(x)$ is a polynomial of degree at most $n_2 - d/n_1$ over $\mathbb{F}_{p^{n_1}}$. So, the degree of $P(x)$ is substantially less than that of $T_0(x)$ which has degree $n_2 - 1$.
2. $P(x) = uT(x) = ug^t T_0(x)$ for some $u \in \mathbb{F}_{p^{d'}}$ and so $\log P(x) = t + \log T_0(x) \bmod \ell$ (using Lemma 6.1).

It was shown in [84] that the efficiency of Algorithm 2 can be further improved using the following two ideas.

Improvement-1: In Algorithm 2, M is obtained as the row echelon form of L. This requires running a Gaussian elimination on L. Another round of Gaussian elimination (again with $\mathbb{F}_{p^{\mathfrak{d}}}$-linear combinations) is run on M from the reverse side to obtain a matrix M' of the following form.

$$\begin{bmatrix} * & \dots & * & * & 0 & \dots & 0 \\ 0 & \ddots & & & & \ddots & \vdots \\ \vdots & \ddots & \ddots & & & \ddots & 0 \\ 0 & \dots & 0 & * & \dots & * & * \end{bmatrix} \tag{6.1}$$

The i-th row of M' is of the form $x^{e_i} P_i(x)$, with degree of $P_i(x) \leq n_2 - d/n_1$ and $e_i \approx (i-1)\mathfrak{d}/n_1$. The element x is a member of the factor basis, and so it is sufficient to obtain the logarithm of P_i. Incorporating this into Algorithm 2 requires two rounds of $\mathbb{F}_{p^{\mathfrak{d}}}$-linear Gaussian elimination. The advantage is that after the two rounds, it provides a set of d' polynomials which are to be tested for B-smoothness. This is to be contrasted with the basic description of Algorithm 2 where one round of $\mathbb{F}_{p^{\mathfrak{d}}}$-linear Gaussian elimination results in only one polynomial to be tested for B-smoothness.

Improvement-2: In each iteration, it is possible to further increase the number of polynomials to be tested for smoothness by taking $\mathbb{F}_{p^{\mathfrak{d}}}$-linear combinations of a small number of consecutive rows. This will increase the degrees of the resulting polynomials by one or two which does not significantly affect the probability of the polynomials being B-smooth.

Cost of Algorithm 2: A one-time computation is required by Algorithm 2 to obtain $U(x)$. Given α, $U(x)$ is computed as $U(x) = \alpha^{(p^n-1)/(p^{d'}-1)}$. So, obtaining $U(x)$ requires an exponentiation which in turn requires $O(n \log p)$ multiplications over $\mathbb{F}_{p^n}$. Each multiplication in $\mathbb{F}_{p^n}$ requires $O(n^2)$ operations over $\mathbb{F}_p$. Using Karatsuba this cost would be $O(n^{1.59})$ operations over $\mathbb{F}_{p^n}$ and using the Fast Fourier Transform will provide even lower asymptotic costs. We take the cost of a multiplication in $\mathbb{F}_{p^n}$ to be $O(n^2)$ so that the cost of obtaining $U(x)$ is $O(n^3 \log p)$. In practice, the actual time for obtaining $U(x)$ is negligible in comparison to the cost of generating

and testing polynomials for smoothness. So, the best asymptotic cost for computing $U(x)$ is not very relevant in practice. Note that it is not required to compute the basis $\{1, U, \ldots, U^{d'-1}\}$.

Apart from smoothness testing, the cost per iteration of Guillevic splitting algorithm consists of the following.

1. An exponentiation over $\mathbb{F}_{p^n}$ to compute g^t.
2. A total of d' multiplications over $\mathbb{F}_{p^n}$ to compute $g^t T_0$ and the products $UT, \ldots, U^{d'-1}T$.
3. Two rounds of $\mathbb{F}_{p^\flat}$-linear Gaussian eliminations (considering Algorithm 2 along with Improvement-1).

Let π_1 be the probability that a polynomial of degree $n_2 - d/n_1$ over $\mathbb{F}_{p^{n_1}}$ is B-smooth. The generated polynomials are not statistically independent. Heuristically however, trying out about $1/\pi_1$ polynomials of degrees $n_2 - d/n_1$, it is likely to obtain one that is B-smooth. It has been shown in [84] that the amortised cost of obtaining one polynomial to be tested for smoothness by Algorithm 2 plus Improvement-1 is $O(n_2^2)$ multiplications over $\mathbb{F}_{p^{n_1}}$ which is the same as that of the Waterloo algorithm.

A single multiplication over $\mathbb{F}_{p^{n_1}}$ consists of a polynomial multiplication followed by a reduction. Asymptotically, the cost of the reduction step is negligible in comparison to the polynomial multiplication step though in practice, reduction consumes a significant fraction of the time for the entire field multiplication. Using the schoolbook method to perform polynomial multiplication requires (n_1^2) multiplications over $\mathbb{F}_p$ and so the $O(n_2^2)$ multiplications over $\mathbb{F}_{p^{n_1}}$ has a cost of $O(n^2)$ multiplications over $\mathbb{F}_p$. Using Karatsuba's algorithm or an asymptotically faster algorithm will yield lower asymptotic costs. However, for small values of n_1, as is typically the case, the schoolbook method will be faster and it may be assumed that for Algorithm 2, the cost of generating a polynomial to be tested for smoothness is $O(n^2)$ multiplications over $\mathbb{F}_p$.

The total cost of Algorithm 2 is the one-time cost plus the cost of generating and testing about $1/\pi_1$ polynomials for smoothness. This cost is $O(n^3 \log p + (n^2 + \mathfrak{t}_1)/\pi_1)$ operations over $\mathbb{F}_p$, where $O(\mathfrak{t}_1)$ is the number of $\mathbb{F}_p$ operations required to test a polynomial of degree $n_2 - d/n_1$ over $\mathbb{F}_{p^{n_1}}$ for B-smoothness.

6.3 A New Algorithm for Initial Splitting

The setting is as in Section 6.2. Given a prime p and a composite integer n, the finite field $\mathbb{F}_{p^n}$ is represented by $(I(x), h(y))$ where $h(y)$ is an irreducible polynomial of degree n_1 over $\mathbb{F}_p$ and $I(x)$ is an irreducible polynomial of degree n_2 over $\mathbb{F}_{p^{n_1}} = \mathbb{F}_p[y]/\langle h(y)\rangle$ so that $\mathbb{F}_{p^n} = \mathbb{F}_{p^{n_1}}[x]/\langle I(x)\rangle$. Also, a generator α of $\mathbb{F}_{p^n}^*$ is available. Given a target element $T_0(x) \in \mathbb{F}_{p^n}$, the goal is to compute the logarithm of T_0 modulo ℓ where ℓ is a prime divisor of $\Phi_n(p)$.

Let d be the largest non-trivial divisor of n and $U(x) \in \mathbb{F}_{p^n}$ be such that $\{1, U(x), \ldots, U^{d-1}(x)\}$ is a polynomial basis for $\mathbb{F}_{p^d}$. Note the difference to Guillevic splitting, where $\{1, U(x), \ldots, U^{d'-1}(x)\}$ is a polynomial basis for $\mathbb{F}_{p^{d'}}$ for $d' = d/\mathfrak{d}$ and $\mathfrak{d} = \gcd(d, n_1)$. In our method, we will not require either $\mathfrak{d}$ or d'.

We also make use of Lemma 6.1. Let

$$\mathbf{a} \quad = \quad (a_0, \ldots, a_{d-1})^T \in \mathbb{F}_p^d. \tag{6.2}$$

Then

$$V \quad = \quad a_0 + a_1 U + \cdots + a_{d-1} U^{d-1} \tag{6.3}$$

is an element of $\mathbb{F}_{p^d}$. Define

$$W \quad = \quad V T_0. \tag{6.4}$$

By Lemma 6.1, the logarithms of W and T_0 are equal modulo ℓ, *i.e.*,

$$\log W \equiv \log T_0 \bmod \ell. \tag{6.5}$$

We introduce a new parameter s which is a positive integer less than n_2. The first step is to obtain $\mathbf{a}$ such that W is a monic polynomial of degree $n_2 - s - 1$ over $\mathbb{F}_{p^{n_1}}$. Next, the polynomial W is tested for B-smoothness.

There is always a possibility that it may not be smooth for the chosen B. Let π_2 be the probability that a monic polynomial of degree $n_2 - s - 1$ over $\mathbb{F}_{p^{n_1}}$ is B-smooth. By generating and testing about $1/\pi_2$ random polynomials W it is likely to obtain a polynomial which is B-smooth. We use linear algebra to generate $1/\pi_2$ polynomials over $\mathbb{F}_{p^{n_1}}$ each of degree $n_2 - s - 1$.

Given T_0, define

$$U_i \quad = \quad U^i T_0, \text{ for } i = 0, \ldots, d-1. \tag{6.6}$$

Each U_i is a polynomial of degree at most $n_2 - 1$ over $\mathbb{F}_{p^{n_1}}$.

Let $T(x) = T_0 + T_1 x + \cdots + T_{n_2-1} x^{n_2-1}$ be a polynomial of degree $n_2 - 1$, where $T_i \in \mathbb{F}_{p^{n_1}}$ for $i = 0, \ldots, n_2 - 1$. Let $T_i(y) = t_{i,0} + t_{i,1} y + \cdots + t_{i,n_1-1} y^{n_1-1}$ where $t_{i,j} \in \mathbb{F}_p$ for $i = 0, \ldots, n_2 - 1$ and $j = 0, \ldots, n_1 - 1$. Then the polynomial $T(x)$ can be encoded by the following vector of dimension n over $\mathbb{F}_p$.

$$[t_{0,0}, \ldots, t_{0,n_1-1}, \ldots, t_{n_2-1,0}, \ldots, t_{n_2-1,n_1-1}]^T. \tag{6.7}$$

Using the above encoding of polynomials to vectors, the polynomial U_i can be represented by a (column) vector $\mathbf{u}_i$ in $\mathbb{F}_p^n$.

We define an $n \times d$ matrix $\mathbf{M}$ with entries from $\mathbb{F}_p$ as follows.

$$\mathbf{M} = [\mathbf{u}_0\ \mathbf{u}_1\ \cdots\ \mathbf{u}_{d-1}] = \begin{bmatrix} \mathbf{M}_0 \\ \mathbf{M}_1 \end{bmatrix} \tag{6.8}$$

where $\mathbf{M}_0$ is an $(n_1(n_2 - s - 1)) \times d$ matrix and $\mathbf{M}_1$ is an $(n_1(s+1)) \times d$ matrix. Note that, given T_0 and U, the matrix $\mathbf{M}$ is fixed and needs to be computed only once.

The polynomial $W(x)$ can be represented as a vector $\mathbf{w} \in \mathbb{F}_p^n$. We write $\mathbf{w}$ as

$$\mathbf{w} = \begin{bmatrix} \mathbf{w}_0 \\ \mathbf{w}_1 \end{bmatrix}$$

where $\mathbf{w}_0$ is in $\mathbb{F}_p^{n_1(n_2-s-1)}$ and $\mathbf{w}_1$ is in $\mathbb{F}_p^{n_1(s+1)}$.

The relation

$$W = VT_0 = (a_0 + a_1U + \cdots + a_{d-1}U^{d-1})T_0 = a_0U_0 + a_1U_1 + \cdots + a_{d-1}U_{d-1}$$

can be written in matrix notation as follows.

$$\begin{bmatrix} \mathbf{w}_0 \\ \mathbf{w}_1 \end{bmatrix} = \mathbf{Ma} = \begin{bmatrix} \mathbf{M}_0 \\ \mathbf{M}_1 \end{bmatrix} \mathbf{a} \tag{6.9}$$

From (6.9), we obtain the following two equations.

$$\mathbf{w}_0 = \mathbf{M_0 a}; \tag{6.10}$$

$$\mathbf{w}_1 = \mathbf{M_1 a}. \tag{6.11}$$

Recall that our goal is to obtain $W(x)$ as a monic polynomial of degree $n_2 - s - 1$ over $\mathbb{F}_{p^{n_1}}$. This puts constraints on the coefficients of $W(x)$, namely, the coefficient of x^{n_2-s-1} has to be one and the coefficients of x^i for $i = n_2 - s, \ldots, n_2 - 1$ have to be zeros. These conditions define the vector $\mathbf{w}_1$ to be the following:

$$\mathbf{w}_1 = [\underbrace{1, 0, \ldots, 0, 0}_{n_1}, \underbrace{0, 0, \ldots, 0}_{n_1 s}]^T. \tag{6.12}$$

Given $\mathbf{w}_1$ and the matrix $\mathbf{M}_1$, the inhomogeneous system of equation given by (6.11) is to be solved for $\mathbf{a}$ and the resulting solution is to be substituted in (6.10) to obtain $\mathbf{w}_0$. This $\mathbf{w}_0$ provides

the polynomial

$$W(x) \quad = \quad x^{n_2-s-1} + W_0(x) \tag{6.13}$$

where $W_0(x) \in \mathbb{F}_{p^{n_1}}[x]$ is the polynomial represented by $\mathbf{w}_0$ and is of degree less than $n_2 - s - 1$. The polynomial $W(x)$ is to be tested for smoothness.

$\mathbf{M}_1$ is an $n_1(s+1) \times d$ matrix and so a necessary condition for a solution to (6.11) to exist is $d \geq n_1(s+1)$. Define

$$r \quad = \quad d - n_1(s+1). \tag{6.14}$$

Then a general solution $\mathbf{a}$ to (6.11) can be written as

$$\mathbf{a} \quad = \quad \mathbf{B}\mathbf{b} + \mathbf{c} \tag{6.15}$$

where $\mathbf{B}$ is an $d \times r$ matrix over $\mathbb{F}_p$ which is a basis for the null space of $\mathbf{M}_1$, $\mathbf{b}$ is in $\mathbb{F}_p^r$ and $\mathbf{c} \in \mathbb{F}_p^d$ is a particular solution to (6.11). Substituting the general solution given by (6.15) into (6.10) we obtain

$$\mathbf{w}_0 \quad = \quad \mathbf{M}_0\mathbf{a} = \mathbf{M}_0\left(\mathbf{B}\mathbf{b} + \mathbf{c}\right) = \mathbf{L}\mathbf{b} + \mathbf{d} \tag{6.16}$$

where

$$\mathbf{L} = \mathbf{M}_0\mathbf{B} \quad \text{and} \quad \mathbf{d} = \mathbf{M}_0\mathbf{c}. \tag{6.17}$$

$\mathbf{L}$ is a matrix of order $n_1(n_2 - s - 1) \times r$ and $\mathbf{d} \in \mathbb{F}_p^{n_1(n_2-s-1)}$. Once the system (6.11) is solved, $\mathbf{B}$ and $\mathbf{c}$ are obtained and from these it is possible to obtain $\mathbf{L}$ and $\mathbf{d}$.

Suppose r is chosen such that

$$r = d - n_1(s+1) \quad \geq \quad \lceil \log_p 1/\pi_2 \rceil. \tag{6.18}$$

Then varying $\mathbf{b}$ over all possible vectors in $\mathbb{F}_p^r$, it is possible to generate more than $1/\pi_2$ distinct polynomials $W(x) = x^{n_2-s-1} + W_0(x)$ of degrees $n_2 - s - 1$. These polynomials are not statistically independent and so theoretically the smoothness estimate does not apply to these polynomials. Heuristically however, it is likely that one of these polynomials is B-smooth. Our experiments confirm this heuristic assumption.

A necessary condition for the method to work is given by (6.18). Using Theorem 1 of [140] and using the estimate $\rho(u) \approx u^{-u}$ of the Dickman function, an estimate of the probability that

a polynomial of degree $n_2 - s - 1$ is B-smooth is as follows.

$$\pi_2 \approx \left(\frac{n_2 - s - 1}{B}\right)^{-\left(\frac{n_2-s-1}{B}\right)} \tag{6.19}$$

Combining with (6.18), we obtain the condition

$$d - n_1(s+1) \geq \left\lceil \left(\frac{n_2 - s - 1}{B}\right) \log_p \left(\frac{n_2 - s - 1}{B}\right) \right\rceil. \tag{6.20}$$

Increasing the value of s decreases the degree of $W(x)$ to be tested for smoothness and hence increases the smoothness probability π_2. On the other hand, increasing the value of s reduces the left hand side of (6.20) and (6.20) may fail to hold. So, the goal is to choose the maximum possible value of s such that (6.20) holds.

An algorithmic description of the above theory is given in Algorithm 3.

Input: An $(I(x), h(y))$-representation of $\mathbb{F}_{p^n}$; a target element $T_0(x)$; a smoothness bound B; and an s satisfying (6.20);

Output: A B-smooth polynomial $W(x)$ such that $\log W(x) \equiv \log T_0(x) \bmod \ell$ where ℓ is a divisor of $\Phi_n(p)$

2.1 Choose the maximum value of s such that (6.20) holds

2.2 Let $r = d - n_1(s+1)$

2.3 Let $U(x)$ be such that $\{1, U(x), \ldots, U^{d-1}(x)\}$ is a polynomial basis for $\mathbb{F}_{p^d}$

2.4 Compute the matrices $\mathbf{M}_0$ and $\mathbf{M}_1$ as given in (6.8)

2.5 Set $\mathbf{w}_1$ as given in (6.12)

2.6 Solve (6.11) to obtain $\mathbf{B}$ and $\mathbf{c}$ as given in (6.15)

2.7 Compute $L = \mathbf{M}_0\mathbf{B}$ and $\mathbf{d} = \mathbf{M}_0\mathbf{c}$

2.8 **for** *each* $\mathbf{b} \in \mathbb{F}_p^r$ **do**

2.9 Compute $\mathbf{w}_0 = \mathbf{L}\mathbf{b} + \mathbf{d}$ and let $W_0(x) \in \mathbb{F}_{p^{n_1}}[x]$ be the polynomial represented by $\mathbf{w}_0$

2.10 Set $W(x) = x^{n_2-s-1} + W_0(x)$

2.11 **if** $W(x)$ *is* B*-smooth* **then**

2.12 break

2.13 **end**

2.14 **end**

2.15 **return** $W(x)$.

Algorithm 3: The new algorithm for initial splitting.

Remarks:

1. A difference between Algorithm 2 and Algorithm 3 pertains to the target. In Algorithm 3, the original target T_0 remains unchanged, whereas in Algorithm 2, for every d' candidate polynomials a fresh target $T_t = g^t T_0$ is computed.

2. In her paper [84], Guillevic had also proposed an initial splitting algorithm for the number field sieve (NFS) algorithm in the large characteristic case. Our method of generating polynomials does not help in improving the efficiency of Guillevic splitting for the NFS algorithm.

6.3.1 Implementation Issues

Since p is small, it is possible to speed up the computation of $\mathbf{Lb} + \mathbf{c}$ at the cost of extra storage. For $k = 1, \ldots, p-1$, let $\mathbf{L}^{(k)} = kL$ and denote by $\mathbf{L}^{(k)}_{*,j}$ the j-th column of $\mathbf{L}^{(k)}$. Suppose the matrices $\mathbf{L}^{(1)}, \ldots, \mathbf{L}^{(p-1)}$ have been computed and stored. Suppose $\mathbf{b} = (b_1, \ldots, b_r)$. Then $\mathbf{Lb} + \mathbf{c}$ can be computed as follows.

$\mathsf{sum} \leftarrow \mathbf{c}$;
for $j = 1, \ldots, r$ do
  if $b_j > 0$, then $\mathsf{sum} \leftarrow \mathsf{sum} + \mathbf{L}^{(b_j)}_{*,j}$
end for;
return sum.

This method of computation avoids all the multiplications over $\mathbb{F}_p$ during the generation of the polynomial $W(x)$.

Parallelism: In Algorithm 3, after the matrix $\mathbf{L}$ and the vector $\mathbf{c}$ have been generated, the generation of the polynomials and testing them for smoothness can be completely parallelised. It is possible to allocate non-intersecting subsets of $\mathbb{F}_p^r$ to different processes. Each process independently uses the vectors in its alloted subset to generate polynomials and test them for smoothness. There is no need for any coordination between the processes.

Guillevic splitting in Algorithm 2 also supports parallelism though of a somewhat restricted kind. The completion of the doubly reduced row echelon form provides a total of d' polynomials to be tested for smoothness by independent processes. The testing of these polynomials can be done in parallel. However, once these polynomials have been checked, the processes have to halt until the next batch of d' polynomials have been generated. Alternatively, there can be a process which successively generates batches of d' polynomials and feeds them to other processes to be tested for smoothness. Neither of these options is as simple as the parallelism that can be obtained from Algorithm 3.

6.3.2 Degrees of Polynomials Generated by Algorithm 3

Each iteration generates a polynomial $W(x)$ of degree $n_2 - s - 1$. As mentioned above, s is a positive integer less than n_2 which is to be chosen as the maximum value satisfying (6.20). We determine the value of s and hence the degrees of $W(x)$ for the two examples considered in [84].

Example-1: For this example, $p = 3$, $n_1 = 6$, $n_2 = 509$ and so $d = 3 \cdot 509$. For $26 \leq B \leq 32$, the maximum value of s satisfying (6.20) is $s = 250$. So, the degrees of the corresponding $W(x)$'s are 258.

Example-2: For this example, $p = 3$, $n_1 = 5$ and $n_2 = 479$ and so $d = n_2 = 479$. For $42 \leq B \leq 50$, the maximum value of s satisfying (6.20) is $s = 91$. So, the degrees of the corresponding $W(x)$'s are 387.

Using Algorithm 2, the degrees of the generated polynomials for Example-1 and Example-2 are 254 and 383 respectively. So, compared to Algorithm 2, the polynomials generated by Algorithm 3 have slightly larger degrees. This has two consequences.

1. The smoothness probability π_2 for Algorithm 3 is slightly smaller than the smoothness probability π_1 for Algorithm 2. For Example-1, π_1/π_2 is in the range $[1.33, 1.46]$ for $32 \geq B \geq 26$ while for Example-2, π_1/π_2 is in the range $[1.20, 1.26]$ for $50 \geq B \geq 42$.

2. The cost of smoothness checking t_2 in Algorithm 3 is slightly greater than the cost of smoothness checking t_1 in Algorithm 2. For the degrees considered, it is reasonable to assume $\mathrm{t}_1 \approx \mathrm{t}_2$.

We note that Algorithm 2 combined with both Improvement-1 and Improvement-2 result in polynomials whose degrees are one or two more than those obtained from Algorithm 2 combined only with Improvement-1. So, the degrees of polynomials generated by Algorithm 2 plus Improvement-1 and Improvement-2 are even closer to the degrees of polynomials generated by Algorithm 3.

6.3.3 Cost of Algorithm 3

The one-time cost of Algorithm 3 consists of the following components.

