STUDIES IN COMPUTATIONAL NUMBER THEORY WITH APPLICATIONS TO CRYPTOGRAPHY

Dan Boneh

A DISSERTATION
PRESENTED TO THE FACULTY
OF PRINCETON UNIVERSITY
IN CANDIDACY FOR THE DEGREE
OF DOCTOR OF PHILOSOPHY

RECOMMENDED FOR ACCEPTANCE
BY THE DEPARTMENT OF
COMPUTER SCIENCE

November 1996

© Copyright by Dan Boneh 1996
All Rights Reserved

Abstract

We study various number theoretic problems that arise from cryptographic protocols. Our results are used to prove the security of new and existing protocols. The thesis is divided into five parts. We begin by studying algebraic algorithms which are a weak model of computation suitable for modeling number theoretic algorithms. We prove a hardness criteria for algebraic algorithms and present several number theoretic consequences of this result. Next we study black box fields and show how they can be used to break algebraically homomorphic encryption schemes. These algorithms show that over the group of points of an elliptic curve the hardness of computing discrete-log implies the security of the Diffie-Hellman protocol. We go on to study the security of portions of the Diffie-Hellman secret. We show that $\sqrt{\log p}$ bits of the Diffie-Hellman secret are as hard to compute as the whole secret. Furthermore, in some cases the individual most significant bit is sufficient. These results lead us to suggest a new variant of the Diffie-Hellman protocol for which the most significant bit of the secret is hard to compute. The fourth part generalizes the Goldreich-Levin theorem to arbitrary finite groups and describes some applications to learning. Finally in the last chapter we study a coding theory problem which arises from the problem of fingerprinting digital data.

Acknowledgements

It is said that this is the hardest and most important section of the thesis. I can only attempt to do a decent job of it and hope that no one is forgotten.

First and foremost, my most sincere thanks go to my advisor Richard Lipton. There is no doubt in my mind that I was extremely fortunate to have such an amazing advisor. I have learned a great deal from his knowledge and guidance. His research methodology will be with me forever.

During my time in Princeton I cooperated with several people in the area. I would like to thank my coauthors Eric Baum and Ramarthnam Venkatesan. Working with both Eric and Venkie was a great pleasure.

Next I would like to thank my family who in recent years has grown by some constant factors. Their support, encouragement, and frequent visits were of great value to me. Thank you mom and dad, Shahi, Ruti, Yifat, Seffi, Galya, Oren, Tamar and the 48 hours old Yuval. My special thanks go to Moni Naor for his interest in my work and for many enlightening conversations.

Being a part of the computer science department at Princeton was a wonderful experience for me. I would like to thank my cheery office mates Chuck Rose, James Shaw and Ramesh Sitaraman. I would also like to thank my friends Pei Cao, Yuqun Chen, Drew Dean, Chris Dunworth, Mariusz Jakubowski, Haim Kaplan, Rob Shillner, Kevin Skadron and Dan Wallach. They have all helped in making my time at Princeton memorable.

Finally, I acknowledge NSF grant CCR–9304718 for its financial support which enabled my studies in Princeton. I am also grateful to DIMACS for organizing many wonderful workshops in the area.