1. Computation of $U(x) = \alpha^{(p^n-1)/(p^d-1)}$. As in the case of Algorithm 2, this cost is $O(n^3 \log p)$. Also, as in the case of Algorithm 2, it is not required to compute the basis $\{1, U(x), \ldots, U^{d-1}(x)\}$.

2. Computation of $\mathbf{M}_0$ and $\mathbf{M}_1$ requires the elements $\{T_0, UT_0, \ldots, U^{d-1}T_0\}$. This requires a total of $d-1$ multiplications in $\mathbb{F}_{p^n}$ which we estimate as $O((d-1)n^2)$ operations over $\mathbb{F}_p$.

3. Solving (6.11) to obtain $\mathbf{B}$ and $\mathbf{c}$. Since $\mathbf{M}_1$ is an $n_1(s+1) \times d$ matrix over $\mathbb{F}_p$, the cost for this step is $O(n_1(s+1)d^2)$ operations over $\mathbb{F}_p$.

4. Computing $\mathbf{L} = \mathbf{M}_0\mathbf{B}$ and $\mathbf{d} = \mathbf{M}_0\mathbf{c}$ where $\mathbf{M}_0$ is an $n_1(n_2 - s - 1) \times d$ matrix over $\mathbb{F}_p$, $\mathbf{B}$ is a $d \times r$ matrix over $\mathbb{F}_p$ and $\mathbf{c}$ is in $\mathbb{F}_p^d$. The cost for the matrix multiplication $\mathbf{M}_0\mathbf{B}$ is $O(n_1(n_2 - s - 1)dr)$ operations over $\mathbb{F}_p$ and the cost for the matrix-vector multiplication $\mathbf{M}_0\mathbf{c}$ is $O(n_1(n_2 - s - 1)d)$ operations over $\mathbb{F}_p$.

The total one-time cost is $O(n^3 \log p + (d-1)n^2 + n_1(s+1)d^2 + n_1(n_2 - s - 1)dr) = O(n^3)$ operations over $\mathbb{F}_p$. In practice, the one-time computation is negligible in comparison to the time for generating and testing polynomials for smoothness.

The cost of generating a polynomial to be tested for smoothness is the cost of computing the matrix-vector multiplication $\mathbf{Lb}$. Since $\mathbf{L}$ is an $n_1(n_2 - s - 1) \times r$ matrix over $\mathbb{F}_p$ and $\mathbf{b} \in \mathbb{F}_p^r$, this cost is $O(n_1(n_2 - s - 1)r)$ operations over $\mathbb{F}_p$. Noting that $n_1 n_2 = n$ and $r \approx \log_p(1/\pi_2)$, the cost of generating each polynomial is $O((n - n_1(s+1))r) = O((n - n_1(s+1)) \log_p(1/\pi_2))$ $\mathbb{F}_p$ operations. The value of s is less than n_2, and so, the cost of generating a polynomial is $O(n \log_p(1/\pi_2))$ $\mathbb{F}_p$-operations.

The total cost of Algorithm 3 is the cost of one-time computation plus the cost for generating and testing the polynomials for B-smoothness. This cost is $O(n^3 \log p + (\mathfrak{t}_2 - n \log \pi_2)/\pi_2)$ operations over $\mathbb{F}_p$, where $O(\mathfrak{t}_2)$ is the number of $\mathbb{F}_p$ operations required to test a polynomial of degree $n_2 - s - 1$ over $\mathbb{F}_{p^{n_1}}$ for B-smoothness.

In contrast, the total cost of Algorithm 2 is $O(n^3 \log p + (\mathfrak{t}_1 + n^2)/\pi_2)$ operations over $\mathbb{F}_p$. Based on the discussion in Section 6.3.2, we may take π_1 and π_2 to be approximately equal and denote by π this common smoothness probability. Similarly, we may take $\mathfrak{t}_1$ and $\mathfrak{t}_2$ to be approximately equal and denote by $\mathfrak{t}$ to be the time for smoothness checking in both the algorithms. Then the total time for Algorithms 2 and 3 are respectively $O(n^3 \log p + (\mathfrak{t} + n^2)/\pi)$ and $O(n^3 \log p + (\mathfrak{t} - n \log \pi)/\pi)$ operations over $\mathbb{F}_p$. The main cost for Algorithm 2 is $O((\mathfrak{t} + n^2)/\pi)$ while for Algorithm 3 it is $O((\mathfrak{t} - n \log \pi)/\pi)$.

From [60], for a degree δ polynomial over $\mathbb{F}_{p^{n_1}}$, the costs of square-free factorisation, distinct degree factorisation and equal degree factorisation are respectively $O(\delta^2)$, $O(\delta^3 \log p^{n_1})$ and $O(\delta^2 \log p^{n_1})$. So, the cost $\mathfrak{t}$ of smoothness checking is substantial. In comparison to Algorithm 2, the main advantage of Algorithm 3 is that it makes the cost of generating a polynomial negligible in comparison to the cost of testing the polynomial for smoothness.

6.4 Computational Results

To demonstrate that Algorithm 3 works, we made a basic Magma implementation for Example-1 mentioned in Section 6.3.2, *i.e.*, $p = 3$, $n_1 = 6$ and $n_2 = 509$. The field $\mathbb{F}_{p^n}$ is represented using $(I(x), h(y))$. The polynomials $h(y)$ and $I(x)$ and the generator $x + y^2$ are given in [84]. Further, we also used the target T_0 that was used in [84]. The largest prime divisor of n is $d = 3 \cdot 509$.

We ran Algorithm 3 for two values of B, namely $B = 28$ and $B = 30$. In both cases, the value of s satisfying (6.20) is 250 and so the degrees of the generated polynomials are $n_2 - s - 1 = 258$. Since $d = 3 \cdot 509$, $n_1 = 6$ and $s = 250$, the value of r from (6.14) is 21.

Given α, $U(x)$ is uniquely defined. Given $T_0(x)$ and $U(x)$, the matrices $\mathbf{M}_0$ and $\mathbf{M}_1$ are completely defined. Since $\mathbf{w}_1$ is fixed by (6.12), given $\mathbf{M}_1$, the matrix $\mathbf{B}$ and the particular solution $\mathbf{c}$ to (6.11) are completely defined. Further, given $\mathbf{B}$ and $\mathbf{c}$, the matrix $\mathbf{L}$ and the vector $\mathbf{d}$ are also completely defined. So, given α and $T_0(x)$, the matrix $\mathbf{L}$ and the vector $\mathbf{d}$ are defined. Different values of $\mathbf{b}$ generates different values of $\mathbf{w}_0$ (equivalently, different values of $W_0(x)$) and so different values of $W(x)$. From (6.5), we have $\log W \equiv \log T_0 \bmod \ell$.

Below we provide the obtained values of $\mathbf{b}$ such that the corresponding $W(x)$'s are B-smooth for $B = 28$ and $B = 30$. Along with the b's we also provide the smoothness probabilities.

$B = 28$: $\pi_2 \approx 2^{-29.5}$, $\mathbf{b} = (0, 1, 1, 1, 1, 0, 2, 1, 2, 1, 1, 1, 2, 1, 2, 1, 1, 0, 0, 2, 0)$.

$B = 30$: $\pi_2 \approx 2^{-26.7}$, $\mathbf{b} = (0, 0, 1, 2, 2, 1, 1, 0, 2, 0, 0, 0, 2, 2, 2, 0, 0, 2, 0, 0, 0)$.

In both the cases, the degrees of the generated polynomials are 258. So, the time for one-time computation and the average times for generating a polynomial and smoothness checking are also the same. The one-time computation took less than 2 hours.

To obtain an estimate of the times required per iteration, we averaged over 1000 iterations. The average time to generate a polynomial is about 10^{-5} seconds while the average time to check the polynomial for smoothness is about 0.4 seconds. As mentioned earlier, the advantage of the new method is that the time for generating a polynomial is negligible in comparison to the time for smoothness checking.

For the actual computations of $\mathbf{b}$ for $B = 28$ and $B = 30$, the iterative part of Algorithm 3 was parallelised as mentioned in Section 6.3.1. The subspace $\mathbb{F}_3^r$ was divided into disjoint subspaces to be searched by 320 parallel processes running on four servers having 100 cores each. The computation for $B = 30$ took less than a day while the computation for $B = 28$ took about 10 days. We did not have exclusive access to the servers during the execution of the programs. The server was loaded with long running R and Matlab programs by other users. Due to this, the exact times for the completion of our programs are not informative and so we do not report these times.

Theoretical improvement of initial splitting when our algorithm is used has been shown. In addition, it is completely parallelizable. The overall speed-up over Guillevic's splitting has not been determined as Guillevic's algorithm though supports parallelism, but cannot be made completely parallel.

6.5 Notes on Computation

We have implemented Algorithm 3 in Magma. The description of the algorithm and the theoretical explanation provided in Section 6.3 are mostly in terms of vectors and matrices. Magma, on the other hand, provides good support for polynomial arithmetic. So, we performed some of the computations using polynomials. While this makes the computation more convenient for Magma, it does not change the cost analysis provided in Section 6.3.3. Below we mention the portion of the computation that has been done using polynomials.

6.5.1 Working with Polynomials

The target T_0, the basis $\{1, U, \ldots, U^{d-1}\}$ of $\mathbb{F}_{p^d}$ and $U_i = U^i T_0$, for $i = 0, \ldots, d-1$ are polynomials of degrees at most $n_2 - 1$ over $\mathbb{F}_{p^{n_1}}$. It has been mentioned that each of the polynomials U_i is encoded by a vector $\mathbf{u}_i$ of dimension n over $\mathbb{F}_p$. This leads to the matrix $\mathbf{M}$ which is written as the stacking of two matrices $\mathbf{M}_0$ and $\mathbf{M}_1$ as shown in (6.8).

For the computation, we indeed formed the matrix $\mathbf{M}_1$. On the other hand, we did not actually construct the matrix $\mathbf{M}_0$. Instead the matrix $\mathbf{M}_0$ was represented using the set of polynomials $\{P_0(x), \ldots, P_{d-1}(x)\}$, where $P_i(x) = U_i(x) \bmod x^{n_2-s-1}$ for $i = 0, \ldots, d-1$.

The matrix $\mathbf{M}_1$ and the vector $\mathbf{w}_1$ as given in (6.12) was used as inputs to the linear algebra solver, providing the matrix $\mathbf{B}$ and the vector $\mathbf{c}$ as outputs.

At this point, it is required to obtain the matrix $\mathbf{L} = \mathbf{M}_0\mathbf{B}$ and the vector $\mathbf{d} = \mathbf{M}_0\mathbf{c}$. Since the matrix $\mathbf{M}_0$ was represented as a set of polynomials, we in fact obtain $\mathbf{L}$ as a set of polynomials and the vector $\mathbf{d}$ as a single polynomial. In more details, let $\mathbf{B} = [[\beta_{i,j}]]$, $0 \le i \le d-1$ and $0 \le j \le r-1$ and $\mathbf{c} = [c_0, \ldots, c_{d-1}]^T$, where $\beta_{i,j}, c_0, \ldots, c_{d-1} \in \mathbb{F}_p$. The matrix $\mathbf{L}$ was represented using a set of polynomials $\{R_0(x), \ldots, R_r(x)\}$, where $R_j(x) = \beta_{0,j}P_0(x) + \beta_{1,j}P_1(x) + \cdots + \beta_{d-1,j}P_{d-1}(x)$ for $j = 0, \ldots, r-1$. The vector $\mathbf{d}$ was represented using the polynomial $D(x) = c_0P_0(x) + c_1P_1(x) + \cdots + c_{d-1}P_{d-1}(x)$.

Given the vector $\mathbf{b}$, the vector $\mathbf{w}_0$ is obtained as $\mathbf{L}\mathbf{b} + \mathbf{d}$. This $\mathbf{w}_0$ represents the polynomial W_0. Since $\mathbf{L}$ and $\mathbf{d}$ are represented as polynomials, the polynomial W_0 is directly computed as $W_0(x) = b_0R_0(x) + b_1R_1(x) + \cdots + b_{r-1}R_{r-1}(x) + D(x)$, where $\mathbf{b} = [b_0, \ldots, b_{r-1}]^T$. Finally, the polynomial $W(x) = x^{n_2-s-1} + W_0(x)$ is tested for smoothness.

6.5.2 Verifying Solutions

Suppose $\mathbf{b}$ is a vector which leads to $W_0(x)$ such that $W(x)$ is B-smooth. Two examples of $\mathbf{b}$ are given in Section 6.4. We briefly mention how the correctness of these solutions can be verified.

Recall that the matrix $\mathbf{B}$ and the vector $\mathbf{c}$ have been obtained as the output of the linear algebra solver. So, given $\mathbf{b}$, the vector $\mathbf{a} = \mathbf{B}\mathbf{b} + \mathbf{c}$ can be obtained. From (6.3), given $\mathbf{a} =$

$[a_0, \ldots, a_{d-1}]^T$, we obtain $V = a_0 + a_1 U + \cdots + a_{d-1} U^{d-1}$. Once V is obtained, we can multiply it to T_0 to obtain $W = VT_0$. This W can then be verified to be B-smooth as required.

So, given the basis $\{1, U, \ldots, U^{d-1}\}$, the matrix $\mathbf{B}$ and the vector $\mathbf{c}$, it is possible to verify that a solution $\mathbf{b}$ is indeed correct.

We provide the Magma code for verifying the correctness of the vectors $\mathbf{b}$ given in Section 6.4 at the following link.

https://github.com/Madhurima11/faster-initial-splitting.git.

6.6 Conclusion

For small characteristic, composite extension degree fields, we have shown that in the initial splitting step, the cost of generating polynomials to be tested for smoothness can be brought down to $O(n \log(1/\pi))$ operations in $\mathbb{F}_p$ from the cost $O(n_2^2)$ multiplications in $\mathbb{F}_{p^{n_1}}$ that is required by the Guillevic splitting algorithm [84]. This improvement should help in the computation of future record discrete logarithm computations over such fields. An estimate of the expected speed-up will depend upon the target field.

Part II

Class Groups of Number Fields

Chapter 7

Class Group Computation

Computation of the class group of an order of the ring of integers of a number field is a basic problem in computational algebraic number theory. It is also of cryptographic interest. Despite its importance, it remains a very difficult problem. The complexity is expressed in terms of the discriminant of the number field. The initial efforts were dedicated to quadratic number fields. Presently, several subexponential algorithms are available for various degrees of number fields.
To describe the attempts made to tackle this problem, at first we need to introduce certain mathematical terms associated with the problem.

7.1 Preliminaries

Let $\mathbb{Q}$, $\mathbb{R}$ and $\mathbb{C}$ respectively denote the fields of rational, real and complex numbers. The ring of integers will be denoted by $\mathbb{Z}$. In the following, a brief overview of some relevant facts regarding the problem of computing the class group is given. Further details can be found in [44, 166, 9].

7.1.1 Basic Definitions

Definition 7.1. *Number Field:* An algebraic number field or simply a number field $\mathcal{K}$ is a subfield of the field of all complex numbers such that it contains $\mathbb{Q}$ as a subset and the dimension of $\mathcal{K}$ as a vector space over $\mathbb{Q}$ is finite.

An algebraic number field is thus a finite extension of $\mathbb{Q}$. It can be expressed in a variety of ways [45, 121, 144]. Mostly, these modes of expression can be transformed into another in polynomial time.
Let n be the extension degree of the number field $\mathcal{K}$, *i.e.*, $n = [\mathcal{K} : \mathbb{Q}]$. Then $\mathcal{K}$ is expressed as

$$\mathcal{K} \cong \mathbb{Q}[x]/\langle T \rangle \tag{7.1}$$

for some monic irreducible polynomial T over $\mathbb{Z}$ of degree n.
Given any number field $\mathcal{K}$ of extension degree n, by primitive element theorem, it is expressible as $\mathcal{K} = \mathbb{Q}(\theta)$ for some $\theta \in \mathcal{K}$. Such an element θ is called a *primitive element* of the number field $\mathcal{K}$.

Definition 7.2. *Defining Polynomial:* Let $\mathcal{K}$ be a number field with extension degree n. Any polynomial T which is the minimal polynomial of some primitive element θ of $\mathcal{K}$ is called a defining polynomial of the number field.

All defining polynomials are monic, irreducible polynomials over $\mathbb{Q}$ of degree equal to the degree of extension of the number field. For each of the polynomials T, an expression of the form 7.1 holds.

As $\mathbb{C}$ is algebraically closed and T is irreducible of degree n, there are exactly n roots of T. Since complex roots occur in conjugate pairs, the number of complex roots is even. Let $r_1, 2r_2$ be the number of real and complex roots respectively. Then the condition $r_1 + 2r_2 = n$ holds. The tuple (r_1, r_2) is called the *signature* of the number field. Every root ξ_i, $i = 1, 2, \ldots, n$, induces a homomorphism from $\mathbb{Q}[X] \to \mathbb{C}$, given by $X \mapsto \xi_i$. Each such map defines an embedding σ_i, $i = 1, 2, \ldots, n$. For $a_0, a_1, \ldots, a_{n-1} \in \mathbb{Q}$,

$$\sigma_i(a_0 + a_1X + \ldots + a_{n-1}X^{n-1}) \mapsto a_0 + a_1\xi_i + \ldots + a_{n-1}{\xi_i}^{n-1}.$$

An embedding is called a *real embedding* if it maps entirely into $\mathbb{R}$, otherwise it is called a *complex embedding*. The embeddings σ_i can be renumbered such that the real embeddings are $\sigma_1, \ldots, \sigma_{r_1}$ while the complex embeddings are $\sigma_{r_1+1}, \ldots, \sigma_n$ with σ_{r_1+j} paired with its complex conjugate $\sigma_{r_1+r_2+j}, j = 1, 2, \ldots, r_2$.
The *canonical embedding* is defined as:

$$\sigma : \mathcal{K} \to \mathbb{R}^{r_1} \times \mathbb{C}^{r_2}$$
$$\sigma(a) = (\sigma_1(a), \ldots, \sigma_n(a)).$$

Definition 7.3. *Norm and Trace:* Let $\mathcal{K}$ be a number field of degree n with n distinct embeddings σ_i defined from it to $\mathbb{C}$. The norm and the trace functions denoted by $\mathcal{N}_{\mathcal{K}/\mathbb{Q}}$ and $\mathcal{T}_{\mathcal{K}/\mathbb{Q}}$ are defined respectively by:

$$\mathcal{N}_{\mathcal{K}/\mathbb{Q}}(\alpha) = \prod_{i=1}^{n} \sigma_i(\alpha)$$

$$\mathcal{T}_{\mathcal{K}/\mathbb{Q}}(\alpha) = \sum_{i=1}^{n} \sigma_i(\alpha).$$

Due to the homomorphic nature of σ_i, the norm and trace functions are multiplicative and additive functions respectively on $\mathcal{K}$.
The norm can be computed conveniently ([44], Proposition 4.3.4) as

$$\mathcal{N}_{\mathcal{K}/\mathbb{Q}}(\alpha) = \frac{Res(T(X), A(X))}{d^n} \tag{7.2}$$

where $\alpha = \frac{1}{d}\big(\sum_{i=0}^{n} a_i\theta^i\big), A(X) = \sum_{i=0}^{n} a_i X^i$ and Res denotes the resultant of the two polynomials.

An element $\alpha \in \mathcal{K}$ is said to be an *integral element* of $\mathcal{K}$ if it is a root of a monic polynomial with coefficients in $\mathbb{Z}$.

Definition 7.4. *Ring of Integers:* The ring of integers of a number field $\mathcal{K}$ is the set of all integral elements of $\mathcal{K}$. It is generally denoted by $\mathcal{O}_\mathcal{K}$.

As sum and product of two integral elements is also integral so $\mathcal{O}_\mathcal{K}$ is a ring. It is a lattice in $\mathcal{K}$ ([166], Proposition 2.4.5). The dimension of this $\mathbb{Z}$-module $\mathcal{O}_\mathcal{K}$ is equal to the extension degree n of $\mathcal{K}$. A $\mathbb{Z}$-basis of $\mathcal{O}_\mathcal{K}$ is called an *integral basis* of $\mathcal{K}$. The field of fractions of $\mathcal{O}_\mathcal{K}$ is $\mathcal{K}$ itself ([166], Lemma 2.3.19). For any $\alpha \in \mathcal{K}$, there is an integer a, such that $a\alpha \in \mathcal{O}_\mathcal{K}$.

Definition 7.5. *Discriminant:* Let $b_1, b_2, \ldots, b_n$ be an integral basis of $\mathcal{K}$. The discriminant $\Delta_\mathcal{K}$ of the number field $\mathcal{K}$ is defined as

$$\Delta_\mathcal{K} = \big(det(\sigma_i(b_j))\big)^2 = det(\mathcal{T}_{\mathcal{K}/\mathbb{Q}}(b_i b_j)) \tag{7.3}$$

where each of the symbols used have their conventional meanings as defined before.

It can be shown ([166], Section 6.2) that this value is independent of the choice of the integral basis. It is the same as the determinant of the canonical embedding of $\mathcal{K}$ and the discriminant of the order $\mathcal{O}_\mathcal{K}$.

7.1.2 Ideals

Definition 7.6. *Ideal:* An ideal $\mathbf{a}$ of $\mathcal{O}_\mathcal{K}$ is a sub-$\mathbb{Z}$-module of $\mathcal{O}_\mathcal{K}$ such that for all $x \in \mathcal{O}_\mathcal{K}$ and $a \in \mathbf{a}$, $xa \in \mathbf{a}$. It also called an integral ideal.

Definition 7.7. *Fractional Ideal:* A fractional ideal $\mathbf{a}$ of $\mathcal{O}_\mathcal{K}$ is an $\mathcal{O}_\mathcal{K}$-submodule of $\mathcal{K}$ with some element $d \in \mathcal{O}_\mathcal{K}$ such that $d\mathbf{a} \subseteq \mathcal{O}_\mathcal{K}$.

The element d is called the *denominator* of $\mathcal{O}_\mathcal{K}$. It is easy to show that $d\mathfrak{a}$ is an ideal of $\mathcal{O}_\mathcal{K}$. Any ordinary ideal of $\mathcal{O}_\mathcal{K}$ is also a fractional ideal (considering $d = 1$) and is referred to as an

integral ideal.
The intersection, sum of two fractional ideals is also fractional [Lemma 3.2.5, [9]].

The product of two ideals $\mathbf{a_1}, \mathbf{a_2}$ is defined as the set of finite sums of products of elements taken from these ideals, *i.e.*, $\mathbf{a_1 a_2} = \{\sum_{i<m}(a_{1_i})(a_{2_i}) \mid a_{k_i} \in \mathbf{a_k};\ k = 1, 2;\ m \in \mathbb{N}\}$. The product of a finite number of ideals whether integral or fractional is also defined in the same way. It is easy to see that the product defined in this fashion is commutative.
The inverse of a fractional ideal $\mathbf{a}$ is another fractional ideal by Proposition 3.2.7 of [9] and is given by $\{\alpha \in \mathcal{K} : \alpha\mathbf{a} \subseteq \mathcal{O}_\mathcal{K}\}$. Any fractional ideal $\mathbf{a}$ can be written as the ratio $\frac{\mathbf{b}}{d\mathcal{O}_\mathcal{K}}$, where $\mathbf{b}$ is an integral ideal and d is the denominator of the ideal $\mathbf{a}$. This means any fractional ideal is expressible as the ratio of two integral ideals.
Let $\mathcal{I}_{\mathcal{O}_\mathcal{K}}$ be the set of all fractional ideals of $\mathcal{O}_\mathcal{K}$. Then from the previous paragraph, $\mathcal{I}_{\mathcal{O}_\mathcal{K}}$ forms a commutative group multiplicatively with the identity element equal to the ideal $\mathcal{O}_\mathcal{K}$.

Definition 7.8. *Principal Ideal:* An ideal $\mathbf{a}$ of $\mathcal{O}_\mathcal{K}$ is said to be a principal ideal if $\mathbf{a} = \alpha\mathcal{O}_\mathcal{K} = \langle\alpha\rangle$ for some element α in $\mathcal{O}_\mathcal{K}$.

Suppose $\mathcal{P}_{\mathcal{O}_\mathcal{K}}$ is the set of all principal fractional ideals of $\mathcal{O}_\mathcal{K}$. Then $\mathcal{P}_{\mathcal{O}_\mathcal{K}}$ is a subgroup of $\mathcal{I}_{\mathcal{O}_\mathcal{K}}$. These groups being commutative, the idea of quotient group can be perceived.

Definition 7.9. *Ideal Class Group:* The ideal class group of $\mathcal{O}_\mathcal{K}$ is defined as the quotient group $\mathcal{I}_{\mathcal{O}_\mathcal{K}}/\mathcal{P}_{\mathcal{O}_\mathcal{K}}$ and is denoted by $Cl_{\mathcal{O}_\mathcal{K}}$.

The ideal class group $Cl_{\mathcal{O}_\mathcal{K}}$ is commutative due to the abelian property of $\mathcal{O}_\mathcal{K}$. The cardinality of this group is finite (Corollary 5.3.6, [9]) and is said to be the *class number* of $\mathcal{K}$.
Let $\mathbf{a}$ be an integral ideal of $\mathcal{O}_\mathcal{K}$. It can be shown that the quotient group $\mathcal{O}_\mathcal{K}/\mathbf{a}$ is finite.

Definition 7.10. *Norm:* The norm of an integral ideal is given by the cardinality of the quotient group $\mathcal{O}_\mathcal{K}/\mathbf{a}$. The norm is denoted by $\mathcal{N}(\mathbf{a})$.

In case of a principal ideal, $\mathbf{a} = \langle\alpha\rangle = \alpha\mathcal{O}_\mathcal{K}$, the norm is same as the absolute value of the norm of the generating element, *i.e.*, $\mathcal{N}(\mathbf{a}) = |\mathcal{N}(\alpha)|$ (Proposition 4.2.6, [9]). This notion of norm can be extended in case of fractional ideals as well. For a fractional ideal $\mathbf{a} = \mathbf{b}/\mathbf{c}$, $\mathcal{N}(\mathbf{a}) = \mathcal{N}(\mathbf{b})/\mathcal{N}(\mathbf{c})$ ([166], Proposition 6.3.4). This definition is not dependent on the way in which any fractional ideal is defined.

Any nonzero fractional ideal of $\mathcal{O}_\mathcal{K}$ is uniquely expressible as a product of prime ideals ([166], Theorem 3.3.1). If $\mathbf{a}$ is a fractional ideal, then there are unique prime ideals $\mathbf{p}_1, \mathbf{p}_2, \ldots, \mathbf{p}_k$ and non-zero integers $e_1, e_2, \ldots, e_k$ such that $\mathbf{a} = \prod_{i=1}^{k} {\mathbf{p}_i}^{e_i}$.

The class group computation is also associated with the computation of units and regulator of the number field. They are explained in the following definitions.

Definition 7.11. *Unit:* Units in a number field are all those algebraic integers whose inverse is also an algebraic integer.

It is easy to see that, for a unit, the absolute value of the norm is one. The converse also holds true ([166], Proposition 8.1.4).
Due to the multiplicativity of the norm, the set of all units form a group. This multiplicative group of all units in $\mathcal{K}$ is denoted by $\mathcal{U}_{\mathcal{K}}$. The roots of unity also form a multiplicative group. This subgroup of $\mathcal{U}_{\mathcal{K}}$ is called the subgroup of the roots of unity in $\mathcal{K}$ and is denoted by $\mu_{\mathcal{K}}$. The groups $\mathcal{U}_{\mathcal{K}}$ and $\mu_{\mathcal{K}}$ can be connected by *Dirichlet's Unit Theorem.*

Theorem 7.12. *Dirichlet's Unit Theorem: Let (r_1, r_2) be the signature of the number field $\mathcal{K}$. The group $\mathcal{U}_{\mathcal{K}}$ of all units of $\mathcal{K}$ is a finitely generated abelian group of rank $r = r_1 + r_2 - 1$. Further, $\mu_{\mathcal{K}}$ is a finite cyclic group and we have the following isomorphism.*

$$\mathcal{U}_{\mathcal{K}} \backsimeq \mu_{\mathcal{K}} \times \mathbb{Z}^r. \tag{7.4}$$

By Theorem 7.12, there exists units $u_1, u_2, \ldots, u_r$ such that every element $u \in \mathcal{U}_{\mathcal{K}}$ can be written as $u = \zeta {u_1}^{n_1} {u_2}^{n_2} \ldots {u_r}^{n_r}$ where $n_1, n_2, \ldots n_r \in \mathbb{Z}$ and ζ is a root of unity. Any such r-tuple $(u_1, u_2, \ldots, u_r)$ is called a *system of fundamental units* of $\mathcal{K}$.
The notion of embedding has already been dealt with in this chapter. Let us now define the *logarithmic embedding* which utilises embeddings. It leads to the concept of *regulator* which is a measure of density of the unit group in the logarithmic space.

Definition 7.13. *Logarithmic embedding:* The logarithmic embedding is defined by the map *Log*.

$$Log : \mathcal{K}^* \rightarrow \mathbb{R}^{r_1+r_2} \tag{7.5}$$

$$x \mapsto \big(log|\sigma_1(x)|, log|\sigma_2(x)|, \ldots, log|\sigma_{r_1}(x)|, 2log|\sigma_{r_1+1}(x)|, \ldots, 2log|\sigma_{r_1+r_2}(x)|\big). \tag{7.6}$$

This idea of logarithmic embedding gives rise to a lattice. The following theorem which is Theorem 4.9.7 of [44] is a restatement of Dirichlet's Unit Theorem, but conveys some important ideas.

Theorem 7.14. *The image of the group $\mathcal{U}_{\mathcal{K}}$ under the logarithmic embedding is a lattice. It lies in the hyperplane $\sum_{i=1}^{r_1+r_2} x_i = 0$ in $\mathbb{R}^{r_1+r_2}$ and is of rank $r = r_1 + r_2 - 1$. It is called the Log-unit lattice. Also, the kernel of this embedding is equal to the group $\mu_{\mathcal{K}}$.*

Definition 7.15. *Regulator:* The determinant of the log-unit lattice is called the regulator of $\mathcal{K}$ and is denoted by $Reg_{\mathcal{K}}$.

$Reg_{\mathcal{K}}$ being equal to the volume of the lattice, is an invariant for any number field.
$Reg_{\mathcal{K}}$ is also defined as the absolute value of the determinant of an arbitrary minor of size r of

the $r \times (r+1)$ matrix with (i,j)th entry as $\big(log\|\sigma_j(u_i)\|\big)$, $1 \leq i \leq r$ and $1 \leq j \leq (r+1)$. The norm $\|\|$ is:

$$\|\sigma(x)\| = |\sigma(x)| \text{ if } \sigma \text{ is real,}$$
$$\|\sigma(x)\| = |\sigma(x)|^2 \text{ if } \sigma \text{ is complex.}$$

Calculation of a close approximation of the regulator is associated with class group computation. It is described in Section 7.3.5 of this Chapter. Algorithms for computation of the torsion subgroup of $\mathcal{U}_{\mathcal{K}}$ can be found in ([44], Section 4.9.2).

7.2 Related Problems and Cryptographic Applications

Several mathematical perspectives as well as cryptographic aspects have already been listed in Sections 1.5.1 and 1.5.2 of this thesis. We provide some more in this section.

7.2.1 Related Problems

The set of relations obtained for computing the class group are also used for computing the regulator and a set of generators of the unit group. In fact, the regulator and the class number are required to be computed together. The techniques for decomposing an ideal over a factor basis are also used to obtain algorithms for the principal ideal problem.

7.2.2 Cryptographic Applications

As discussed before, there is no polynomial-time algorithm to compute the class group and the best-known algorithms have sub-exponential complexity. So, for a number field with a sufficiently large discriminant, the order of the class group can be considered to be unknown. Such a hidden order group forms the basis for several cryptographic primitives [26, 40]. Concrete suggestions for instantiating these primitives have been made using class groups of imaginary quadratic fields. In principle, though, the class group of a general number field can also be used. The security versus efficiency question of using class groups of general number fields versus those of imaginary quadratic fields remains to be studied. Progress in algorithms for computing class groups influences the choice of number fields over which the relevant cryptographic primitives can be securely implemented.

7.3 General Method for Class Group Computation

The best asymptotic algorithms to compute class groups follow the general framework of the index calculus method. The steps are broadly classified as:

1. *Choice of Factor Base:* Fixing a factor base consisting of elements *small* in certain sense, but enough to generate the class group.
2. *Relation Collection Phase:* Collecting relations among the factor basis elements.
3. *Linear Algebra Step:* Solving the set of equations obtained as relations. Thereby the logarithms of the factor base elements are obtained.

After the completion of the steps of the index calculus method, the class group is computed from the results obtained in the linear algebra step. A verification step is also kept to ensure that the computed class group is valid.
These steps may be further detailed as below.

7.3.1 Selection of Factor Base

The factor base is defined as the set of all prime ideals in $\mathcal{O}_\mathcal{K}$ having norm below some fixed bound B. This integer B is called the *smoothness bound*. Let $\mathcal{B}$ denote the factor base and N denote the number of such prime ideals. Then,

$$\mathcal{B} = \{\mathfrak{p}_1, \mathfrak{p}_2, \ldots, \mathfrak{p}_N : \mathcal{N}(\mathfrak{p}_i) \leq B \ \forall \ i \in [1, N]\}. \tag{7.7}$$

B must be carefully chosen keeping into account two considerations. Firstly, the factor base must be sufficient to generate the class group. Secondly, the size of the factor base should not be too large as otherwise, it will make the linear algebra phase costly.
In general, the value is assumed to be of the form $B = L_{|\Delta_\mathcal{K}|}(\beta, c_b)$ for some $0 < \beta < 1$ and $c_b > 0$. The values of β, c_b are determined so that the complexity of the entire method is optimal.

Bach Bound: Assuming Extended Riemann Hypothesis (ERH), Bach ([11], Theorem 4.4) showed that taking classes of ideals with a representative norm at most $B = 12(\log|\Delta_K|)^2$, is sufficient to generate the class group. This quantity $12(\log|\Delta_K|)^2$ is known as the Bach Bound. In the case of quadratic number fields, it can be reduced to $6(\log|\Delta_K|)^2$. In practice, B may be taken somewhat higher to facilitate the collection of relations.

An asymptotic estimate of the number of prime ideals having norm below B is obtained from Landau's prime ideal theorem [116]. This is a generalization of the prime number theorem for

number fields. It is stated in Theorem 7.16 below. This makes the cardinality of the factor base to be about B.

Theorem 7.16. *Given a number field $\mathcal{K}$ and a bound $B \in \mathbb{N}$, the number of prime ideals of $\mathcal{K}$ having norm below B denoted by $\pi_{\mathcal{K}}(B)$ satisfies:*

$$\pi_{\mathcal{K}}(B) \sim \frac{B}{\log B}. \tag{7.8}$$

7.3.2 Relation Collection

It is assumed that the factor base $\mathcal{B}$ has been chosen such that it generates the class group. Thus, there is a surjective morphism (as in equation 7.9) which maps N tuples of integers to the class group representative.

$$\begin{gathered} \mathbb{Z}^N \xrightarrow{\phi} \mathcal{I} \xrightarrow{\pi} Cl_{\mathcal{O}_{\mathcal{K}}} \\ (e_1, e_2, \ldots, e_N) \mapsto \prod_{i=1}^{N} \mathfrak{p}_i{}^{e_i} \mapsto \prod_{i=1}^{N} [\mathfrak{p}_i]^{e_i}. \end{gathered} \tag{7.9}$$

The class group can be computed as

$$Cl_{\mathcal{O}_{\mathcal{K}}} \backsimeq \mathbb{Z}^N / \mathrm{Ker}(\pi \circ \phi). \tag{7.10}$$

From equation (7.10), it is sufficient to compute the kernel $\mathrm{Ker}(\pi \circ \phi)$ in order to compute the class group. From the maps in the mapping (7.9), the kernel

$$\mathrm{Ker}(\pi \circ \phi) = \{(e_1, e_2, \ldots, e_N) : \prod_{i=1}^{N} \mathfrak{p}_i{}^{e_i} = \langle x \rangle\} \tag{7.11}$$

consists of all N tuples which are mapped to some principal ideal $\langle x \rangle$ of $\mathcal{O}_{\mathcal{K}}$. Thus the kernel can be deduced from the lattice of $(e_1, e_2, \ldots, e_N) \in \mathbb{Z}^N$ such that $\prod_{i=1}^{N} \mathfrak{p}_i{}^{e_i} = \langle x \rangle$. The purpose of relation collection is to collect relations of the form given in equation (7.12). These relations are utilised to construct the lattice:

$$\prod_{i=1}^{N} \mathfrak{p}_i{}^{e_{i,j}} = \langle x_j \rangle. \tag{7.12}$$

On finding such a relation a row of exponents is appended to the *relation matrix* $M := \left((e_{i,j})_{1 \le i \le N}\right)$ where $(e_{1,j}, e_{2,j}, \ldots, e_{N,j})$ denotes the N-tuple of exponents for the j-th relation. The search for relations can be halted when the lattice generated by the rows of M has rank N. It is assumed heuristically that this search for relations can end when some more than N relations is collected.

7.3.3 Linear Algebra

Once a sufficient number of relations are collected a linear algebra step is performed on the matrix $M \in \mathbb{Z}^{m \times N}$ where m denotes the number of relations collected. The aim is to compute the *Smith-Normal-Form* (SNF) of M, *i.e.*, integers $d_1, d_2, \ldots, d_N$ are found with

$$d_N | d_{N-1} | \ldots | d_1$$

such that there exists two unimodular matrices $U \in \mathbb{Z}^{m \times m}$ and $V \in \mathbb{Z}^{N \times N}$, with the condition

$$M = U \left(\begin{array}{ccc} d_1 & & (0) \\ & \ddots & \\ (0) & & d_N \\ \hline & & \\ & (0) & \\ & & \end{array}\right) V.$$

The matrix

$$S = \left(\begin{array}{ccc} d_1 & & (0) \\ & \ddots & \\ (0) & & d_N \\ \hline & & \\ & (0) & \\ & & \end{array}\right) \tag{7.13}$$

is the SNF of M.

If $m = N$ the class group structure can be computed from the SNF.

The *Hermite Normal Form* (HNF) of the relation matrix is also useful in general.

For every matrix $M \in \mathbb{Z}^{m \times N}$, there is a unimodular matrix $W \in \mathbb{Z}^{m \times m}$ and a matrix H such that $H = WM$. The matrix H is of the form,

$$H = \left(\begin{array}{cccc} h_{1,1} & 0 & \ldots & 0 \\ \vdots & h_{2,2} & \ddots & \vdots \\ \vdots & \vdots & \ddots & 0 \\ * & * & \ldots & h_{N,N} \\ \hline & & & \\ & (0) & & \\ & & & \end{array}\right) \tag{7.14}$$

along with the conditions $0 \leq h_{ij} < h_{ii}\ \forall j < i$ and $h_{ij} = 0\ \forall j > i$. This matrix H is called the HNF of M. Computing HNF consists only of row operations, so the rows of H also represent relations.

7.3.4 Computation of the Class Group

The class group is computed after the linear algebra step is performed.
Case 1: If $m = N$: In this case, the lattice L spanned by the rows of M gives the class number. From the SNF with diagonal entries as $d_1, \ldots, d_N$ as in equation (7.13) we get

$$\mathbb{Z}^N/L \backsimeq \mathbb{Z}/d_1\mathbb{Z} \times \mathbb{Z}/d_2\mathbb{Z} \times \ldots \mathbb{Z}/d_N\mathbb{Z}. \tag{7.15}$$

Furthermore as the exponents of relations (*i.e.*, as in relation equation (7.12)) are elements of the kernel (given in equation (7.11)), the lattice L is same as the kernel of the surjective morphism. From equations (7.15) and (7.10) the class group can be computed directly as

$$Cl_{\mathcal{O}_\mathcal{K}} \backsimeq \mathbb{Z}/d_1\mathbb{Z} \times \mathbb{Z}/d_2\mathbb{Z} \times \ldots \mathbb{Z}/d_N\mathbb{Z}. \tag{7.16}$$

Case 2: Next, the case of rectangular relation matrix is described. In practice relation matrices obtained in the process are mostly full-rank when the number of relations collected exceeds the cardinality of the factor base, *i.e.*, $M \in \mathbb{Z}^{m \times N}$ with $m > N$. In this case, HNF (equation (7.14)) is utilised to obtain the class group.
For the HNF matrix H, there exists an integer l, $1 \leq l \leq N$ such that $h_{i,i} = 1\ \forall i \geq l$. The upper left $l \times l$ minor of H is called the *essential part*. This essential part being a square matrix, its SNF similarly gives the class group structure as in case 1. Additionally, this essential matrix has a smaller dimension compared to the original matrix.

Both the regulator and the fundamental units of $\mathcal{K}$ can be obtained from the kernel vectors of the relation matrix. The actual regulator and the fundamental units are procured from the relation matrix when the assumption of the completeness of its lattice holds. The procedure is described briefly below. Further details are available in [44].

7.3.5 Computation of the Regulator and the Fundamental Units

Matrix Construction for Regulator Computation: The matrix construction in Section 7.3.2 is aimed solely for class group computation. The matrix required for computing the regulator differs from that. Let us denote the matrix as M_{Reg} when the target is to compute the regulator

along with the class group. The matrix M_{Reg} has the following structure.

$$M_{Reg} = \left(\quad M \quad \middle| \, M_l \, \right)$$

where M is the relation matrix that has been used during class group computation and M_l is the matrix of rows of logarithms constituted for regulator computation. They can be written as

$$M = (e_{ij}) \in \mathbb{Z}^{m \times N}$$

$$M_l = (\log |\sigma_i|_j) \in \mathbb{R}^{m \times (r+1)}.$$

After finding a relation of the form (7.12), $(e_{1,j}, e_{2,j}, \ldots, e_{N,j}, \log|\sigma_1(x_j)|, \log|\sigma_2(x_j)|, \ldots, \log|\sigma_{r+1}(x_j)|)$ forms a row of M_{Reg}. Let L_l be the lattice generated by the rows of M_l.

Any vector $V = (v_1, v_2, \ldots, v_m)$ in the kernel of the relation matrix M, satisfies $VM = 0$. Then,

$$(\prod_j {x_j}^{v_j})\mathcal{O}_\mathcal{K} = \mathcal{O}_\mathcal{K} = \langle 1 \rangle.$$

Let $\gamma := (\prod_j {x_j}^{v_j})$. From above, γ is a unit. Thus, the units of $\mathcal{K}$ can be computed from the kernel of the relation matrix. Once the units are computed, the generators of the group spanned by these units can be obtained. The determinant of the generators confirm whether they form a system of fundamental units. If these generators form a set of fundamental units, the regulator is obtained. Otherwise, the group spanned is a subgroup of the unit group $\mathcal{U}_\mathcal{K}$ and hence a multiple of the regulator is obtained. Let us now give a detailed procedure to compute the regulator.
If $\dim(\ker(M)) = d$, then d units $\gamma_1, \gamma_2, \ldots, \gamma_d$ can be obtained. This allows to compute the matrix

$$A_\mathbb{R} = \big(Log(\gamma_i)\big)_{i \le d} \in \mathbb{R}^{d \times (r+1)}$$

where the *Log* map is defined in mapping (7.5). A basis of the lattice L_l generated by the rows of $A_\mathbb{R}$ is next computed. LLL algorithm may be used to do so. If $T = (t_1, t_2, \ldots, t_d)$ is a vector of the transformation matrix then the unit $u := {\gamma_1}^{t_1}{\gamma_2}^{t_2} \ldots {\gamma_d}^{t_d}$ corresponds to the row vector $TA_\mathbb{R} = \big(Log(u)\big)$. The transformation vectors $T_1, T_2, \ldots, T_r$ obtained from the LLL algorithm facilitate the computation of the system of fundamental units $(u_1, \ldots, u_r)$ and a matrix $B_\mathbb{R} = \big(Log(u_i)\big)_{i \le r}$. The determinant of any r-sized minor of the $r \times (r+1)$ matrix $B_\mathbb{R}$ supplies the hypothetical regulator for the relations collected.

7.3.6 Verification

The number of relations collected in the relation collection phase may not be sufficient to assure that the lattice generates the kernel of the surjective morphism in mapping (7.9). Thus the

assumption made on the completeness of the lattice of relations has to be verified.
Let us use the following notations for the number field $\mathcal{K}$.

$$\left.\begin{array}{lcl} h_{\mathcal{K}} & : & \text{class number;} \\ w_{\mathcal{K}} & : & \text{number of roots of unity;} \\ \zeta_{\mathcal{K}} & : & \text{Dedekind zeta function ;} \\ \Delta & : & \text{Log-unit lattice;} \end{array}\right\} \tag{7.17}$$

The *Dedekind zeta function* is defined as follows.

Definition 7.17. *Dedekind Zeta Function:* Given a number field $\mathcal{K}$, the Dedekind zeta function is defined by the formula

$$\zeta_{\mathcal{K}} = \sum_{a} \frac{1}{\mathcal{N}(a)^s} = \prod_{p} \frac{1}{1 - \frac{1}{\mathcal{N}(p)^s}} \tag{7.18}$$

where the sum is taken over all non-zero integral ideals and the product over all non-zero prime ideals of $\mathcal{O}_{\mathcal{K}}$.