Contents

Abstract iii

Acknowledgements iv

1 Introduction 1

1.1 Thesis outline 2

1.1.1 Algebraic algorithms 3

1.1.2 Black box fields 3

1.1.3 Diffie-Hellman bit security 4

1.1.4 A group theoretic Goldreich-Levin theorem 5

1.1.5 Collusion secure fingerprinting 6

1.2 Knowledge assumed of the reader 7

1.3 Notation used throughout the thesis 7

2 Lower Bounds for Algebraic Algorithms 8

2.1 Straight line complexity 9

2.2 Algebraic algorithms 12

2.3 The Hardness of factoring 16

2.4 A hardness criteria in characteristic 0 21

2.4.1 Preliminaries 22

2.4.2 Proof of the hardness criteria 27

2.5 Applications 31

2.5.1 Recurrence relations 31

2.5.2 Chromatic Polynomials 33
2.6 History/Related work 33
2.7 Summary 35

3 Algorithms for Black-Box Fields **36**
3.1 Black-box fields 38
3.2 Algorithms for black-box fields 41
3.2.1 A sub-exponential algorithm for BBFP 41
3.2.2 A two step algorithm 44
3.3 Applications 47
3.3.1 Algebraically homomorphic encryption schemes 47
3.3.2 The Diffie-Hellman protocol 49
3.3.3 The power of algebraic algorithms 54
3.4 Black-box fields in characteristic 0 55
3.5 Conclusions and open problems 57

4 Diffie-Hellman Bit Security **59**
4.1 Discovering a number given an MSB oracle 60
4.2 Using Square Root log$p$ Bits 61
4.2.1 Proof of the square root $\log p$ bit theorem 65
4.3 The case of a small generator 69
4.3.1 A variant of Diffie-Hellman and its bit security 71
4.4 A nonuniform result 72
4.4.1 An application to the Okamoto conference scheme 78
4.4.2 An alternate proof of the small generator case 79
4.5 Conclusions 80

5 A Group Theoretic Goldreich-Levin Theorem **82**
5.1 Hard core bits 83
5.2 Learning using orthonormal bases 86
5.3 Group Representations 88

5.4 An algorithm for finding large Fourier coefficients 93
5.4.1 Finding Representations ρ with $\| \hat{f}_\rho \| > \beta$ 95
5.5 Applications 100
5.6 Generalization to arbitrary groups 102
5.7 Related work and some extensions 105
5.8 Conclusions 106

6 Collusion-Secure Fingerprinting **107**
6.1 Fingerprinting codes 109
6.2 Protection Against Naive Redistribution 110
6.2.1 Construction of c-frameproof codes 111
6.3 c-secure Codes 114
6.4 Construction of collusion secure codes 116
6.4.1 Logarithmic length c-secure codes 121
6.5 A lower bound 123
6.6 Distribution Scheme 125
6.7 Conclusions 127

Bibliography **128**

Chapter 1

Introduction

Computational number theory has fascinated mathematicians throughout history. Euclid ($\sim$ -365 – $\sim$ -300), Eratosthenes ($\sim$-276 – $\sim$-197), Fermat (1601–1655), Euler (1707–1783), Gauss (1777–1855) and Jacobi (1804–1851) have all contributed many ingenious algorithms to the theory of numbers [Wei84]. These algorithms enable one to generate prime numbers, express an integer as a product of primes and construct integer solutions to certain algebraic equations. Gauss did a great service to the field with his famous quote saying that the problems of testing primality and factoring integers into primes are "one of the most important and useful in all of arithmetic".

Cryptography, the science of information hiding, is another discipline that has been used throughout time. One of the earliest uses of cryptography occurs in the famous Greek drama the 'Iliad', in which Bellerophon is sent to the king with a secret tablet telling the king to have him put to death. The king tries to kill him by having him fight several mythical creatures, but Bellerophon wins every battle. Julius Caesar used a system of cryptography (i.e. the 'Caesar Cipher') which shifted each letter 2 places further through the alphabet (e.g. Y shifts to A, R shifts to T, etc.). Since the time of the greeks and romans cryptographic techniques have become more and more elaborate culminating in the famous cryptosystems of world war II. See [Kah67, Kah91, Gar79] for an historic account of these cryptosystems.

Up until 30 years ago no one would have dreamed that computational number theory and cryptography would become so closely intertwined. The link between the two disciplines was first observed in a seminal paper by Diffie and Hellman [DH76] published in 1976. The paper shows how problems that arise naturally in computational number theory can be used to construct cryptographic systems. These cryptosystems are far more flexible than their classical counterparts. The paper marks the birth of modern day cryptography. The results in it opened the door to a beautiful and elegant theory. The ever growing popularity of the Internet has turned the theory of modern cryptography into an extremely practical one which effects our daily lives. This link between two unrelated disciplines is a wonderful example of how a topic which has been regarded as purely theoretical for the past two thousand years can turn into big business in less than twenty years.