The residue can be approximated by equation (7.19) ([44], Theorem 4.9.12)

$$\lim_{s \to 1}(s-1)\zeta_{\mathcal{K}}(s) = \frac{2^{r_1} 2\pi^{r_2} h_{\mathcal{K}} Reg_{\mathcal{K}}}{w_{\mathcal{K}} \sqrt{|\Delta_{\mathcal{K}}|}} \tag{7.19}$$

where (r_1, r_2) is the signature of $\mathcal{K}$.
It may be noticed that all quantities in equation (7.19) except $h_{\mathcal{K}} Reg_{\mathcal{K}}$ can be computed easily. The unknown product can be estimated once the left-hand side of the equation is known. The residue in equation (7.19) can also be expressed as the Euler Product $\prod_p \frac{1-\frac{1}{p}}{\prod_{\mathbf{p}|p}(1-\frac{1}{\mathcal{N}(\mathbf{p})})}$ where the product runs over all primes p. This approximation can be obtained after selection of the factor base. Bach [12] showed a good approximation is obtained in polynomial time considering primes of norm below $O(\log |\Delta_{\mathcal{K}}|^2)$. Then an easy approximation of $h_{\mathcal{K}} Reg_{\mathcal{K}}$ is obtained from equation (7.20).

$$\frac{2^{r_1} 2\pi^{r_2} h_{\mathcal{K}} Reg_{\mathcal{K}}}{w_{\mathcal{K}} \sqrt{|\Delta_{\mathcal{K}}|}} = \prod_{p} \frac{1 - \frac{1}{p}}{\prod_{\mathbf{p}|p}(1 - \frac{1}{\mathcal{N}(\mathbf{p})})}. \tag{7.20}$$

Section 7.3.5 of this chapter describes how to obtain an approximation of the regulator of $\mathcal{K}$. After deriving the hypothetical regulator, the hypothetical class number can be obtained from the product of the diagonal entries of the SNF at the linear algebra step itself. For the sake of verification, the product of these two is compared with the approximation calculated from the Euler product. These approximations allows to derive a bound $\bar{h}$ in polynomial time under ERH such that the inequality (7.21) [12] holds.

$$\bar{h} \leq h_{\mathcal{K}} Reg_{\mathcal{K}} \leq 2\bar{h}. \tag{7.21}$$

If the values of the determinant of Δ and L do not satisfy the inequality (7.21), then more relations have to be collected. More specifically, if the ratio after the approximations are obtained in equation (7.20) is about 1, then the target class number and regulator have been obtained. If not, then more relations have to be added to the relation matrix. Adding a relation when sufficient number of prime ideals and units are present decreases the candidate class number and the regulator respectively by an integer factor.

7.4 Previous Works

In this section, several attempts to compute class group has been listed. The initial attempts were of exponential complexity which were later improved to sub-exponential timing estimates.

The earlier attempts were in the restrictive class of imaginary quadratic orders. In 1968, Shanks' [159, 160] proposed an exponential algorithm to compute the ideal class group of an imaginary quadratic number field. It is based on the baby-step giant-step algorithm and runs in time $O(|\Delta_{\mathcal{K}}|^{\frac{1}{4}+\epsilon})$ or $O(|\Delta_{\mathcal{K}}|^{\frac{1}{5}+\epsilon})$ under the extended Riemann hypothesis [122]. This algorithm can be utilised to compute class groups of discriminants having up to 20 or 25 decimal digits. Pohst and Zassenhaus [145] developed an algorithm to determine the class group of arbitrary number fields.

The best algorithms for class group computation are of subexponential complexity and is done using the index calculus methods.

In 1989, the first subexponential algorithm for computing class group of an imaginary quadratic field was proposed by Hafner and McCurley [85] assuming generalised Riemann hypothesis (GRH). The expected running time of this method is $L_{|\Delta_{\mathcal{K}}|}(\frac{1}{2}, \sqrt{2} + o(1))$. Buchmann [34] extended the method to all number fields. The heuristic complexity was $L_{|\Delta_{\mathcal{K}}|}(\frac{1}{2}, 1.7 + o(1))$ for fixed degree n and $\Delta_{\mathcal{K}}$ tending to infinity. This method was used by Adleman, Huang and DeMarrais to compute discrete logarithms in the Jacobian of hyperelliptic curves. Later, Enge *et al.* [56, 57] developed methods to compute the group structure of the Jacobian and solving the discrete logarithm problem for certain classes of curves in time $L(\frac{1}{3})$. The height of the defining polynomial appears when bounding the norm of algebraic integers. A smaller height also allows ease of computations. Keeping the degree fixed, the best choice amongst the infinite ways of constructing a polynomial for a number field is the one that has the smallest height. Cohen and Diaz y Diaz [48] proposed an algorithm for reducing defining polynomials.
Buchmann and Düllmann implemented the random walk method of Hafner and McCurley [85] in [38]. They also suggested improvements and implemented a practical version of Hafner McCurley's algorithm in [36], [37]. The largest discriminant for which this algorithm could compute the Class Group was 55. Trial division combined with a single large prime variant was used by Buchmann

and Düllmann [37] to compute the class group in about 10 days on a distributed system of workstations. An advancement to this method was suggested by Jacobson [89]. Considerable speed-up was obtained by using quadratic sieving strategies to the relation collection phase. Class groups up to 90 decimal digits were computed with this algorithm. This made way for the application of existing methods to improve the quadratic sieving method. The single large prime variant first suggested by Buchmann and Düllmann [37] was adapted by Jacobson for the quadratic sieve. Sutherland [168] computed class groups using generic methods for discriminants up to 100 decimal digits. A major point of difference of this algorithm with others till then was the fact that it was dependent on the structure of the class group. The improvement relied on the values of the discriminant for that particular class group and could be further sped up for suitable values of the discriminant. Cohen, Diaz Y Diaz and Olivier [46] made some practical improvements to Buchmann's [34] algorithm.

Recently, for extension degree n tending to infinity under certain restrictions, Biasse [19] described an algorithm for computing ideal class group and unit group of $\mathcal{O} = \mathbf{Z}[\theta]$ in heuristic time complexity $L_{\Delta_\mathcal{K}}(\frac{1}{3}, c)$ for some $c > 0$ valid for certain infinite classes of number fields. Biasse and Fieker [20], [21] showed that for all classes of number fields there is a heuristic subexponential algorithm. The methods of [21] results in better asymptotic complexity when applied to cyclotomic fields. It has a complexity of $L_{\Delta_\mathcal{K}}(\frac{2}{3}+\epsilon)$ in the general case and $L_{\Delta_\mathcal{K}}(\frac{1}{2})$ when the extension degree n satisfies $n \leq (\log(|\Delta_\mathcal{K}|))^{\frac{3}{4}-\epsilon}$. For some restrictive classes the complexity can be $L_{\Delta_\mathcal{K}}(a)$ where a is possibly as low as $\frac{1}{3}$. To be specific, this decrease in complexity occurs when a defining polynomial is known with small coefficients compared to the discriminant of the field. The *Block Korkin Zolotarev* (BKZ) lattice reduction algorithm [10, 155, 157] has been used to finally obtain an ideal decomposition. This leads to a trade-off between the time spent for reduction and the approximation factor of short vectors. This trade-off is achieved with *block size*. For a lattice with dimension n, the block size denoted by β is an integer such that $2 \leq \beta \leq n$ and the BKZ algorithm works with blocks of the lattice each having size β. The shortest vector v obtained as output after running BKZ, has norm $||v||$ which is bounded as $||v|| \leq \beta^{\frac{n-1}{2(\beta-1)}} (\det \mathcal{L})^{\frac{1}{n}}$ where $\mathcal{L}$ is the lattice with dimension n. Eventually it results in a complexity that permits both the degree and the discriminant to be large. The complexity of the relation collection phase is reduced due to the use of efficient smoothness tests like ECM. The q-descent strategy [21] reduces the complexity to $L_{|\Delta_\mathcal{K}|}(a)$ with $\frac{1}{3} \leq a \leq \frac{1}{2}$ when the defining polynomial of the number field has small height. Gélin and Joux [71] extended the improvement of [21] to larger classes of fields by describing an algorithm for obtaining polynomials with small coefficients for certain fields. The target fields are those for which it is possible to find a reduced polynomial whose coefficients are small compared to the field discriminant. This algorithm when used as a preliminary step with Biasse-Fieker's [21]

algorithm reduces the complexity. Classification of fields based on certain parameters were introduced which has been crucially utilised in all later works to date. Gélin [70] later proposed sieving based relation collection algorithm on this line.

Quite recently, Gélin [69] further improved the results of [21]. The improvement in relation collection then reduces the complexity for large degree number fields to $L_{|\Delta_\mathcal{K}|}(a)$ with $a \in [\frac{1}{2}, \frac{2}{3}]$ from the previous complexity of $L_{|\Delta_\mathcal{K}|}(\frac{2}{3} + \epsilon)$ as in [21]. This algorithm is specially designed for fields that do not have small defining polynomial. The complexity of $L_{|\Delta_\mathcal{K}|}(\frac{1}{2})$ mentioned in [21] is also made explicit as $L_{|\Delta_\mathcal{K}|}(\frac{1}{2}, \frac{(\omega-1)}{2\sqrt{\omega}})$. Using some special techniques of lattice reduction the complexity is shown to vary between $L_{|\Delta_\mathcal{K}|}(\frac{1}{2})$ and $L_{|\Delta_\mathcal{K}|}(\frac{3}{5})$. Besides this, it is also shown that the classification of number fields introduced in [71] accommodates all number fields. The values of the parameters have been suggested keeping the various cases of classification into account. In his later work, Gélin [70] had further improved his previous results for certain number fields. The paper introduces *sieving strategy* which outperforms the previous ideal reduction algorithm in [69] for some number fields. A neat categorisation based on associated parameters depicts the exact region for which each algorithm in the two papers of Gelin [69, 70] may be suitable. Conditional improvement is achieved based on the smallness of the defining polynomial. This reduces the complexity estimates from the previous paper of Gélin. It extends the sieving strategy of [19] to all number fields and obtains a low complexity of $L_{|\Delta_\mathcal{K}|}(\frac{1}{3})$. In some category of fields, it improves the second constant of [19]. It is to be noted that the complexity analysis of all the algorithms are heuristic; one of the main heuristic assumptions being that smoothness results for general ideals are assumed to apply to the restricted set of ideals considered by the algorithms.

Chapter 8

Pseudo-Random Walk on Ideals: Practical Speed-Up in Relation Collection for Class Group Computation

In the present chapter, we provide a new heuristic algorithm to improve the practical runtime of the relation collection step in Gélin's algorithm [69]. The core idea of previous ideal reduction algorithms is to randomly generate ideals on which lattice techniques are used to obtain principal ideals of bounded norms which can then be tested for smoothness over a predefined factor basis. Presently, the most simplified form of this algorithm is due to Gélin [69]. The random generation of the ideals in Gélin's algorithm requires exponentiating some randomly chosen prime ideals in the factor basis and then multiplying the resulting ideals together. This requires performing a number of ideal multiplications.

We propose to perform a pseudo-random walk on ideals. Once the initial ideal has been computed, each step of the walk requires a single ideal multiplication. The ideals generated in each step can be processed using lattice techniques as in Gélin's algorithm. The parameters which determine the asymptotic complexity of Gélin's algorithm [69] are not changed. So, under the heuristic assumption that the ideals in the pseudo-random walk behave like random ideals, the asymptotic complexity of the new algorithm remains unchanged from that of Gélin's algorithm. The improvement is a practical gain in the time to generate a relation. Table 8.5 of this chapter provides the actual estimates for the fields chosen.

We have made a Magma implementation of both the new algorithm and Gélin's algorithm [69]. Since our goal is to compare the new algorithm with Gélin's algorithm [69], we did not try to compute the class group for one particular field. Instead, we chose four number

fields and conducted experiments with both the algorithms to collect a number of relations. The timing results obtained from these experiments indeed confirm the speed improvement of the new algorithm over Gélin's algorithm [69].

8.1 Generating Relations

Suppose that $\mathcal{B} = \{\mathfrak{p}_1, \ldots, \mathfrak{p}_N\}$ is the set of all prime ideals having norms less than B where B itself is at least as large as $12(\log|\Delta_\mathcal{K}|)^2$. For the asymptotic analysis, B is chosen to be $B = L_{|\Delta_\mathcal{K}|}(\delta, c_b)$ for suitable values of δ and c_b; this point is discussed later. For appropriate choices of δ and c_b the overall runtime of the algorithm is sub-exponential. We refer to [69] for details.

The set $\mathcal{B}$ is said to be the factor basis. For $x \in \mathcal{O}_\mathcal{K}$ and integers $e_1, \ldots, e_N$, a relation is given by

$$\mathfrak{p}_1^{e_1} \cdots \mathfrak{p}_N^{e_N} \quad = \quad \langle x \rangle. \tag{8.1}$$

The main task of the class group computation algorithm is to generate relations of the form (8.1).

The basic idea for obtaining relations is to generate random principal ideals and then check whether it is smooth over $\mathcal{B}$. In practice, given a principal ideal $\langle x \rangle$, one checks whether the positive integer $\mathcal{N}(\langle x \rangle) = |\mathcal{N}(x)|$ is B-smooth. If this holds, then $\langle x \rangle$ is factored.

Given a random element of x of $\mathcal{O}_\mathcal{K}$, the probability of $|\mathcal{N}(x)|$ being B-smooth is very small. So, simply trying out random elements of $\mathcal{O}_\mathcal{K}$ will not lead to an efficient algorithm. To ensure that the probability of $|\mathcal{N}(x)|$ being B-smooth is reasonable, it is required to choose x in a manner which ensures that $|\mathcal{N}(x)|$ satisfies some pre-determined upper bound. The literature contains various methods of choosing x such that $|\mathcal{N}(x)|$ is below a desired bound.

Given an element $\alpha \in \mathcal{O}_\mathcal{K}$, its canonical embedding $\sigma(\alpha)$ into $\mathbb{C}^n$ is $\sigma(\alpha) = (\sigma_1(\alpha), \ldots, \sigma_n(\alpha))$. Since the complex embeddings occur in pairs, the vector $(\sigma_1(\alpha), \ldots, \sigma_n(\alpha))$ can be considered to be represented by a vector from $\mathbb{R}^n$. So, henceforth, we will consider $\sigma(\alpha)$ be a vector in $\mathbb{R}^n$. Consider the lattice $\sigma(\mathfrak{a})$ obtained by the embedding of an ideal $\mathfrak{a}$ of $\mathcal{O}_\mathcal{K}$. Suppose v is a short vector in $\sigma(\mathfrak{a})$ and let x_v be the corresponding element in $\mathfrak{a}$ such that $\sigma(x_v) = v$. Then $|\mathcal{N}(x_v)|$ is small and so the principal ideal $\langle x_v \rangle$ also has small norm.

Algorithm 4 describes Gélin's method of generating relations using ideal reduction. The parameters of the algorithm are k, A, β and B. The parameter k determines the number of ideals to be considered in each iteration, A determines the maximum value of the exponent to which the ideals are raised, the parameter β determines the block size of the BKZ-reduction, and the parameter B is the bound on the norms of the ideals in the factor basis. Gélin specifies k and A to be poly$(\log|\Delta_\mathcal{K}|)$. Further, the value of β is to be chosen such that the overall complexity is subexponential in $|\Delta_\mathcal{K}|$.

In Step 3.5 of Algorithm 4, a single vector from the BKZ reduction is returned. It is, however, possible that several vectors from the BKZ reduction have sufficiently small norms and a linear combination of these vectors can be tried. This saves the number of BKZ reductions and leads to practical speed-ups. Such improvement does not change the asymptotic complexity. We refer to [69] for details of this improvement and other implementation notes.

Input: The factor base $\mathcal{B} = \{\mathfrak{p}_1, \ldots, \mathfrak{p}_N\}$.
Output: The set of generated relations.
3.1 **while** *sufficient relations have not been obtained* **do**
3.2 Choose k random prime ideals $\mathfrak{p}_{j_1}, \mathfrak{p}_{j_2}, \ldots, \mathfrak{p}_{j_k}$ from $\mathcal{B}$
3.3 Choose k random exponents $e_{j_1}, e_{j_2}, \ldots, e_{j_k}$ from $\{1, 2, \ldots, A\}$
3.4 Set $\mathfrak{a} = \prod_{i=1}^{N} \mathfrak{p}_i{}^{e_i}$ with $e_i = 0$ if $i \notin \{j_1, \ldots, j_k\}$
3.5 Compute the smallest element v of the BKZ$_\beta$ reduced basis of the lattice $\sigma(\mathfrak{a})$
3.6 Obtain the algebraic integer x_v corresponding to v
3.7 Set $\mathfrak{b}$ as the unique ideal such that $\langle x_v \rangle = \mathfrak{a}\mathfrak{b}$
3.8 **if** $|\mathcal{N}(\mathfrak{b})|$ *is* B*-smooth* **then**
3.9 Obtain the factorization of $\mathfrak{b}$ such that $\mathfrak{b} = \prod_{i=1}^{N} \mathfrak{p}_i{}^{e'_i}$
3.10 Store the relation $\langle x_v \rangle = \prod \mathfrak{p}_i{}^{e_i + e'_i}$
3.11 **end**
3.12 **end**

Algorithm 4: Gélin's method for generating relation using ideal reduction [69].

Step 3.8 of Algorithm 4 checks whether $\mathcal{N}(\mathfrak{b})$ is B-smooth. The actual requirement is that the ideal $\mathfrak{b}$ is $\mathcal{B}$-smooth. If $\mathcal{N}(\mathfrak{b})$ is B-smooth, then it follows that $\mathfrak{b}$ is $\mathcal{B}$-smooth and a factorisation of $\mathfrak{b}$ can be obtained from a factorisation of $\mathcal{N}(\mathfrak{b})$. We refer to [69] for details.
A crucial issue in the relation collection has been pointed out by Gélin [69]. The relation collection phase is completed when the number of relations is larger than $\#\mathcal{B}$ and when all ideals of norms below Bach's bound are involved in at least one relation.

Notation: Before proceeding further, we fix some notation.

B	upper bound on the size of the ideals in the factor basis;
$\mathcal{B}$	the factor basis;
N	number of prime ideals in the factor basis, i.e., $N = \#\mathcal{B}$;
C	Bach's bound, i.e., $C = 12(\log \lvert\Delta_K\rvert)^2$;
$\mathcal{C}$	the set of prime ideals whose norms are at most C;
N_b	number of prime ideals whose norms are at most C, i.e., $N_b = \#\mathcal{C}$;
β	block size of BKZ reduction.

8.2 Generating Relations from a Pseudo-Random Walk on Ideals

Consider Algorithm 4. The asymptotic complexity of the algorithm is determined by B and β. The parameter β determines the upper bound on the norm of the element x_v; B and β determine the smoothness probability of the ideal $\langle x_v \rangle$.

We do not propose to change these parameters. Instead, we focus on the concrete complexity of an individual iteration. This consists of computing the ideal $\mathfrak{a}$, computing the BKZ-reduction, and the smoothness check of the norm of $\langle x_v \rangle$. The portions on BKZ-reduction and the smoothness check are also not modified. Our focus is on reducing the cost of computing the ideal $\mathfrak{a}$.

In Step 3.4 of Algorithm 4, the ideal $\langle \mathfrak{a} \rangle$ is computed as

$$\mathfrak{a} \;=\; \prod_{i=1}^{k} \mathfrak{p}_{j_i}^{e_{j_i}}. \tag{8.2}$$

Since each e_{j_i} is chosen randomly from $\{1, 2, \ldots, A\}$, computing $\mathfrak{p}_{j_i}^{e_{j_i}}$ requires about $\log_2 A$ ideal multiplications. So, the total cost of computing $\mathfrak{a}$ in (8.2) is about $(k-1)\log_2 A$ ideal multiplications. Below we describe a new algorithm for generating $\langle \mathfrak{a} \rangle$ which requires a single ideal multiplication.

Our idea is to perform a pseudo-random walk on a sufficiently large set of ideals. Each step of the walk will generate an ideal $\mathfrak{a}$ to which the BKZ-reduction and the rest of Algorithm 4 can be applied. The cost of each step will consist of a single ideal multiplication. The pseudo-random walk that we construct is inspired by Pollard's rho algorithm.

Suppose $[\mathfrak{t}_1, \ldots, \mathfrak{t}_m]$ is a list of m pre-computed ideals. Let $\mathcal{H}$ be a hash function which maps an ideal to the set $\{0, \ldots, m-1\}$. For $i \geq 0$, the pseudo-random walk proceeds as follows. An ideal $\mathfrak{a}_0$ is chosen and for $i \geq 1$, a_i is obtained as follows: let $m_i = \mathcal{H}(\mathfrak{a}_{i-1})$ and set $\mathfrak{a}_i = \mathfrak{a}_{i-1} \cdot \mathfrak{t}_{m_i}$.

The list $[\mathfrak{t}_1, \ldots, \mathfrak{t}_m]$ and some additional information are stored in a table $\mathfrak{T}$. We explain how the ideals $\mathfrak{t}_1, \ldots, \mathfrak{t}_m$ are constructed and the entries of the table $\mathfrak{T}$.

Let $\mathcal{C} = \{\mathfrak{q}_1, \ldots, \mathfrak{q}_{N_b}\}$ be the prime ideals whose norms are below Bach's bound. Let κ be a parameter and set $q = \lfloor N_b/\kappa \rfloor$ and $r = N_b - q\kappa$ so that we have $N_b = q\kappa + r = r(\kappa+1) + (q-r)\kappa$. The set $\mathcal{C}$ is randomly partitioned into q groups where the first r groups each have $\kappa+1$ ideals and the last $q-r$ groups each have κ ideals. The ideals in each of the groups are multiplied together and the products are stored in $\mathfrak{T}$. Along with the product ideal, the information identifying the ideals that have been multiplied to obtain the product is also stored in $\mathfrak{T}$. The random partitioning of the ideals in $\mathcal{C}$ into groups, multiplying together the ideals in each group and storing them in table $\mathfrak{T}$ is carried out a total of R times. So, the number of entries in $\mathfrak{T}$ is qR, i.e., $m = qR$. Further, each ideal in $\mathcal{C}$ is represented a total of R times in the table $\mathfrak{T}$. The method for constructing $\mathfrak{T}$ is shown in Algorithm 5.

Input: The set of prime ideals $\mathfrak{C} = \{\mathfrak{q}_1, \ldots, \mathfrak{q}_{N_b}\}$ whose norms are below Bach's bound.
Output: The table $\mathfrak{T}$.

4.1 $q \leftarrow \lfloor N_b/\kappa \rfloor$, $r \leftarrow N_b - q\kappa$
4.2 $\mathfrak{T} \leftarrow \emptyset$
4.3 $\mathcal{J} \leftarrow \{1, 2, \ldots, N_b\}$
4.4 **for** $i_1 \leftarrow 1$ ***to*** R **do**
4.5 $\mathcal{I} \leftarrow \mathcal{J}$
4.6 **for** $i_2 \leftarrow 1$ ***to*** q **do**
4.7 **if** $i_2 \leq r$ **then**
4.8 $s \leftarrow (\kappa + 1)$
4.9 **end**
4.10 **else**
4.11 $s \leftarrow \kappa$
4.12 **end**
4.13 $\{j_1, j_2, \ldots, j_s\} \leftarrow$ random set of s distinct integers chosen from $\mathcal{I}$
4.14 $\mathcal{I} \leftarrow \mathcal{I} \setminus \{j_1, j_2, \ldots, j_s\}$
4.15 $\mathfrak{b} \leftarrow \mathfrak{q}_{j_1} \cdots \mathfrak{q}_{j_s}$
4.16 Append $(\mathfrak{b}, (j_1, \ldots, j_s))$ to $\mathfrak{T}$
4.17 **end**
4.18 **end**

Algorithm 5: Construction of the pre-computed table $\mathfrak{T}$.

There are a total of Rq entries in $\mathfrak{T}$, i.e., $m = Rq$. The entries of $\mathfrak{T}$ are pairs. For $0 \leq i \leq m-1$, $\mathfrak{T}[i]$ is an entry of the form $(\mathfrak{b}, (j_1, \ldots, j_s))$. By $\mathfrak{T}[i].\mathsf{ideal}$ we will denote the ideal $\mathfrak{b}$ and by $\mathfrak{T}[i].\mathsf{index}$ we will denote the tuple $(j_1, \ldots, j_s)$.

The table $\mathfrak{T}$ is constructed in a pre-computation phase. This pre-computation consists of about $Rq\kappa \approx RN_b$ ideal multiplications. When R is a constant, the number of ideal multiplications required to construct $\mathfrak{T}$ is negligible with respect to the number of ideal multiplications required in all the iterations for relation collection.

How long should the pseudo-random walk proceed? There are several aspects to this question.

1. As the walk progresses, both the number and the exponents of the prime ideals occurring as factors in the ideals visited by the walk increases. So, a long walk can result in ideals with large norms.

2. From a practical point of view, in our Magma implementation we have observed that as the norms of the ideals increase, it becomes difficult to construct the lattices corresponding to the ideals.

In view of the above two points, long walks are not feasible, at least for Magma implementation. One way is to continue the walk as long as it is feasible to construct the associated lattice. Alternatively, one may put an *a priori* bound on the length of an individual walk. Determining the bound requires performing some initial experiments to obtain an idea of the number of steps that the walk can proceed without encountering the failure of lattice construction.

The algorithm for relation collection based on the pseudo-random walk is shown in Algorithm 6. The parameters β and B are the same as those in Algorithm 4. It is assumed that the table $\mathfrak{T}$ has been constructed prior to the execution of Algorithm 6. Recall that the ideals stored in $\mathfrak{T}$ are products of either κ or $\kappa+1$ prime ideals whose norms are below N_b. Algorithm 6 uses an additional parameter κ_0 which determines the number of prime ideals to be multiplied together to obtain the starting ideal of a walk. The variable wlen records the current length of the walk, while the parameter λ represents the maximum length of each walk.

As above, let $\mathcal{C} = \{\mathfrak{q}_1, \ldots, \mathfrak{q}_{N_b}\}$ be the set of all prime ideals whose norms are below Bach's bound. The actual factor basis $\mathcal{B} = \{\mathfrak{p}_1, \ldots, \mathfrak{p}_N\}$ consists of all prime ideals whose norms are below the bound B. We assume that for $i = 1, \ldots, N_b$, $\mathfrak{p}_i = \mathfrak{q}_i$, i.e., the first N_b prime ideals in $\mathcal{B}$ have norms below Bach's bound.

Input: The factor base $\mathcal{B} = \{\mathfrak{p}_1, \ldots, \mathfrak{p}_N\}$.
Output: The set of generated relations.

5.1 **while** *Sufficient relations have not been found* **do**
5.2 $\quad$ exp$[i] = 0$ for $i = 1, \ldots, N$
5.3 $\quad$ Choose s randomly from $\{1, \ldots, \kappa_0\}$
5.4 $\quad$ Choose $i_1, \ldots, i_s$ randomly from $\{1, \ldots, N_b\}$
5.5 $\quad$ Set $\mathfrak{a} = \mathfrak{p}_{i_1} \cdots \mathfrak{p}_{i_s}$
5.6 $\quad$ exp$[i] \leftarrow 1$ for $i = i_1, \ldots, i_s$;
5.7 $\quad$ wlen $\leftarrow 1$;
5.8 $\quad$ **while** wlen $\leq \lambda$ **do**
5.9 $\quad\quad$ Compute the smallest element v of the BKZ$_\beta$ reduced basis of the lattice $\sigma(\mathfrak{a})$
5.10 $\quad\quad$ Obtain the algebraic integer x_v corresponding to v
5.11 $\quad\quad$ **if** $|\mathcal{N}(x_v)/\mathcal{N}(\mathfrak{a})|$ *is* B*-smooth* **then**
5.12 $\quad\quad\quad$ Set $\mathfrak{b}$ as the unique ideal such that $\langle x_v \rangle = \mathfrak{a}\mathfrak{b}$
5.13 $\quad\quad\quad$ Obtain the factorization of $\mathfrak{b}$ such that $\mathfrak{b} = \prod_{i=1}^{N} {p_i}^{e'_i}$
5.14 $\quad\quad\quad$ **for** $i \leftarrow 1$ *to* N **do**
5.15 $\quad\quad\quad\quad$ exp$[i] \leftarrow$ exp$[i] + e'_i$
5.16 $\quad\quad\quad$ **end**
5.17 $\quad\quad\quad$ Store the relation $\langle x_v \rangle = \prod_{i=1}^{N} {\mathfrak{p}_i}^{\exp[i]}$
5.18 $\quad\quad$ **end**
5.19 $\quad\quad$ $\ell \leftarrow \mathcal{H}(\mathfrak{a})$
5.20 $\quad\quad$ $\mathfrak{a} \leftarrow \mathfrak{a} \cdot \mathfrak{T}[\ell]$.ideal
5.21 $\quad\quad$ exp $\leftarrow$ exp $+\mathfrak{T}[\ell]$.index
5.22 $\quad\quad$ wlen $\leftarrow$ wlen $+ 1$;
5.23 $\quad$ **end**
5.24 **end**

Algorithm 6: Relation collection using pseudo-random walk.

In Algorithm 6, the ideals $\mathfrak{a}$ which are visited by the pseudo-random walk are products of prime ideals whose norms are below C. This is because a walk starts with such an ideal and for each step, the present ideal is multiplied with an ideal from the precomputed table. Since the

ideals in the precomputed table are also of the same type, the property of being products of prime ideals whose norms are below C holds for all the ideals $\mathfrak{a}$ visited by the walk. On the other hand, the smoothness check of $\mathcal{N}(\mathfrak{b})$ is with respect to B. As a result, the algorithm tries to ensure that $\mathfrak{b}$ is smooth over the factor basis $\mathcal{B}$ rather than $\mathcal{C}$. Trying to ensure the smoothness of $\mathfrak{b}$ over $\mathcal{C}$ will result in the theoretical smoothness probability being too low. In practice, however, it may be possible to work with $\mathcal{C}$ as the factor basis as we discuss later. Recall that one of the stopping criterion for relation collection is that each element of $\mathcal{C}$ is involved in at least one relation. Since the ideals $\mathfrak{a}$ are products of ideals from $\mathcal{C}$, this criterion becomes somewhat easier to ensure.

Remarks:

1. The list exp in Algorithm 6 is an array of N integers. This list is likely to be very sparse. So, it would be more efficient to represent exp as a list of pairs $[(i_1, e_1), \ldots, (i_s, e_s)]$, such that $\mathsf{exp}[i] = e_j$ if $i = i_j$, $j = 1, \ldots, s$ and $\mathsf{exp}[i] = 0$, otherwise. Another possibility is to represent exp as a list of integers where i_1 is repeated e_1 times, i_2 is repeated e_2 times, and so on. Since the integers $e_1, e_2, \ldots$ are quite likely to be equal to 1, this representation would be even more compact than storing the pairs $(i_1, e_1), (i_2, e_2), \ldots$. The operations on exp are to be suitably modified to be used with a compact representation.

2. The computation of the ideal $\mathfrak{b}$ in Step 3.7 of Algorithm 4 requires computing $\mathfrak{a}^{-1}$. Since $\mathcal{N}(\mathfrak{b})$ may not turn out to be B-smooth in Step 3.8, the computation of the ideal $\mathfrak{b}$ may actually not be required. Slightly altering the sequence of instructions, it is possible to avoid the computation of $\mathfrak{a}^{-1}$ in the cases where $\mathcal{N}(\mathfrak{b})$ is not B-smooth. Since $\mathcal{N}(\mathfrak{b}) = \mathcal{N}(x_v)/\mathcal{N}(\mathfrak{a})$, Algorithm 6 checks for the B-smoothness of $|\mathcal{N}(x_v)/\mathcal{N}(\mathfrak{a})|$ and computes $\mathfrak{b}$ only if the check is successful.

3. Note that the norms of the ideals in $\mathfrak{T}$ are known. The table $\mathfrak{T}$ can be expanded to store the norms of the ideals. This speeds up the computation of the norm of $\mathfrak{a}$ required in Step 5.11 of Algorithm 6. Along with the current ideal $\mathfrak{a}$ of the walk, its norm is also stored. The norm of the next ideal in the walk is obtained by multiplying the norm of the current ideal $\mathfrak{a}$ and the norm of the ideal in the location $\mathfrak{T}[\mathcal{H}(\mathfrak{a})]$.

4. The pseudo-random walk is a sequential procedure. Parallelism can be incorporated by starting independent pseudo-random walks. The table $\mathfrak{T}$ remains the same for all the walks. The starting ideals are chosen independently. This allows the separate walks to proceed independently.

5. Algorithm 6 has been described keeping in mind that the precision is fixed. Ideals in the initial steps of the walk have smaller norms compared to ideals in the later stages of the walk. So, an alternative approach would be to start with a smaller precision and increase

the precision as the walk progresses. Since precision determines the efficiency of the various computational steps, working with a lower precision in the initial stages would lead to improved efficiency. The idea of increasing precision as the walk progresses can work up to a certain point, since as the precision becomes too large, the computation slows down considerably. At that point, it is better to switch to a new initial point and start a new walk.

We compare the number of ideal multiplications required by Algorithms 4 and 6. For the comparison, we ignore the number of ideal multiplications required to prepare the table $\mathfrak{T}$. This is a one-time activity whose cost has been mentioned above; amortised over the iterations required for generating all the relations, this cost is negligible. We revisit this issue with respect to the experiments that have been conducted. A pseudo-random walk in Algorithm 6 proceeds for λ steps. These λ steps require at most $\kappa_0 + \lambda - 2$ ideal multiplications. In comparison, Algorithm 4 requires about $\lambda(k-1)\log_2 A$ ideal multiplications for generating λ ideals $\mathfrak{a}$ in λ iterations. The other costs of both Algorithms 4 and 6, namely BKZ-reduction, smoothness checking and ideal factorisation, remain the same, though there is the issue of the norms of the ideals $\mathfrak{a}$ to be considered. We discuss this point below.

Though in principle, the reduction in the number of ideal multiplications should lead to improvement in time, there is a practical aspect that needs to be kept in mind. In practice, the time to multiply two ideals depends on the norms of the ideals. Even if one of the ideals has a relatively large norm, there is a noticeable increase in the time to compute the product. While determining the walk length, this aspect needs to be kept in the mind. As the walk length increases, so does the norms of the ideals visited by the walk. This means that even though each step of the walk requires a single multiplication, the time for this multiplication increases with the length of the walk. So, if a walk is too long, then it may turn out that computing the next ideal of the walk takes very long time. In effect, this means that for Algorithm 6 to be competitive with Algorithm 4, the walk length should not be too long. In our experiments, we have fixed the walk length so that the norms of the ideals visited by the walk in Algorithm 6 are less than the norms of the ideals generated by Algorithm 4. In later sections, we provide experimental results from our Magma implementation to show that such a strategy indeed provides a substantial improvement in the relation collection time.

We further consider the issue of increase in the norms of the ideals visited by the walk. The first ideal in the walk has norm at most $\kappa_0 C$. At each step, the ideal in the walk is multiplied by κ or $\kappa+1$ prime ideals having norms at most C. So, the ideal obtained after i steps of the walk has norm at most $(\kappa_0 + i(\kappa+1))C$ which is at most $(\kappa_0 + \lambda(\kappa+1))C$ since $i \leq \lambda$. In comparison, the ideals $\mathfrak{a}$ generated in each iteration of Algorithm 4 have norms to be at most about $kAB/2$. So, the norms of the ideals in the walk in Algorithm 6 are at most the norms of ideals the ideals $\mathfrak{a}$ in Algorithm 4 if the condition $(\kappa_0 + \lambda(\kappa+1))C \leq kAB/2$ holds. The parameters may be chosen

to satisfy this condition. Note that in Algorithm 4, the norms of all the ideals are about $kAB/2$, while for Algorithm 6, the norms of the ideals in the initial steps of the walk are lower.

A pseudo-random walk chooses the first ideal randomly while the subsequent ideals are chosen deterministically. A crucial requirement in the asymptotic analysis of the algorithm is the smoothness probability of random ideals. Since the ideals appearing in a walk do not have independent randomness, one has to heuristically assume that the result on the smoothness probability of random ideals applies to the ideals occurring in the walk. Our experiments show that there is no substantial effect on the smoothness probability.

Let Π denote the probability that an ideal $\mathfrak{a}$ in the walk leads to an ideal $\mathfrak{b}$ computed in Step 5.12 of Algorithm 6 which is smooth over $\mathcal{B}$ and hence leads to a relation. About $1/\Pi$ ideals need to be considered to obtain a single relation. A total of about N relations are required. Consequently, about N/Π ideals need to be considered to obtain all the relations. Each walk visits λ ideals. The total number of walks required to consider N/Π ideals is $N/(\Pi\lambda)$. The total number of starting points is $\sum_{s=1}^{\kappa_0}\binom{N_b}{s}$. So, to ensure that $N/(\Pi\lambda)$ walks are possible, we must have

$$\sum_{s=1}^{\kappa_0}\binom{N_b}{\kappa_0} \geq \frac{N}{\Pi\lambda}. \tag{8.3}$$

The parameter κ_0 has to be chosen to satisfy (8.3).

As proved in [69], the norm of $\mathfrak{b}$ considered in Step 5.12 of Algorithm 6 satisfies the bound $\mathcal{N}(\mathfrak{b}) \leq \beta^{n(n-1)/(2(\beta-1))}\sqrt{|\Delta_\mathcal{K}|}$. Further, the B-smoothness of $\mathcal{N}(\mathfrak{b})$ depends on the value of B. We do not suggest any change to either the value of β or to the value of B. The difference between Algorithms 4 and 6 is in the generation of the ideal $\mathfrak{a}$. As discussed above, the norm of $\mathfrak{a}$ considered by Algorithm 6 is never more than the norm of $\mathfrak{a}$ considered by Algorithm 4. The net effect of all these considerations is that the asymptotic result obtained in [69] for Algorithm 4 also holds for Algorithm 6. The advantage of Algorithm 6 over Algorithm 4 is in improved practical efficiency.

8.3 Implementation

We have implemented Algorithms 4 and 6 using Magma, version V2.22-3. We did not perform the entire class group computation. Rather, we performed two kinds of experiments. The set of experiments compares the times required for relation collection by Algorithms 4 and 6. These experiments show that in general Algorithm 6 performs better than Algorithm 4. The second set of experiments performs the entire relation collection step for two fields having discriminants of sizes about 157 and 256 bits. This demonstrates that Algorithm 6 can actually work in practice.

Various issues arise while implementing Algorithm 4. For guidance in determining parameters, we considered the asymptotic analysis. This, however, fails to provide sufficient information and in some cases lead to substantially less efficient parameter choices. Below, we mention these issues as they arise in the description of our implementation effort.

8.3.1 Choice of Number Fields

Gélin and Joux [71] provide the following classification of number fields. Let $n_0 > 1$ be a real parameter arbitrarily close to 1, $d_0 > 0$, $\alpha \in [0, 1]$ and $\gamma \geq (1 - \alpha)$. The class $\mathcal{D}_{n_0,d_0,\alpha,\gamma}$ is defined to be the set of all number fields $\mathcal{K}$ with discriminant $\Delta_\mathcal{K}$ having a monic defining polynomial $T \in \mathbb{Z}[X]$ of degree n such the following inequalities hold true.

$$\frac{1}{n_0}\left(\frac{\log(|\Delta_\mathcal{K}|)}{\log(\log(|\Delta_\mathcal{K}|))}\right)^\alpha \leq n \leq n_0\left(\frac{\log(|\Delta_\mathcal{K}|)}{\log(\log(|\Delta_\mathcal{K}|))}\right)^\alpha \tag{8.4}$$

and

$$\log(\|T\|_\infty) \quad \leq \quad d_0(\log(|\Delta_\mathcal{K}|)^\gamma \log(\log(|\Delta_\mathcal{K}|))^{(1-\gamma)}. \tag{8.5}$$

Here, $\|T\|_\infty$ denotes the maximum of the absolute values of the coefficients of T. This parameter has also been called the height of the polynomial T. It has been shown in Gélin [69] that asymptotically the classification covers all number fields.

The actual number fields that we have used have been chosen from the online database of number fields [92]. We considered four number fields defined by the following four polynomials.

$$\begin{aligned}
T_1(X) &= X^{10} - 20X^8 - 170X^6 - 1704X^5 - 2100X^4 - 1680X^3 - 23865X^2 - 36360X + 15984,\\
T_2(X) &= X^{15} - 15X^{13} + 105X^{11} - 78X^{10} - 425X^9 + 780X^8 + 1050X^7 - 3510X^6 - 2832X^5\\
&\quad +7800X^4 + 7660X^3 - 7800X^2 - 13320X - 8856,\\
T_3(X) &= X^{20} - 5X^{19} + 76X^{18} - 247X^{17} + 1197X^{16} - 8474X^{15} + 15561X^{14} - 112347X^{13} + 325793X^{12}\\
&\quad -787322X^{11} + 3851661X^{10} - 5756183X^9 + 20865344X^8 - 48001353X^7 + 45895165X^6\\
&\quad -245996344X^5 + 8889264X^4 - 588303992X^3 - 54940704X^2 - 538817408X + 31141888,\\
T_4(X) &= X^{25} - 344X^{23} - 316X^{22} + 45906X^{21} + 78964X^{20} - 3003991X^{19} - 7163070X^{18} + 101409224X^{17}\\
&\quad +293673294X^{16} - 1740399699X^{15} - 5640650024X^{14} + 15351660849X^{13} + 53959254132X^{12}\\
&\quad -67237888386X^{11} - 259371867838X^{10} + 117450950109X^9 + 587040491084X^8 + 30969137155X^7 - 547923508138X^6\\
&\quad -206267098153X^5 + 109439981776X^4 + 40995170780X^3 - 9046378504X^2 - 1197994128X + 80434784.
\end{aligned}$$

Given a number field, the values of n_0, d_0, α and γ required in the Gélin-Joux classification are not unique. The definition mentions that $n_0 > 1$ and also that n_0 is arbitrarily close to 1. For a concrete number field, interpreting the latter condition is difficult. In fact, the values of n_0 and α need to be simultaneously determined so that (8.4) holds. Another important issue is that for $\alpha \in (3/4, 1]$, the value of α determines the complexity of the algorithm as can be

field	poly	n_0	α	$\log_2 \lvert\Delta\rvert$
$\mathcal{K}_1$	$T_1(X)$	1.10	0.935	63.5
$\mathcal{K}_2$	$T_2(X)$	1.11	0.990	81.5
$\mathcal{K}_3$	$T_2(X)$	1.01	0.952	157.2
$\mathcal{K}_4$	$T_2(X)$	1.01	0.911	256.2

TABLE 8.1: The number fields used in this chapter along with the values of n_0 and α used in the classification from [69]. Also, the size of the discriminant is shown for each field.

seen from the second row of Table 8.2. The lower the value of α, the lower the complexity. So, there is a motivation to choose n_0 and α such that α is as small as possible. It may be noted from (8.4) that if we assume that n_0 equals 1, then the value of α is equal to $\log n/\nu$, with $\nu = \log|\Delta_\mathcal{K}|/\log(\log|\Delta_\mathcal{K}|)$. The definition, however, requires $n_0 > 1$; further, the value of $\log n/\nu$ can turn out to be greater than 1 as is the case for the field $\mathcal{K}_3$ shown in Table 8.1. So, to determine the values of α and n_0, one may use the value $\log n/\nu$ as a starting guideline and then try to reduce the value of α as much as possible. We have followed this strategy. Additionally, we tried to determine n_0 and α such that the only integer in the range determined by the bounds in (8.4) is n, since this implies that the pair (α, n_0) uniquely determines n. It was possible to achieve this condition for the values of $n = 10, 20$ and 25; for $n = 15$, however, the range includes the integers 13, 14 and 15 and we found it difficult to tune the values of n_0 and α such that only 15 is in the desired range. The values of n_0, α and the size of the discriminant for the four number fields are shown in Table 8.2.

For the classification of the number fields, the values of d_0 and γ are also required to be determined. The conditions on these two parameters are that $d_0 > 0$ and $\gamma \geq 1 - \alpha$. From Table 8.2, the complexity of the algorithm does not depend on the values of d_0 and γ. So, one simple method of determining d_0 and γ is to fix the value of γ to be equal to $1 - \alpha$ and choose d_0 to be the least integer satisfying from (8.5). There are other strategies such as setting $d_0 = 1$ and then determining γ from (8.5). This strategy, however, does not necessarily ensure that $\gamma \geq 1 - \alpha$, though for the chosen fields, this condition holds. Since the values of d_0 and γ do not affect the complexity of the algorithm, we do not provide the values of these parameters.

8.3.2 Determining B and β

For a number field $\mathcal{K}$, based on the value of α, Gélin [69] provides the appropriate choices of β and B and the corresponding complexities. These are shown in Table 8.2. Note that $\alpha \geq 1/2$ in Table 8.2. In [69], number fields for which $\alpha \geq 1/2$ have been termed large degree number fields.