The past twenty year have seen a slew of applications of computational number theory to cryptography. Most notable is the famous result of Rivest, Shamir and Adleman [RSA78] showing how a public key cryptosystem can be built based on the hardness of factoring integers. Many other cryptosystems rely on the hardness of certain number theoretic problems for their security. As a result many new open problems were introduced in computational number theory. Thus, computational number theory has benefited as well from its link to cryptography.

In this thesis we study several new number theoretic problems that are motivated by cryptographic schemes. We define several abstract problems, study them and apply our results to the cryptographic schemes which motivated these problems. As is often the case, our solutions to the abstract problems turn out to have many other applications as well.

1.1 Thesis outline

We begin by motivating the results in the thesis without going into any technical details. The reader may wish to consult the survey article by Rivest [Riv90] for the appropriate background in cryptography. For a more number theoretic introduction

to cryptography see [Kob87a].

The main body of the thesis is composed of five chapters dealing with various number theoretic issues that arise in cryptography. These issues include the Diffie-Hellman protocol, pseudo random number generation and copyright protection. With the exception of Chapter 2, all chapters are the full version of results that have already been published [BL96, BV96, Bon95, BS95].

1.1.1 Algebraic algorithms

An algebraic algorithm is a weak computational model. The algorithm's ability to manipulate its input is severely limited. Informally, the algorithm may only apply elementary arithmetic operations and equality tests to its input. Many number theoretic algorithms can be modeled as algebraic algorithms.

The objectives of Chapter 2 are twofold. First, we define and study algebraic algorithms. The main result of the chapter is a hardness criteria for when a certain function can not be computed by an algebraic algorithm. The validity of this criteria relies on a standard cryptographic assumption; namely, the hardness of factoring integers. We then go on to describe several number theoretic applications of this hardness criteria.

The second objective is to show that the hardness of factoring has interesting number theoretic consequences. In general we show that several Diophantine problems can only have a finite number of solutions unless factoring integers is easy. The statement claiming that integers can not be factored in polynomial time is extremely deep. It is therefore likely to have many deep number theoretic consequences. One may generate new number theoretic conjectures or test existing conjectures by showing that they follow from the hardness of factoring. We give many such examples.

1.1.2 Black box fields

Black box fields arise in several cryptographic scenarios. A field is called *black box* if its elements are represented as arbitrary binary strings, i.e. in some non canonical

form. For example, let E be an encryption function which takes plain-texts in the field $\mathbb{F}_p$. The encryption of the field elements would be a black box representation of the field. Given two black box elements, one is assumed to have the ability to form the black box representation of their sum and product.

The main question we address is the following: consider a black box representation of the field $\mathbb{F}_p$. Given a black box element α how quickly can one find the explicit field element represented by α. In other words, how quickly can one invert the encryption function E? Algorithms for solving this problem enable one to break *algebraically homomorphic* encryption schemes. These concepts are defined in the chapter. Second, this problem turns out to be of great importance for proving the security of the Diffie-Hellman protocol.

We present several algorithms for solving the above black box field problem. These algorithms require sub-exponential time in $\log p$ and are based on a technique due to Maurer [Mau94]. Consequently, any algebraically homomorphic encryption scheme can be broken in sub-exponential time. Furthermore, we deduce that breaking the Diffie-Hellman protocol is as hard as computing discrete log when one is working over the group of points of an elliptic curve. Such a result holds for several other groups as well.

Finally, we consider a variant of the black box field problem over the field of rational numbers. The hardness criteria developed in Chapter 2 is used to show that over the rationals this problem is as hard as factoring integers. This result suggests that manipulating black box fields of characteristic 0 is harder than manipulating black box finite fields.