To implement Algorithm 4 (and also Algorithm 6), it is necessary to fix the values of β and B. For a given number field $\mathcal{K}$, the value of $|\Delta_\mathcal{K}|$ is known. Using this value of $|\Delta_\mathcal{K}|$ in the

α	β	c_b	B	complexity of reln coll	overall complexity
$[1/2, 3/4]$	$(\log \lvert\Delta_{\mathcal{K}}\rvert)^{1/2}$	$\frac{1}{2\sqrt{\omega}}$	$L_{\lvert\Delta_{\mathcal{K}}\rvert}\left(\frac{1}{2}, c_b\right)$	$L_{\lvert\Delta_{\mathcal{K}}\rvert}\left(\frac{1}{2}, \frac{1}{4c_b} + c_b\right)$	$L_{\lvert\Delta_{\mathcal{K}}\rvert}\left(\frac{1}{2}, \frac{\omega+1}{2\sqrt{\omega}}\right)$
$(3/4, 1]$	$(\log \lvert\Delta_{\mathcal{K}}\rvert)^{2\alpha/3}$	$\sqrt{\frac{2\alpha c}{3}}$	$L_{\lvert\Delta_{\mathcal{K}}\rvert}\left(\frac{2\alpha}{3}, c_b\right)$	$L_{\lvert\Delta_{\mathcal{K}}\rvert}\left(\frac{2\alpha}{3}, 2c_b\right)$	$L_{\lvert\Delta_{\mathcal{K}}\rvert}\left(\frac{2\alpha}{3}, o(1)\right)$

TABLE 8.2: Choices of β and B and the corresponding asymptotic complexities provided in [69]. In the table, ω is the exponent of linear algebra and determination of c is explained in Section 5.2 of [69].

expressions for β and B given in Table 8.2, it is possible to find concrete values of β and B. For the number fields that we have used, the value of α lies in the range $(3/4, 1]$.

We would like to highlight that asymptotically speaking B is much larger than C. The value of C is $12(\log |\Delta_{\mathcal{K}}|)^2$ which is $L_{|\Delta_{\mathcal{K}}|}(0, \cdot)$, whereas B is $L_{|\Delta_{\mathcal{K}}|}\left(\frac{2\alpha}{3}, c_b\right)$. Below we discuss the several issues related to using the asymptotic expression in deriving an appropriate value of B for concrete number fields. To do this, we need to evaluate the L-notation for concrete values of $|\Delta_{\mathcal{K}}|$ and the arguments. From (1.3), we note that there is an $o(1)$ term in the definition of the sub-exponential notation. We take this term to be 0.

Determining the value of B for $\alpha \in (3/4, 1]$ requires determining the value of c. The asymptotic analysis in [69] mentions that c can be taken to be any positive value which is very close to 0. In the concrete setting, however, a low value of c (such as 10^{-3}) leads to the size of the factor basis being about one or two for all the fields in Table 8.1, which is useless. Since the asymptotic analysis does not help in determining the value of c for a concrete setting, somewhat arbitrarily, one may set the value of c to be 1 for determining the value of B. The corresponding values of B are shown in Table 8.3. In this context, we note that the value of c can be chosen so that the value of B becomes close to the value of C.

- For $\mathcal{K}_1$, choosing $c = 0.53$ leads to B being 22921, while C is 23222.
- For $\mathcal{K}_2$, choosing $c = 0.32$ leads to B being 39459, while C is 38262.
- For $\mathcal{K}_3$, choosing $c = 0.18$ leads to B being 117218, while C is 142426.
- For $\mathcal{K}_4$, choosing $c = 0.138$ leads to B being 373691, while C is 378313.

The value of β obtained from Table 8.2 is denoted by β_{th}. It turns out that the value of β_{th} is equal to n for $n = 15$ and 20; is equal to $n - 1$ for $n = 25$; and equal to $n + 1$ for $n = 10$. Since β_{th} is the block size, its value is at most n, so for $n = 10$, the value of β_{th} should be 10. The values of β_{th} indicate that for the fields under consideration, the $\text{BKZ}_{\beta_{th}}$ reduction returns a smallest vector in the lattice. The values of β_{th}, B as well as the values of C are shown in Table 8.3 for the various number fields in Table 8.1.

The values of B and β_{th} shown in Table 8.3 are obtained from the asymptotic analysis. There are significant difficulties in implementing Algorithm 4 using these values of B and β_{th}.

field	β_{th}	B	C
$\mathcal{K}_1$	10	$970357 \approx 2^{19.9}$	$23222 \approx 2^{14.5}$
$\mathcal{K}_2$	15	$132526880 \approx 2^{27.0}$	$38262 \approx 2^{15.2}$
$\mathcal{K}_3$	20	$883661511529 \approx 2^{39.7}$	$142426 \approx 2^{17.1}$
$\mathcal{K}_4$	24	$997850477380847 \approx 2^{49.8}$	$378313 \approx 2^{18.5}$

TABLE 8.3: Values of β_{th}, B and Bach's bound C for the number fields in Table 8.1. For the computation of B, the value of c in Table 8.2 has been assumed to be 1.

1. For running BKZ$_\beta$, Magma has a limitation. Our experiments show that while Magma is able to execute BKZ$_\beta$ for $\beta = 2$, for higher values of β this is not ensured. In several runs, we have seen that the Magma is sometimes able to perform the computation and sometimes the process terminates abnormally. Further, in the cases that Magma did complete the execution, the norms of the smallest elements returned by BKZ$_\beta$ for various values of $\beta > 2$ turned out to be similar in size of the norms of the smallest elements returned by BKZ$_2$. In view of this, we decided to fix $\beta = 2$ for the actual relation collection.

2. The construction of the ideal $\mathfrak{a}$ in Algorithm 4 requires randomly choosing k prime ideals from the factor basis. To be able to make such choices, it is required to generate and store the entire factor basis before Algorithm 4 can be run. For $\mathcal{K}_2$, $\mathcal{K}_3$ and $\mathcal{K}_4$, the size of the factor basis is quite large (for the value of B obtained by setting $c = 1$). Magma is unable to construct such a large factor basis. In particular, the factor bases for $\mathcal{K}_3$ and $\mathcal{K}_4$ are too large to be stored on our systems. We next discuss how we have handled this difficulty.

As mentioned earlier, it has been proved in [69] that the norm of the ideal $\mathfrak{b}$ in Algorithm 4 satisfies the bound $\mathcal{N}(\mathfrak{b}) \leq \mathfrak{B}_\beta = \beta^{n(n-1)/(2(\beta-1))}\sqrt{|\Delta_\mathcal{K}|}$. A standard heuristic is that the probability $\mathcal{P}(x, y)$ that an ideal of norm bounded by x is y-smooth satisfies $\mathcal{P}(x, y) \geq \exp(-u(\log u)(1 + o(1)))$, where $u = \log x/\log y$. In [69], this heuristic is used in the asymptotic analysis of Algorithm 4 where x is taken to be equal to $\mathfrak{B}_\beta$ and y is taken to be equal to B. The expression $\mathfrak{B}_\beta$ is an *upper bound* on the norm of $\mathfrak{b}$. We ran some experiments to check the tightness of this bound. The experiments consisted of running the algorithm for 1000 iterations and noting the norms of $\mathfrak{b}$. For all the fields that we considered, it turned out that the actual ideals $\mathfrak{b}$ that are generated have norms which are much less than the upper bound. Since the norms are significantly lower, we decided to check the smoothness probabilities $\mathcal{P}(x, y)$ where x is the maximum norm that was experimentally observed and y is equal to C. It turned out that these smoothness probabilities are sufficiently high for relation collection to proceed.

In Table 8.4, for the various fields considered in this chapter, we provide the values of the upper bound $\mathfrak{B}_\beta$ on the norm of $\mathfrak{b}$ for $\beta = 2$ and $\beta = \beta_{th}$, the corresponding smoothness probabilities $\mathcal{P}(\mathfrak{B}_\beta, B)$, the experimentally observed maximum norms (denoted as $\mathfrak{M}$) of the ideal $\mathfrak{b}$ and the corresponding smoothness probabilities $\mathcal{P}(\mathfrak{M}, C)$. The values of B used in the

field	$(\mathfrak{M},\ \mathcal{P}(\mathfrak{M}, C))$	$(\beta,\ \mathfrak{B}_\beta,\ \mathcal{P}(\mathfrak{B}_\beta, B))$	N_b
$\mathcal{K}_1$	$(2^{16}, 0.897)$	$(2, 2^{76.75}, 2^{-7.51})$ $(10, 2^{48.36}, 0.116)$	2630
$\mathcal{K}_2$	$(2^{15}, \approx 1)$	$(2, 2^{145.75}, 2^{-13.13})$ $(15, 2^{70.05}, 0.084)$	4150
$\mathcal{K}_3$	$(2^{42}, 0.110)$	$(2, 2^{268.60}, 2^{-18.66})$ $(20, 2^{121.82}, 0.032)$	13097
$\mathcal{K}_4$	$(2^{74}, 0.004)$	$(2, 2^{428.10}, 2^{-26.68})$ $(24, 2^{187.90}, 0.007)$	32385

TABLE 8.4: Various smoothness probabilities and the number of ideals with norms at most C for the fields considered in this chapter.

computation of $\mathcal{P}(\mathfrak{B}_\beta, B)$ are from Table 8.3, which corresponds to $c = 1$. As discussed earlier, by suitably choosing the value of c, the value of B can be made close to the value of C. The corresponding smoothness probabilities $\mathcal{P}(\mathfrak{B}_\beta, C)$ are very low. If the norms of the ideals $\mathfrak{b}$ are indeed close to the upper bound $\mathfrak{B}_\beta$, then using C as the smoothness bound will make relation collection very inefficient.

From Table 8.4, it may be noted that the value of $\mathfrak{M}$ is substantially lower than the value of $\mathfrak{B}_\beta$ for both $\beta = 2$ and $\beta = \beta_{th}$. For $n = 10$ and $n = 15$, the value of $\mathfrak{M}$ is close to the value of C so that $\mathcal{P}(\mathfrak{M}, C)$ is very close to one. For $n = 20$ and $n = 25$, the value of $\mathcal{P}(\mathfrak{M}, C)$ is close to the value of $\mathcal{P}(\mathfrak{B}_{\beta_{th}}, B)$. Based on the values in Table 8.4, we may conclude that if relation collection is done with the factor basis as the prime ideals whose norms are below Bach's bound, then in practice, the smoothness probability is sufficiently high for relation collection to be possible. Due to this, we decided to proceed with C as the smoothness bound in Algorithm 4 instead of B which is obtained from Table 8.3. Note that this conclusion is for the number fields that have been considered in this chapter. Whether such an observation would hold in general for larger number fields is not clear. Nonetheless, our experiments indicate that even for larger number fields, it would be worthwhile to experimentally obtain $\mathfrak{M}$ and then decide whether B can be taken to equal to C, or, whether it needs to be chosen to be greater than C, and if so, how much larger. Using the upper bound on the norm of $\mathfrak{b}$ as the guideline for choosing B (as has been done in the asymptotic analysis [69]), may lead to much larger factor basis and a significantly less efficient algorithm.

For the number fields that we have chosen, it has turned out that using C as the smoothness bound is sufficient in practice. As noted earlier, in asymptotic terms B is much larger than C. So, for larger fields, choosing B to be equal to C will perhaps not be appropriate. Further experiments with larger fields are required to determine how much larger should B be compared to C.

8.3.3 Fixation of Parameters for Algorithms 4 and 6

The parameters k and A need to be fixed for Algorithm 4. Gélin [69] specifies these two parameters to be $\text{poly}\log|\Delta_{\mathcal{K}}|$. In practice, we need concrete values of these parameters. As the values of A and k increase, the norms of the ideals $\mathfrak{a}$ in Algorithm 4 also increase. This causes certain difficulties in the Magma implementation. The lattice $\sigma(\mathfrak{a})$ needs to be constructed and then the BKZ reduction needs to be performed on this lattice. Both of these steps crucially depend on the value of the precision used by Magma for the computations. From the experiments, we have the following observations regarding the issue of precision.

1. For a particular value of the precision, lattice construction is possible for ideals having norms below a certain bound. We could not, however, determine the relationship between the precision and the bound on norm. The general observation we have is that by increasing precision, it becomes possible to perform lattice construction for ideals with larger norms.

2. For the BKZ reduction to be possible, Magma requires the Gram matrix to be positive definite. The check for positive definiteness requires a higher precision than the precision required for the lattice construction.

3. Increasing precision slows down the computations.

In view of the above issues, choosing high values of A and k lead to abnormal termination of the Magma programs. So, we set A to be equal to 2 and did some experiments with varying precision to determine a suitable value of k. Finally, we set $k = 15$. The corresponding precision for lattice construction was fixed to be 2000 and the precision for the positive definiteness check on the gram matrix was set to be 6000.

For Algorithm 6, the values of the parameters κ_0 and λ need to be determined. Additionally, the values of κ and R which determine the size of the table $\mathfrak{T}$ also need to be fixed. Note that the number of rows in $\mathfrak{T}$ is about R/κ times N_b. We have chosen $R = 2$ and $\kappa = 4$ so that the number of pre-computed ideals stored in $\mathfrak{T}$ is about half of N_b. The value κ_0 has been set to 2 and we have considered the start points of the walks to be products of two ideals in $\mathfrak{C}$. Recall that based on our experiments, we have fixed the factor basis itself to be $\mathcal{C}$ and so using products of a pair of ideals in $\mathcal{C}$ provides sufficiently many walks. As mentioned earlier, the value of λ needs to be fixed so as to ensure that the maximum norm of the ideals visited by a walk is around the same value as the norms of the ideals $\mathfrak{a}$ generated in Algorithm 4. Since, we fixed k to be equal to 15, we fixed $\lambda = 8$. We conducted several experiments to confirm that with $\lambda = 8$, there was no failure in either the lattice construction, or the BKZ-reduction. The precision used for the lattice construction was 2000, which is the same as that used for Algorithm 4. Since the initial experiments indicated that there is no failure when $\lambda = 8$, we did not perform an explicit check for the positive definiteness of the gram matrix.

The hash function $\mathcal{H}$ used to access entries of the pre-computed table $\mathfrak{T}$ should ensure a more or less uniform distribution over the entries of the table. In our implementations, we have used the in-built hash function of Magma reduced modulo m (the size of the pre-computed table) to instantiate $\mathcal{H}$. We have also experimented with other hash functions defined using simple arithmetic. There was no significant change in the results.

8.3.4 Experimental Set-Up and Timing Results

All our computations were done on a server which has the configuration of Intel Xeon E7-8890 @ 2.50 GHz with 72 physical cores and 144 logical cores. As mentioned earlier, the computations were done using Magma version V2.22-3. We conducted two sets of experiments.

The goal of the first set of experiments is to compare the performances of Algorithms 4 and 6. Since we have set the smoothness bound B to be equal to C, the number of ideals in the factor basis is N_b. For the four fields, we generated the factor basis and determined N_b. The values of N_b corresponding to the fields are shown in Table 8.4. The number of relations required is a little more than the size of the factor basis. Since N_b is quite small for $\mathcal{K}_1$ and $\mathcal{K}_2$, we decided to generate a complete set of relations for these two fields using both Algorithms 4 and 6; for $\mathcal{K}_1$, we generated 3000 relations while for $\mathcal{K}_2$, we generated 5000 relations. In comparison, for $\mathcal{K}_3$ and $\mathcal{K}_4$, N_b is a little higher, so, for these two fields we decided to compare the performances of the two algorithms for collecting 5000 relations. For each of the experiments, 25 processes were run in parallel. For $\mathcal{K}_1$, each of the processes was tasked with collecting 120 relations, while for the other three fields, each of the processes was tasked with collecting 200 relations. For all of the fields, all the relations obtained by both Algorithms 4 and 6 were distinct. In Table 8.7, we provide the total number of iterations and the total times (in seconds) required by all the processes for both the algorithms.

From Table 8.5, it is to be noted that the total number of iterations required by Algorithm 6 is slightly more than that required by Algorithm 4. Recall that in both Algorithms 4 and 6, a relation is obtained whenever the ideal $\mathfrak{b}$ is smooth over the factor basis. An important parameter in the assessment of the complexity is the probability of smoothness. Empirically, the total number of relations collected divided by the total number of iterations required is an estimate of the smoothness probability of $\mathfrak{b}$. These estimates are shown in Table 8.6. The probabilities corresponding to Algorithm 4 are slightly higher. It is perhaps useful to compare these probabilities with the probability estimates given in Table 8.4. While for $\mathcal{K}_1$ and $\mathcal{K}_2$, the empirical estimates in Table 8.6 are close to $\mathcal{P}(\mathfrak{M}, C)$, for $\mathcal{K}_3$ and $\mathcal{K}_4$, the empirical estimates in Table 8.6 are substantially larger. The reason for this is the fact that in the computation $\mathcal{P}(\mathfrak{M}, C)$, $\mathfrak{M}$ has been taken to be the maximum of the norms of the ideals $\mathfrak{b}$ generated in about 1000 trials. The average of the norms is substantially lower than the maximum, so that $\mathcal{P}(\mathfrak{M}, C)$ is a substantial underestimate of the probability of smoothness of $\mathfrak{b}$.

field	Algorithm 4		Algorithm 6		
	# iter	time (t_1)	# iter	time (t_3)	t_1/t_3
$\mathcal{K}_1$	3004	876	3385	266	3.29
$\mathcal{K}_2$	5007	4024	5642	1405	2.86
$\mathcal{K}_3$	8370	17702	9285	5903	3.00
$\mathcal{K}_4$	165840	717386	183631	211381	3.39

TABLE 8.5: Comparison of Algorithms 4 and 6.

Algorithm	$\mathcal{K}_1$	$\mathcal{K}_2$	$\mathcal{K}_3$	$\mathcal{K}_4$
Algorithm 4	≈ 1.00	≈ 1.00	≈ 0.60	≈ 0.03
Algorithm 6	≈ 0.89	≈ 0.89	≈ 0.54	≈ 0.03

TABLE 8.6: Empirical estimates of the smoothness probabilities for Algorithms 4 and 6 obtained from Table 8.5.

The main point to observe from Table 8.5 is the ratio t_1/t_3. This figure varies from 2.86 to 3.39 indicating that in practice, Algorithm 6 is about three times faster than Algorithm 4. The speed-up factor of three is determined by the choice of the parameters for Algorithms 4 and 6 mentioned earlier. A different choice of parameters may lead to a different speed-up factor. As explained earlier, the main advantage of Algorithm 6 over Algorithm 4 is the reduction in the number of ideal multiplications in the generation of the ideal $\mathfrak{a}$. The main point of the experiments is to show that this advantage can indeed be realised in practice.

The first set of experiments provide evidence that in practice Algorithm 6 performs better than Algorithm 4. A second set of experiments was conducted to demonstrate that Algorithm 6 can indeed generate the full set of relations for reasonable size number fields. To this end, we used Algorithm 6 to collect 15000 relations for $\mathcal{K}_3$ and 35000 relations for $\mathcal{K}_4$. For $\mathcal{K}_3$, we ran 75 processes each tasked with collecting 200 relations while for $\mathcal{K}_4$, we ran 70 processes each tasked with collecting 500 relations. In the case of $\mathcal{K}_3$ all the obtained relations were distinct, while for $\mathcal{K}_4$, 34994 relations were distinct. The total number of iterations and the total times required by all the processes for these experiments are shown in Table 8.7.

From Table 8.5, we note that the estimated probabilities of smoothness of the ideal $\mathfrak{b}$ (i.e., the number of relations divided by the number of iterations) are 0.53 and 0.03 for $\mathcal{K}_3$ and $\mathcal{K}_4$ respectively. These are close to the probability estimates given in Table 8.6 for Algorithm 6. The main point of the second set of experiments is to show that Algorithm 6 can indeed be used to generate a complete set of relations.

The time to generate the pre-computed table $\mathfrak{T}$ has not been considered in either Table 8.5 or 8.7. We would like to highlight that for a reasonable size number field, the time to generate the full set of relations is substantially higher than the time to generate $\mathfrak{T}$. Instead of comparing the times, we compare the number of ideal multiplications required for the two tasks in the case

field	# iter	time
$\mathcal{K}_3$	28162	22937
$\mathcal{K}_4$	1268434	1865071

TABLE 8.7: Number of iterations and times (in seconds) to collect 15000 relations for $\mathcal{K}_3$ and 35000 relations for $\mathcal{K}_4$ using Algorithm 6.

of $\mathcal{K}_4$. From Table 8.7, the number of iterations required to collect 35000 relations is 1268434. So, Algorithm 6 considers this number of ideals $\mathfrak{a}$. The generation of the start ideal of each walk requires a single ideal multiplication and each step of the walk also requires a single ideal multiplication. So, the number of ideal multiplications to generate 1268434 ideals is also 1268434. Now, consider the generation of $\mathfrak{T}$. The number of entries in $\mathfrak{T}$ is $16192 = q \times R$, where $q = 8096$ and $R = 2$. Of the 16192 ideals in $\mathfrak{T}$, 16190 ideals are products of 4 primes ideals from $\mathcal{C}$, while 2 ideals are products of 5 prime ideals from $\mathcal{C}$. So, the total number of ideal multiplications required to generate the ideals in $\mathfrak{T}$ $16190 \times 3 + 2 \times 4 = 48578$, which is about 3.8% of the number of ideal multiplications required to generate all the ideals. Consequently, even if the time for the generation of $\mathfrak{T}$ is taken into consideration, Algorithm 6 will perform better than Algorithm 4 for collecting a complete set of relations for a large enough number field.

Simple Setting of Parameters. For the previously mentioned experiments, the walk length in Algorithm 6 was set to 8 and the start points were products of two ideals in $\mathcal{C}$. Given that the concrete number fields are not too large, we decided to experiment with relation collection using Algorithm 6 with walk length 1. This means that each walk visits only one ideal which is the start point, where each start point is a product of two ideals. With this setting, Algorithm 6 essentially becomes the same as Algorithm 4 with $k = 2$ and $A = 1$. Since the setting was simple, we also reduced the precision to the default precision for Magma which is 167. We wished to find out whether such a basic setting is adequate for collecting relations.

We ran the relation collection processes for $\mathcal{K}_3$ and $\mathcal{K}_4$ as before with the goal of collecting 15000 and 35000 relations respectively. The number of iterations required were 24901 and 970114; and the number of distinct relations obtained were 14999 and 34904 respectively. The number of distinct relations obtained for $\mathcal{K}_3$ was adequate, but for $\mathcal{K}_4$ there was a sharp drop from the target of 35000 relations. The required times for $\mathcal{K}_3$ and $\mathcal{K}_4$ were 5419 and 295533 seconds respectively. Note that compared to the times in Table 8.7, there is a marked improvement in the times.

So, for the number fields that have been considered, the above mentioned simple setting leads to improved times, though for $\mathcal{K}_4$ the number of distinct relations obtained are not sufficient and more relations would need to be obtained to make up the deficit. For larger number fields, however, it is unlikely that such a simple setting would be sufficient for generating the required

relations. A non-trivial walk length would be required in Algorithm 6 to be able to explore a larger portion of the ideal space.

8.4 Conclusion

In this chapter, we have introduced a technique to perform a pseudo-random walk over ideals. After the first step, each step of the walk requires a single ideal multiplication. The ideals visited by the walk are used for relation collection in exactly the same manner as used in Gélin's algorithm [69]. The practical advantage over Gélin's algorithm is the reduction in the number of ideal multiplications required to generate the next ideal to be tested. Our Magma implementations of both the new algorithm and Gélin's algorithm confirm that there is indeed a practical speed-up.

Chapter 9

Conclusion and Future Works

In this chapter we encapsulate the contributions of the thesis. Along with that, we try to indicate some works that may be undertaken in the future.
In this thesis, we have focussed mainly on application of index calculus strategies for the discrete logarithm problem and the class group computation problem. In case of DLP, we have performed a new record discrete logarithm computation in general medium characteristic fields, proposed a new algorithm for small characteristic composite extension degree fields to efficiently perform initial splitting which is a sub-step of the individual logarithm step and suggested ways to apply Montgomery multiplication to the tag tracing variant of Pollard's Rho algorithm applied to prime order fields. We have proposed an improved method for collecting relations when applying index calculus method to compute class groups of *large degree* number fields.
We try to identify some works that may be undertaken in this arena in near future. They are listed below along with the chapters.

9.1 Future Works

9.1.1 Performing New Record Discrete Log Computations in the General Medium Characteristic Fields Extending Methods Used in Chapter 5

Record discrete log computations can be tried using the same techniques as in the present record computation done by us [135]. As an example, we have already shown that for any 32-bit prime and moderate extension degree, the relation collection and descent phases can be done adopting the same methodology as in this paper. With proper resources to solve the bottleneck linear algebra phase, discrete log can be attempted for fields with 32-bit characteristic.

9.1.2 Discrete Log Computations over Small Characteristic Composite Extension Degree Fields Using Algorithm of Chapter 6

To attempt record discrete log computations over small characteristic (where *small characteristic* is defined in Section 4.1 of this thesis) composite extension degree fields in future, the new initial splitting algorithm [133] can be used in the individual logarithm step. Apart from this, further theoretical improvement of reduction of cost of initial splitting step for small characteristic composite extension degree fields may be tried. Possibility of an analogous algorithm with some tweaking for larger characteristic can be analysed.

9.1.3 Implementing Discrete Logarithm Computations Using Substitutions Suggested in Chapter 3

Given any target group, the tag tracing version of Pollard's Rho algorithm with Montgomery multiplications [132] wherever necessary, may be used to compute discrete logarithms modulo the smaller prime factors of the cardinality of the group. This would lead to a gain in time in those cases where a number of small prime factors are present.

9.1.4 Computing Class Groups of Number Fields with Large Discriminants Using the Walk in Chapter 8

Our algorithm can be used in the relation collection phase to compute class groups of *large degree* number fields. It would be particularly helpful for large fields where multiplications are costly. Due to the limitations of computational resources, experimental results for number fields with 512-bit and higher size discriminants using the new algorithm could not be reported. This would be an immediate target.

"***Somewhere, something incredible is waiting to be known.***"
-Carl Sagan.

Bibliography

[1] Gora Adj et al. "Computing Discrete Logarithms in $\mathbb{F}_{3^{6\cdot 137}}$ and $\mathbb{F}_{3^{6\cdot 163}}$ Using Magma". In: *Arithmetic of Finite Fields - 5th International Workshop, WAIFI 2014, Gebze, Turkey, September 27-28, 2014. Revised Selected Papers*. Ed. by Çetin Kaya Koç, Sihem Mesnager, and Erkay Savas. Vol. 9061. Lecture Notes in Computer Science. Springer, 2014, pp. 3–22. DOI: 10.1007/978-3-319-16277-5_1. URL: https://doi.org/10.1007/978-3-319-16277-5%5C_1.

[2] Gora Adj et al. "Weakness of $\mathbb{F}_{3^{6\cdot 509}}$ for Discrete Logarithm Cryptography". In: *PAIRING 2013*. Ed. by Zhenfu Cao and Fangguo Zhang. Vol. 8365. LNCS. Springer, Heidelberg, Nov. 2014, pp. 20–44. DOI: 10.1007/978-3-319-04873-4_2.

[3] Gora Adj et al. "Weakness of $\mathbb{F}_{6^{6\cdot 1429}}$ and $\mathbb{F}_{2^{4\cdot 3041}}$ for discrete logarithm cryptography". In: *Finite Fields Their Appl.* 32 (2015), pp. 148–170. DOI: 10.1016/j.ffa.2014.10.009. URL: https://doi.org/10.1016/j.ffa.2014.10.009.

[4] Leonard M. Adleman. "A Subexponential Algorithm for the Discrete Logarithm Problem with Applications to Cryptography (Abstract)". In: *20th Annual Symposium on Foundations of Computer Science, San Juan, Puerto Rico, 29-31 October 1979*. IEEE Computer Society, 1979, pp. 55–60. DOI: 10.1109/SFCS.1979.2. URL: https://doi.org/10.1109/SFCS.1979.2.

[5] Leonard M Adleman. "The function field sieve". In: *International Algorithmic Number Theory Symposium*. Springer. 1994, pp. 108–121.

[6] Leonard M Adleman, Jonathan DeMarrais, and Ming-Deh Huang. "A subexponential algorithm for discrete logarithms over hyperelliptic curves of large genus over GF(q)". In: *Theoretical Computer Science* 226.1-2 (1999), pp. 7–18.

[7] Leonard M Adleman, Jonathan DeMarrais, and Ming-Deh Huang. "A subexponential algorithm for discrete logarithms over the rational subgroup of the Jacobians of large genus hyperelliptic curves over finite fields". In: *International Algorithmic Number Theory Symposium*. Springer. 1994, pp. 28–40.

[8] Leonard M Adleman and Ming-Deh A Huang. "Function field sieve method for discrete logarithms over finite fields". In: *Information and Computation* 151.1-2 (1999), pp. 5–16.

[9] Robert B Ash. *A course in algebraic number theory*. Courier Corporation, 2010.

[10] László Babai. "On Lovász' lattice reduction and the nearest lattice point problem". In: *Comb.* 6.1 (1986), pp. 1–13. DOI: 10.1007/BF02579403. URL: https://doi.org/10.1007/BF02579403.

[11] Eric Bach. "Explicit bounds for primality testing and related problems". In: *Mathematics of Computation* 55.191 (1990), pp. 355–380.

[12] Eric Bach. "Improved approximations for Euler products". In: *Number theory*. 1995, pp. 13–28.

[13] Razvan Barbulescu. "Algorithms for discrete logarithm in finite fields". PhD thesis. Université de Lorraine, 2013.

[14] Razvan Barbulescu, Pierrick Gaudry, and Thorsten Kleinjung. "The Tower Number Field Sieve". In: *ASIACRYPT 2015, Part II*. Ed. by Tetsu Iwata and Jung Hee Cheon. Vol. 9453. LNCS. Springer, Heidelberg, Nov. 2015, pp. 31–55. DOI: 10.1007/978-3-662-48800-3_2.

[15] Razvan Barbulescu and Cécile Pierrot. "The multiple number field sieve for medium-and high-characteristic finite fields". In: *LMS Journal of Computation and Mathematics* 17.A (2014), pp. 230–246.

[16] Razvan Barbulescu et al. "A Heuristic Quasi-Polynomial Algorithm for Discrete Logarithm in Finite Fields of Small Characteristic". In: *EUROCRYPT 2014*. Ed. by Phong Q. Nguyen and Elisabeth Oswald. Vol. 8441. LNCS. Springer, Heidelberg, May 2014, pp. 1–16. DOI: 10.1007/978-3-642-55220-5_1.

[17] Razvan Barbulescu et al. "Discrete logarithm in $GF(2^{809})$ with FFS". In: *International Workshop on Public Key Cryptography*. Springer. 2014, pp. 221–238.

[18] Razvan Barbulescu et al. "Improving NFS for the Discrete Logarithm Problem in Non-prime Finite Fields". In: *Advances in Cryptology - EUROCRYPT 2015 - 34th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Sofia, Bulgaria, April 26-30, 2015, Proceedings, Part I*. Ed. by Elisabeth Oswald and Marc Fischlin. Vol. 9056. Lecture Notes in Computer Science. Springer, 2015, pp. 129–155. DOI: 10.1007/978-3-662-46800-5_6. URL: https://doi.org/10.1007/978-3-662-46800-5%5C_6.

[19] Jean-François Biasse. "An $L(1/3)$ algorithm for ideal class group and regulator computation in certain number fields". In: *Mathematics of Computation* 83.288 (2014), pp. 2005–2031.

[20] Jean-François Biasse. "Subexponential time relations in the class group of large degree number fields". In: *Advances in Mathematics of Communications* 8.4 (2014), p. 407.

[21] Jean-François Biasse and Claus Fieker. "Subexponential class group and unit group computation in large degree number fields". In: *LMS Journal of Computation and Mathematics* 17.A (2014), pp. 385–403.

[22] Ingrid Biehl, Johannes Buchmann, and Christoph Thiel. "Cryptographic Protocols Based on Discrete Logarithms in Real-quadratic Orders". In: *CRYPTO'94*. Ed. by Yvo Desmedt. Vol. 839. LNCS. Springer, Heidelberg, Aug. 1994, pp. 56–60. DOI: 10.1007/3-540-48658-5_7.

[23] Ingrid Biehl et al. "A signature scheme based on the intractability of computing roots". In: *Designs, Codes and Cryptography* 25.3 (2002), pp. 223–236.

[24] Ian F. Blake, Ronald C. Mullin, and Scott A. Vanstone. "Computing Logarithms in GF(2^n)". In: *CRYPTO'84*. Ed. by G. R. Blakley and David Chaum. Vol. 196. LNCS. Springer, Heidelberg, Aug. 1984, pp. 73–82.

[25] Ian F Blake et al. "Computing logarithms in finite fields of characteristic two". In: *SIAM Journal on Algebraic Discrete Methods* 5.2 (1984), pp. 276–285.

[26] Dan Boneh, Benedikt Bünz, and Ben Fisch. "Batching techniques for accumulators with applications to iops and stateless blockchains". In: *Annual International Cryptology Conference*. Springer. 2019, pp. 561–586.

[27] Dan Boneh and Matthew K. Franklin. "Identity-Based Encryption from the Weil Pairing". In: *CRYPTO 2001*. Ed. by Joe Kilian. Vol. 2139. LNCS. Springer, Heidelberg, Aug. 2001, pp. 213–229. DOI: 10.1007/3-540-44647-8_13.

[28] Dan Boneh, Ben Lynn, and Hovav Shacham. "Short Signatures from the Weil Pairing". In: *ASIACRYPT 2001*. Ed. by Colin Boyd. Vol. 2248. LNCS. Springer, Heidelberg, Dec. 2001, pp. 514–532. DOI: 10.1007/3-540-45682-1_30.

[29] Dan Boneh, Ben Lynn, and Hovav Shacham. "Short Signatures from the Weil Pairing". In: *J. Cryptol.* 17.4 (2004), pp. 297–319. DOI: 10.1007/s00145-004-0314-9. URL: https://doi.org/10.1007/s00145-004-0314-9.

[30] Joppe W. Bos and Peter L. Montgomery. *Montgomery Arithmetic from a Software Perspective*. Cryptology ePrint Archive, Report 2017/1057. https://eprint.iacr.org/2017/1057. 2017.

[31] Joppe W. Bos et al. "Solving a 112-bit prime elliptic curve discrete logarithm problem on game consoles using sloppy reduction". In: *Int. J. Appl. Cryptogr.* 2.3 (2012), pp. 212–228. DOI: 10.1504/IJACT.2012.045590. URL: https://doi.org/10.1504/IJACT.2012.045590.

[32] Fabrice Boudot et al. "Comparing the Difficulty of Factorization and Discrete Logarithm: A 240-Digit Experiment". In: *CRYPTO 2020, Part II*. Ed. by Daniele Micciancio and Thomas Ristenpart. Vol. 12171. LNCS. Springer, Heidelberg, Aug. 2020, pp. 62–91. DOI: 10.1007/978-3-030-56880-1_3.

[33] Cyril Bouvier. "Algorithmes pour la factorisation d'entiers et le calcul de logarithme discret". PhD thesis. 2015.

[34] Johannes Buchmann. "A subexponential algorithm for the determination of class groups and regulators of algebraic number fields". In: *Séminaire de théorie des nombres, Paris* 1989.1990 (1988), pp. 27–41.

[35] Johannes Buchmann. "The computation of the fundamental unit of totally complex quartic orders". In: *Mathematics of Computation* 48.177 (1987), pp. 39–54.

[36] Johannes Buchmann and S Düllmann. "A probabilistic class group and regulator algorithm and its implementation". In: *Computational number theory* (1991), pp. 53–72.

[37] Johannes Buchmann and Stephan Düllmann. "Distributed class group computation". In: *Informatik*. Springer, 1992, pp. 69–79.

[38] Johannes Buchmann and Stephan Düllmann. "On the computation of discrete logarithms in class groups". In: *Conference on the Theory and Application of Cryptography*. Springer. 1990, pp. 134–139.

[39] Johannes Buchmann and Hugh C. Williams. "A Key-Exchange System Based on Imaginary Quadratic Fields". In: *Journal of Cryptology* 1.2 (June 1988), pp. 107–118. DOI: 10.1007/BF02351719.

[40] Benedikt Bünz, Ben Fisch, and Alan Szepieniec. "Transparent SNARKs from DARK compilers". In: *Annual International Conference on the Theory and Applications of Cryptographic Techniques*. Springer. 2020, pp. 677–706.

[41] E Rodney Canfield, Paul Erdös, and Carl Pomerance. "On a problem of Oppenheim concerning "factorisatio numerorum"". In: *Journal of Number Theory* 17.1 (1983), pp. 1–28.

[42] Jae Choon Cha and Jung Hee Cheon. "An Identity-Based Signature from Gap Diffie-Hellman Groups". In: *PKC 2003*. Ed. by Yvo Desmedt. Vol. 2567. LNCS. Springer, Heidelberg, Jan. 2003, pp. 18–30. DOI: 10.1007/3-540-36288-6_2.

[43] Jung Hee Cheon, Jin Hong, and Minkyu Kim. "Accelerating Pollard's Rho Algorithm on Finite Fields". In: *J. Cryptol.* 25.2 (2012), pp. 195–242. DOI: 10.1007/s00145-010-9093-7. URL: https://doi.org/10.1007/s00145-010-9093-7.

[44] Henri Cohen. "A course in computational algebraic number theory". In: *Graduate texts in Math.* 138 (1993), p. 88.

[45] Henri Cohen. *A course in computational algebraic number theory*. Vol. 138. Springer Science & Business Media, 2013.

[46] Henri Cohen, F Diaz Y Diaz, and Michel Olivier. "Subexponential algorithms for class group and unit computations". In: *Journal of Symbolic Computation* 24.3-4 (1997), pp. 433–441.

[47] Henri Cohen and Hendrik W Lenstra. "Heuristics on class groups of number fields". In: *Number Theory Noordwijkerhout 1983*. Springer, 1984, pp. 33–62.

[48] Henri Cohen, Diaz Y Diaz, et al. "A polynomial reduction algorithm". In: *Journal de théorie des nombres de Bordeaux* 3.2 (1991), pp. 351–360.

[49] Don Coppersmith. "Fast evaluation of logarithms in fields of characteristic two". In: *IEEE Trans. Inf. Theory* 30.4 (1984), pp. 587–593. DOI: 10.1109/TIT.1984.1056941. URL: https://doi.org/10.1109/TIT.1984.1056941.

[50] Don Coppersmith. "Modifications to the Number Field Sieve". In: *Journal of Cryptology* 6.3 (Mar. 1993), pp. 169–180. DOI: 10.1007/BF00198464.

[51] Gabrielle De Micheli, Pierrick Gaudry, and Cécile Pierrot. "Asymptotic Complexities of Discrete Logarithm Algorithms in Pairing-Relevant Finite Fields". In: *CRYPTO 2020, Part II*. Ed. by Daniele Micciancio and Thomas Ristenpart. Vol. 12171. LNCS. Springer, Heidelberg, Aug. 2020, pp. 32–61. DOI: 10.1007/978-3-030-56880-1_2.

[52] Jérémie Detrey, Pierrick Gaudry, and Marion Videau. "Relation Collection for the Function Field Sieve". In: *21st IEEE Symposium on Computer Arithmetic, ARITH 2013, Austin, TX, USA, April 7-10, 2013*. Ed. by Alberto Nannarelli, Peter-Michael Seidel, and Ping Tak Peter Tang. IEEE Computer Society, 2013, pp. 201–210. DOI: 10.1109/ARITH.2013.28. URL: https://doi.org/10.1109/ARITH.2013.28.

[53] Claus Diem and Sebastian Kochinke. "Computing discrete logarithms with special linear systems". In: *preprint* (2013).

[54] Whitfield Diffie and Martin E. Hellman. "New Directions in Cryptography". In: *IEEE Transactions on Information Theory* 22.6 (1976), pp. 644–654.

[55] Taher ElGamal. "A Public Key Cryptosystem and a Signature Scheme Based on Discrete Logarithms". In: *CRYPTO'84*. Ed. by G. R. Blakley and David Chaum. Vol. 196. LNCS. Springer, Heidelberg, Aug. 1984, pp. 10–18.

[56] Andreas Enge. "Computing discrete logarithms in high-genus hyperelliptic Jacobians in provably subexponential time". In: *Mathematics of Computation* 71.238 (2002), pp. 729–742.

[57] Andreas Enge, Pierrick Gaudry, and Emmanuel Thomé. "An $L(1/3)$ Discrete Logarithm Algorithm for Low Degree Curves". In: *Journal of Cryptology* 24.1 (Jan. 2011), pp. 24–41. DOI: 10.1007/s00145-010-9057-y.

[58] Amos Fiat and Adi Shamir. "How to Prove Yourself: Practical Solutions to Identification and Signature Problems". In: *CRYPTO'86*. Ed. by Andrew M. Odlyzko. Vol. 263. LNCS. Springer, Heidelberg, Aug. 1987, pp. 186–194. DOI: 10.1007/3-540-47721-7_12.

[59] Claus Fieker. "Minimizing representations over number fields". In: *Journal of Symbolic Computation* 38.1 (2004), pp. 833–842.

[60] Philippe Flajolet, Xavier Gourdon, and Daniel Panario. "The Complete Analysis of a Polynomial Factorization Algorithm over Finite Fields". In: *J. Algorithms* 40.1 (2001), pp. 37–81. DOI: 10.1006/jagm.2001.1158. URL: https://doi.org/10.1006/jagm.2001.1158.

[61] Joshua Fried et al. "A Kilobit Hidden SNFS Discrete Logarithm Computation". In: *Advances in Cryptology - EUROCRYPT 2017 - 36th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Paris, France, April 30 - May 4, 2017, Proceedings, Part I*. Ed. by Jean-Sébastien Coron and Jesper Buus Nielsen. Vol. 10210. Lecture Notes in Computer Science. 2017, pp. 202–231. DOI: 10.1007/978-3-319-56620-7_8. URL: https://doi.org/10.1007/978-3-319-56620-7%5C_8.

[62] Steven Galbraith. *Quasi-polynomial-time algorithm for discrete logarithm in finite fields of small/medium characteristic. The Elliptic Curve Cryptography blog, June 2013*. 2001.

[63] Steven D Galbraith. *Mathematics of public key cryptography*. Cambridge University Press, 2012.

[64] Steven D Galbraith and Pierrick Gaudry. "Recent progress on the elliptic curve discrete logarithm problem". In: *Designs, Codes and Cryptography* 78.1 (2016), pp. 51–72.

[65] Pierrick Gaudry. "An Algorithm for Solving the Discrete Log Problem on Hyperelliptic Curves". In: *EUROCRYPT 2000*. Ed. by Bart Preneel. Vol. 1807. LNCS. Springer, Heidelberg, May 2000, pp. 19–34. DOI: 10.1007/3-540-45539-6_2.

[66] Pierrick Gaudry et al. "A double large prime variation for small genus hyperelliptic index calculus". In: *Mathematics of computation* 76.257 (2007), pp. 475–492.

[67] Carl Friedrich Gauss. *Disquisitiones arithmeticae*. Yale University Press, 1966.

[68] KF Gauss, Disquisitiones Arithmeticae, and Fleischer Leipzig. *Translation into English by Arthur A. Clarke*. 1986.

[69] Alexandre Gélin. "On the complexity of class group computations for large degree number fields". In: *arXiv preprint arXiv:1810.11396* (2018).

[70] Alexandre Gélin. "Reducing the complexity for class group computations using small defining polynomials". In: *arXiv preprint arXiv:1810.12010* (2018).

[71] Alexandre Gélin and Antoine Joux. "Reducing number field defining polynomials: An application to class group computations". In: *LMS Journal of Computation and Mathematics* 19.A (2016), pp. 315–331.

[72] Jay Goldman. *The queen of mathematics: a historically motivated guide to number theory*. CRC Press, 1997.

[73] Faruk Göloglu et al. "Discrete Logarithms in $GF(2^{1271})$". In: *NMBRTHRY list* 19 (2013).

[74] Faruk Göloglu et al. "On the Function Field Sieve and the Impact of Higher Splitting Probabilities - Application to Discrete Logarithms in and". In: *Advances in Cryptology - CRYPTO 2013 - 33rd Annual Cryptology Conference, Santa Barbara, CA, USA, August 18-22, 2013. Proceedings, Part II*. Ed. by Ran Canetti and Juan A. Garay. Vol. 8043. Lecture Notes in Computer Science. Springer, 2013, pp. 109–128. DOI: 10.1007/978-3-642-40084-1_7. URL: https://doi.org/10.1007/978-3-642-40084-1%5C_7.

[75] Faruk Göloglu et al. "Solving a 6120-bit DLP on a Desktop Computer". In: LNCS 8282 (Aug. 2014). Ed. by Tanja Lange, Kristin Lauter, and Petr Lisonek, pp. 136–152. DOI: 10.1007/978-3-662-43414-7_7.

[76] Daniel M. Gordon. "Discrete Logarithms in *GF(P)* Using the Number Field Sieve". In: *SIAM J. Discret. Math.* 6.1 (1993), pp. 124–138. DOI: 10.1137/0406010. URL: https://doi.org/10.1137/0406010.

[77] Daniel M. Gordon and Kevin S. McCurley. "Massively Parallel Computation of Discrete Logarithms". In: *CRYPTO'92*. Ed. by Ernest F. Brickell. Vol. 740. LNCS. Springer, Heidelberg, Aug. 1993, pp. 312–323. DOI: 10.1007/3-540-48071-4_22.

[78] Robert Granger, Thorsten Kleinjung, and Jens Zumbrägel. "Breaking '128-bit Secure' Supersingular Binary Curves". In: *Advances in Cryptology - CRYPTO 2014 - 34th Annual Cryptology Conference, Santa Barbara, CA, USA, August 17-21, 2014, Proceedings, Part II*. Ed. by Juan A. Garay and Rosario Gennaro. Vol. 8617. Lecture Notes in Computer Science. Springer, 2014, pp. 126–145. DOI: 10.1007/978-3-662-44381-1_8. URL: https://doi.org/10.1007/978-3-662-44381-1%5C_8.

[79] Robert Granger, Thorsten Kleinjung, and Jens Zumbrägel. "Discrete logarithms in $GF(2^{9234})$". In: *NMBRTHRY list, January* (2014).

[80] Robert Granger, Thorsten Kleinjung, and Jens Zumbrägel. *On the discrete logarithm problem in finite fields of fixed characteristic*. Cryptology ePrint Archive, Report 2015/685. https://eprint.iacr.org/2015/685. 2015.

[81] Robert Granger, Thorsten Kleinjung, and Jens Zumbrägel. *On the Powers of 2*. Cryptology ePrint Archive, Report 2014/300. https://eprint.iacr.org/2014/300. 2014.

[82] Robert Granger et al. *Computation of a 30750-Bit Binary Field Discrete Logarithm*. Cryptology ePrint Archive, Report 2020/965. https://eprint.iacr.org/2020/965. 2020.

[83] Aurore Guillevic. "Computing Individual Discrete Logarithms Faster in GF(p^n) with the NFS-DL Algorithm". In: *ASIACRYPT 2015, Part I*. Ed. by Tetsu Iwata and Jung Hee Cheon. Vol. 9452. LNCS. Springer, Heidelberg, Nov. 2015, pp. 149–173. DOI: 10.1007/978-3-662-48797-6_7.

[84] Aurore Guillevic. "Faster individual discrete logarithms in finite fields of composite extension degree". In: *Math. Comput.* 88.317 (2019), pp. 1273–1301. DOI: 10.1090/mcom/3376. URL: https://doi.org/10.1090/mcom/3376.

[85] James L Hafner and Kevin S McCurley. "A rigorous subexponential algorithm for computation of class groups". In: *Journal of the American mathematical society* 2.4 (1989), pp. 837–850.

[86] Safuat Hamdy and Bodo Möller. "Security of Cryptosystems Based on Class Groups of Imaginary Quadratic Orders". In: *ASIACRYPT 2000*. Ed. by Tatsuaki Okamoto. Vol. 1976. LNCS. Springer, Heidelberg, Dec. 2000, pp. 234–247. DOI: 10.1007/3-540-44448-3_18.

[87] Takuya Hayashi et al. "Solving a 676-Bit Discrete Logarithm Problem in GF(3^{6n})". In: *PKC 2010*. Ed. by Phong Q. Nguyen and David Pointcheval. Vol. 6056. LNCS. Springer, Heidelberg, May 2010, pp. 351–367. DOI: 10.1007/978-3-642-13013-7_21.

[88] Martin E. Hellman and Justin M. Reyneri. "Fast Computation of Discrete Logarithms in GF(q)". In: *CRYPTO'82*. Ed. by David Chaum, Ronald L. Rivest, and Alan T. Sherman. Plenum Press, New York, USA, 1982, pp. 3–13.

[89] Michael J Jacobson. *Subexponential class group computation in quadratic orders*. Shaker, 1999.

[90] Michael J Jacobson and Hugh C Williams. *Solving the Pell equation*. Springer, 2009.

[91] David Jao and Vladimir Soukharev. "A subexponential algorithm for evaluating large degree isogenies". In: *International Algorithmic Number Theory Symposium*. Springer. 2010, pp. 219–233.

[92] John W Jones and David P Roberts. "A database of number fields". In: *LMS Journal of Computation and Mathematics* 17.1 (2014), pp. 595–618.

[93] A Joux. *Discrete logarithms in GF(2^{6168})*. 2013.

[94] Antoine Joux. "A New Index Calculus Algorithm with Complexity $L(1/4+o(1))$ in Small Characteristic". In: *SAC 2013*. Ed. by Tanja Lange, Kristin Lauter, and Petr Lisonek. Vol. 8282. LNCS. Springer, Heidelberg, Aug. 2014, pp. 355–379. DOI: 10.1007/978-3-662-43414-7_18.

[95] Antoine Joux. "A One Round Protocol for Tripartite Diffie-Hellman". In: *Journal of Cryptology* 17.4 (Sept. 2004), pp. 263–276. DOI: 10.1007/s00145-004-0312-y.

[96] Antoine Joux. *Algorithmic cryptanalysis*. CRC press, 2009.

[97] Antoine Joux. "Discrete Logarithms in GF (2^ 4080)." In: *NMBRTHRY list* (2013). https://listserv.nodak.edu/cgi-bin/wa.exe?A2=NMBRTHRY;e7a0dba1.1303&S=.

[98] Antoine Joux. "Discrete logarithms in $GF(2^{607})$ and $GF(2^{613})$". In: *http://listserv. nodak. edu/cgi-bin/wa. exe? A2= ind0509&L= nmbrthry&T= 0&P= 3690* (2005).

[99] Antoine Joux. "Faster Index Calculus for the Medium Prime Case Application to 1175-bit and 1425-bit Finite Fields". In: *EUROCRYPT 2013*. Ed. by Thomas Johansson and Phong Q. Nguyen. Vol. 7881. LNCS. Springer, Heidelberg, May 2013, pp. 177–193. DOI: 10.1007/978-3-642-38348-9_11.

[100] Antoine Joux and Reynald Lercier. "The Function Field Sieve in the Medium Prime Case". In: *EUROCRYPT 2006*. Ed. by Serge Vaudenay. Vol. 4004. LNCS. Springer, Heidelberg, May 2006, pp. 254–270. DOI: 10.1007/11761679_16.

[101] Antoine Joux and Reynald Lercier. "The function field sieve is quite special". In: *International Algorithmic Number Theory Symposium*. Springer. 2002, pp. 431–445.

[102] Antoine Joux and Cécile Pierrot. *Discrete logarithm record in characteristic 3, $GF(3^{5\cdot 479})$ a 3796-bit field, Number Theory list, item 004745, Sep 2014*.

[103] Antoine Joux and Cécile Pierrot. "Improving the Polynomial time Precomputation of Frobenius Representation Discrete Logarithm Algorithms - Simplified Setting for Small Characteristic Finite Fields". In: *ASIACRYPT 2014, Part I*. Ed. by Palash Sarkar and Tetsu Iwata. Vol. 8873. LNCS. Springer, Heidelberg, Dec. 2014, pp. 378–397. DOI: 10.1007/978-3-662-45611-8_20.

[104] Antoine Joux and Cécile Pierrot. "The Special Number Field Sieve in $\mathbb{F}_{p^n}$ - Application to Pairing-Friendly Constructions". In: *PAIRING 2013*. Ed. by Zhenfu Cao and Fangguo Zhang. Vol. 8365. LNCS. Springer, Heidelberg, Nov. 2014, pp. 45–61. DOI: 10.1007/978-3-319-04873-4_3.

[105] Antoine Joux et al. "The Number Field Sieve in the Medium Prime Case". In: *CRYPTO 2006*. Ed. by Cynthia Dwork. Vol. 4117. LNCS. Springer, Heidelberg, Aug. 2006, pp. 326–344. DOI: 10.1007/11818175_19.

[106] Cameron F Kerry and Patrick D Gallagher. "Digital signature standard (DSS)". In: *FIPS PUB* (2013), pp. 186–4.

[107] Taechan Kim and Razvan Barbulescu. "Extended Tower Number Field Sieve: A New Complexity for the Medium Prime Case". In: *CRYPTO 2016, Part I*. Ed. by Matthew Robshaw and Jonathan Katz. Vol. 9814. LNCS. Springer, Heidelberg, Aug. 2016, pp. 543–571. DOI: 10.1007/978-3-662-53018-4_20.

[108] Taechan Kim and Jinhyuck Jeong. "Extended Tower Number Field Sieve with Application to Finite Fields of Arbitrary Composite Extension Degree". In: *PKC 2017, Part I*. Ed. by Serge Fehr. Vol. 10174. LNCS. Springer, Heidelberg, Mar. 2017, pp. 388–408. DOI: 10.1007/978-3-662-54365-8_16.

[109] Thorsten Kleinjung. "Discrete logarithms in $GF(2^{1279})$". In: *NMBRTHRY list, 17/10/2014* (2014).

[110] Donald E Knuth. *Art of computer programming, volume 2: Seminumerical algorithms*. Addison-Wesley Professional, 2014.

[111] Neal Koblitz. "Elliptic curve cryptosystems". In: *Mathematics of computation* 48.177 (1987), pp. 203–209.

[112] Neal Koblitz. "Hyperelliptic Cryptosystems". In: *Journal of Cryptology* 1.3 (Oct. 1989), pp. 139–150. DOI: 10.1007/BF02252872.

[113] James Kraft and Lawrence Washington. *An introduction to number theory with cryptography*. CRC press, 2018.

[114] M Kraitchik. "Théorie des nombres, vol. 1". In: *Gauthier-Villars, Paris* (1922).

[115] Brian A. LaMacchia and Andrew M. Odlyzko. "Solving Large Sparse Linear Systems over Finite Fields". In: *CRYPTO'90*. Ed. by Alfred J. Menezes and Scott A. Vanstone. Vol. 537. LNCS. Springer, Heidelberg, Aug. 1991, pp. 109–133. DOI: 10.1007/3-540-38424-3_8.

[116] Edmund Landau. "Neuer beweis des primzahlsatzes und beweis des primidealsatzes". In: *Mathematische Annalen* 56.4 (1903), pp. 645–670.

[117] T. Lange. "Digital signature: DSA with medium fields." In: (2011.). URL: https://doi.org/10.1137/0406010.

[118] Emma Lehmer. "Connection between Gaussian periods and cyclic units". In: *Mathematics of Computation* 50.182 (1988), pp. 535–541.

[119] Hendrik W Lenstra Jr. "Solving the Pell equation". In: *Notices of the AMS* 49.2 (2002), pp. 182–192.

[120] Arjen K. Lenstra et al. "The Number Field Sieve". In: *22nd ACM STOC*. ACM Press, May 1990, pp. 564–572. DOI: 10.1145/100216.100295.

[121] Hendrik W Lenstra. "Algorithms in algebraic number theory". In: *Bulletin of the American Mathematical Society* 26.2 (1992), pp. 211–244.

[122] Hendrik W Lenstra. "On the calculation of regulators and class numbers of quadratic fields". In: *Journées Arithmétiques 1980, London Math. Soc. Lecture Note Ser. 56* (1982), pp. 123–150.

[123] John E Littlewood. "On the class-number of the corpus $P(\sqrt{-k})$". In: *Proceedings of the London Mathematical Society* 2.1 (1928), pp. 358–372.

[124] Renet Lovorn. "Rigorous, subexponential algorithms for discrete logarithms over finite fields." In: (1993).

[125] Dmitrii Viktorovich Matyukhin. "Effective version of the number field sieve for discrete logarithm in a field $GF(2^{1279})$". In: *Trudy po Diskretnoi Matematike* 9 (2006), pp. 121–151.

[126] K McCurley. "Cryptographic key distribution and computation in class groups". In: *Proceedings of NATO ASI Number Theory and applications* (1989), pp. 459–479.

[127] Andreas Meyer, Stefan Neis, and Thomas Pfahler. "First Implementation of Cryptographic Protocols Based on Algebraic Number Fields". In: *ACISP 01*. Ed. by Vijay Varadharajan and Yi Mu. Vol. 2119. LNCS. Springer, Heidelberg, July 2001, pp. 84–103. DOI: 10.1007/3-540-47719-5_9.

[128] Victor S Miller. "Use of elliptic curves in cryptography". In: *Conference on the theory and application of cryptographic techniques*. Springer. 1985, pp. 417–426.

[129] Shigeo Mitsunari, Ryuichi Sakai, and Masao Kasahara. "A New Traitor Tracing". In: *IEICE Transactions* E85-A.2 (Feb. 2002), pp. 481–484.

[130] Ravi Montenegro and Prasad Tetali. "How long does it take to catch a wild kangaroo?" In: *41st ACM STOC*. Ed. by Michael Mitzenmacher. ACM Press, May 2009, pp. 553–560. DOI: 10.1145/1536414.1536490.

[131] Peter L Montgomery. "Modular multiplication without trial division". In: *Mathematics of computation* 44.170 (1985), pp. 519–521.

[132] Madhurima Mukhopadhyay and Palash Sarkar. "Combining Montgomery Multiplication with Tag Tracing for the Pollard's Rho Algorithm in Prime Order Fields". In: (2021).

[133] Madhurima Mukhopadhyay and Palash Sarkar. "Faster initial splitting for small characteristic composite extension degree fields". In: *Finite Fields and Their Applications* 62 (2020), p. 101629.

[134] Madhurima Mukhopadhyay and Palash Sarkar. *Pseudo-Random Walk on Ideals: Practical Speed-Up in Relation Collection for Class Group Computation*. Cryptology ePrint Archive, Report 2021/792. https://eprint.iacr.org/2021/792. 2021.

[135] Madhurima Mukhopadhyay et al. "New discrete logarithm computation for the medium prime case using the function field sieve". In: *Advances in Mathematics of Communications* (2020).

[136] Vassiliy Ilyich Nechaev. "Complexity of a determinate algorithm for the discrete logarithm". In: *Mathematical Notes* 55.2 (1994), pp. 165–172.

[137] Gabriel Nivasch. "Cycle detection using a stack". In: *Inf. Process. Lett.* 90.3 (2004), pp. 135–140. DOI: 10.1016/j.ipl.2004.01.016. URL: https://doi.org/10.1016/j.ipl.2004.01.016.

[138] Andrew M. Odlyzko. "Discrete Logarithms in Finite Fields and Their Cryptographic Significance". In: *EUROCRYPT'84*. Ed. by Thomas Beth, Norbert Cot, and Ingemar Ingemarsson. Vol. 209. LNCS. Springer, Heidelberg, Apr. 1985, pp. 224–314. DOI: 10.1007/3-540-39757-4_20.

[139] Pascal Paillier. "Public-Key Cryptosystems Based on Composite Degree Residuosity Classes". In: *EUROCRYPT'99*. Ed. by Jacques Stern. Vol. 1592. LNCS. Springer, Heidelberg, May 1999, pp. 223–238. DOI: 10.1007/3-540-48910-X_16.

[140] Daniel Panario, Xavier Gourdon, and Philippe Flajolet. "An Analytic Approach to Smooth Polynominals over Finite Fields". In: *Algorithmic Number Theory, Third International Symposium, ANTS-III, Portland, Oregon, USA, June 21-25, 1998, Proceedings*. Ed. by Joe Buhler. Vol. 1423. Lecture Notes in Computer Science. Springer, 1998, pp. 226–236. DOI: 10.1007/BFb0054865. URL: https://doi.org/10.1007/BFb0054865.

[141] Sachar Paulus and Tsuyoshi Takagi. "A New Public-Key Cryptosystem over a Quadratic Order with Quadratic Decryption Time". In: *Journal of Cryptology* 13.2 (Mar. 2000), pp. 263–272. DOI: 10.1007/s001459910010.

[142] Michael Pohst. *Algorithmic Methods in Algebra and Number Theory*. Academic Press, 1987.

[143] Michael E Pohst and Hans Zassenhaus. *Algorithmic algebraic number theory, Cam*. 1989.

[144] Michael Pohst and Hans Zassenhaus. *Algorithmic algebraic number theory*. Vol. 30. Cambridge University Press, 1997.

[145] Michael Pohst and Hans Zassenhaus. "Über die Berechnung von Klassenzahlen und Klassengruppen algebraischer Zahlkörper." In: (1985).

[146] John M. Pollard. "Kangaroos, Monopoly and Discrete Logarithms". In: *Journal of Cryptology* 13.4 (Sept. 2000), pp. 437–447. DOI: 10.1007/s001450010010.

[147] John M Pollard. "Monte Carlo methods for index computation (mod p)". In: *Mathematics of computation* 32.143 (1978), pp. 918–924.

[148] Hans-Georg Rück. "On the discrete logarithm in the divisor class group of curves". In: *Math. Comput.* 68.226 (1999), pp. 805–806. DOI: 10.1090/S0025-5718-99-01043-1. URL: https://doi.org/10.1090/S0025-5718-99-01043-1.

[149] W. A. Stein et al. *Sage Mathematics Software*. http://www.sagemath.org. The Sage Development Team. 2013.

[150] Ryuichi Sakai, Kiyoshi Ohgishi, and Masao Kasahara. "Cryptosystems based on Pairing". In: *SCIS 2000*. Okinawa, Japan, Jan. 2000.

[151] Palash Sarkar and Shashank Singh. "Fine Tuning the Function Field Sieve Algorithm for the Medium Prime Case". In: *IEEE Trans. Inf. Theory* 62.4 (2016), pp. 2233–2253. DOI: 10.1109/TIT.2016.2528996. URL: https://doi.org/10.1109/TIT.2016.2528996.

[152] Takakazu Satoh, Kiyomichi Araki, et al. "Fermat quotients and the polynomial time discrete log algorithm for anomalous elliptic curves". In: *Rikkyo Daigaku sugaku zasshi* 47.1 (1998), pp. 81–92.

[153] Renate Scheidler, Johannes Buchmann, and Hugh C. Williams. "A Key-Exchange Protocol Using Real Quadratic Fields". In: *Journal of Cryptology* 7.3 (Sept. 1994), pp. 171–199. DOI: 10.1007/BF02318548.

[154] Oliver Schirokauer. "Discrete logarithms and local units". In: *Philosophical Transactions of the Royal Society of London. Series A: Physical and Engineering Sciences* 345.1676 (1993), pp. 409–423.

[155] Claus Peter Schnorr. *Block Korkin-Zolotarev bases and successive minima*. Citeseer, 1992.

[156] Claus-Peter Schnorr. "Efficient Signature Generation by Smart Cards". In: *Journal of Cryptology* 4.3 (Jan. 1991), pp. 161–174. DOI: 10.1007/BF00196725.

[157] Claus-Peter Schnorr and Martin Euchner. "Lattice basis reduction: Improved practical algorithms and solving subset sum problems". In: *Mathematical programming* 66.1 (1994), pp. 181–199.

[158] C-P Schnorr and Hendrik W Lenstra. "A Monte Carlo factoring algorithm with linear storage". In: *Mathematics of Computation* 43.167 (1984), pp. 289–311.

[159] Daniel Shanks. "Class number, a theory of factorization, and genera". In: *Proc. of Symp. Math. Soc., 1971*. Vol. 20. 1971, pp. 41–440.

[160] Daniel Shanks. "The infrastructure of a real quadratic field and its applications". In: *Proceedings of the Number Theory Conference*. 1972, pp. 217–224.

[161] Naoyuki Shinohara et al. "Key Length Estimation of Pairing-Based Cryptosystems Using η_T Pairing over $GF(3^n)$". In: *IEICE Trans. Fundam. Electron. Commun. Comput. Sci.* 97-A.1 (2014), pp. 236–244. DOI: 10.1587/transfun.E97.A.236. URL: https://doi.org/10.1587/transfun.E97.A.236.

[162] Victor Shoup. "Lower Bounds for Discrete Logarithms and Related Problems". In: *EUROCRYPT'97*. Ed. by Walter Fumy. Vol. 1233. LNCS. Springer, Heidelberg, May 1997, pp. 256–266. DOI: 10.1007/3-540-69053-0_18.

[163] Shashank Singh. "Studies on Index Calculus Techniques for the Discrete Log Problem". PhD thesis. Indian Statistical Institute, Kolkata, June 2016.

[164] Nigel P. Smart. "The Discrete Logarithm Problem on Elliptic Curves of Trace One". In: *Journal of Cryptology* 12.3 (June 1999), pp. 193–196. DOI: 10.1007/s001459900052.

[165] Nigel P. Smart and Frederik Vercauteren. "Fully Homomorphic Encryption with Relatively Small Key and Ciphertext Sizes". In: *PKC 2010*. Ed. by Phong Q. Nguyen and David Pointcheval. Vol. 6056. LNCS. Springer, Heidelberg, May 2010, pp. 420–443. DOI: 10.1007/978-3-642-13013-7_25.

[166] William Stein. "Algebraic number theory, a computational approach". In: *Harvard, Massachusetts* (2012).

[167] Hans-Joachim Stender. "Eine Formel für Grundeinheiten in reinen algebraischen Zahlkörpern dritten, vierten und sechsten Grades". In: *Journal of Number Theory* 7.2 (1975), pp. 235–250.

[168] Andrew V Sutherland. "Order computations in generic groups". PhD thesis. Massachusetts Institute of Technology, 2007.

[169] The CADO-NFS Development Team. *CADO-NFS, An Implementation of the Number Field Sieve Algorithm*. Release 2.3.0. 2017. URL: http://cado-nfs.gforge.inria.fr/.

[170] Edlyn Teske. "Computing discrete logarithms with the parallelized kangaroo method," in: *Discret. Appl. Math.* 130.1 (2003), pp. 61–82. DOI: 10.1016/S0166-218X(02)00590-5. URL: https://doi.org/10.1016/S0166-218X(02)00590-5.

[171] Edlyn Teske. "Speeding Up Pollard's Rho Method for Computing Discrete Logarithms". In: *Algorithmic Number Theory, Third International Symposium, ANTS-III, Portland, Oregon, USA, June 21-25, 1998, Proceedings*. Ed. by Joe Buhler. Vol. 1423. Lecture Notes in Computer Science. Springer, 1998, pp. 541–554. DOI: 10.1007/BFb0054891. URL: https://doi.org/10.1007/BFb0054891.

[172] Nicolas Thériault. "Index Calculus Attack for Hyperelliptic Curves of Small Genus". In: *ASIACRYPT 2003*. Ed. by Chi-Sung Laih. Vol. 2894. LNCS. Springer, Heidelberg, Nov. 2003, pp. 75–92. DOI: 10.1007/978-3-540-40061-5_5.

[173] Emmanuel Thomé. "Algorithmes de calcul de logarithmes discrets dans les corps finis". PhD thesis. 2003.

[174] Emmanuel Thomé. "Computation of Discrete Logarithms in $\mathbb{F}_{2^{607}}$". In: *ASIACRYPT 2001*. Ed. by Colin Boyd. Vol. 2248. LNCS. Springer, Heidelberg, Dec. 2001, pp. 107–124. DOI: 10.1007/3-540-45682-1_7.

[175] Paul C. van Oorschot and Michael J. Wiener. "Parallel Collision Search with Cryptanalytic Applications". In: *Journal of Cryptology* 12.1 (Jan. 1999), pp. 1–28. DOI: 10.1007/PL00003816.

[176] Alfred Edward Western et al. *Tables of indices and primitive roots*. University Press Cambridge, 1968.

[177] Douglas Wiedemann. "Solving sparse linear equations over finite fields". In: *IEEE transactions on information theory* 32.1 (1986), pp. 54–62.