1.1.3 Diffie-Hellman bit security

The Diffie-Hellman protocol enables two parties to share a secret which can not be discovered by a passive eavesdropper. For security reasons the current size of the shared secret is about 1024 bits. In practice the two parties need about 128 bits of shared secret which they use as the key for a secret-key encryption scheme such as DES or IDEA. The straight forward approach for converting the 1024 bit Diffie-Hellman

secret to a 128 bit secret is to simply use the 128 most significant bits of the secret. However, an adversary who may not be able to compute the entire Diffie-Hellman secret may still be able to compute its most significant bits.

In this chapter we study the hardness of computing the most significant bits of the Diffie-Hellman secret. We show that computing $3\sqrt{\log p}$ bits is as hard as computing the entire secret. Hence, in the example above, the 128 most significant bits are secure as the entire secret. Our methods are based on lattice basis reductions and the LLL algorithm. These methods are general enough so that they apply to many other cryptographic schemes. For instance, we show that computing $3\sqrt{\log p}$ of the secret message in the ElGamal public key cryptosystem is as hard as computing the entire message. The same holds for Shamir's three pass protocol and Okamoto's conference key scheme. These cryptosystems are all described in the chapter.

To address these issues we define an abstract problem which, for lack of a better name, we call *the hidden number problem*. In some cases the hidden number problem can be solved using only the single most significant bit. This result leads us to suggest a new variant of the Diffie-Hellman protocol for which the single most significant bit is as hard to compute as the entire secret.

Finally, using techniques of rounding in lattices we show that in a non-uniform model the $3\sqrt{\log p}$ bits result can be improved to $\log \log p$ bits. Currently this result is purely of theoretical interest. However the techniques used are likely to play a role in improving the $3\sqrt{\log p}$ result.

1.1.4 A group theoretic Goldreich-Levin theorem

The results of the previous section prove the hardness of computing parts of the Diffie-Hellman secret. The ultimate goal of this line of research is to prove that even predicting the most significant bit of the Diffie-Hellman secret with non-negligible probability is as hard as computing the entire secret. In other words we wish to show that the most significant bit is a *hard core* of the Diffie-Hellman function. Such a result is known for the RSA public key system, but it still open for the Diffie-Hellman key exchange.

A very powerful theorem due to Goldreich and Levin shows that any one-way function (a function which is easy to evaluate, but hard to invert) has a hard core bit (a bit which is as hard to predict with non-negligible probability as inverting the one-way function). This theorem enables one to construct secure pseudo random number generators from one-way permutations.

In this chapter we generalize the theorem and show that in fact a one-way function has many hard-core values. The method relies on embedding different groups structures in the boolean cube $\{0,1\}^n$. We use the linear representations of these finite groups to prove the generalized Goldreich-Levin theorem.

This generalized theorem has applications to learning theory. We generalize an approach due to Kushilevitz and Mansour to show that group decision trees can be learned in polynomial time. Roughly speaking, a group decision tree is a decision tree where internal nodes test the value of an expression involving group operations. This approach also provides a general tool which is useful for learning various classes of functions.

1.1.5 Collusion secure fingerprinting

The final chapter discusses a coding theory problem that arises in the context of fingerprinting digital data. Fingerprinting is an old technique for intellectual property protection. Fingerprinting consists of uniquely marking and registering each copy of the data. This marking allows a distributor to detect any unauthorized copy and trace it back to the user.

When users collude they can compare two different fingerprinted objects. The differences between the objects will expose the location of the fingerprint. The users can then alter the fingerprint to mask their identities. We construct fingerprinting codes which are secure in the context of collusion. Our scheme is efficient and can be used to protect many types of digital data.

1.2 Knowledge assumed of the reader

This entire chapter is written for the most general computer science reader. Unfortunately, the rest of the thesis assumes substantial knowledge on the part of the reader. The following rule of thumb was used in the exposition of the material:

> Any technical term which can be adequately defined in less than five lines of text is defined in the thesis body. All other technical terms which are not crucial for understanding the main results are defined through a reference to the literature.

1.3 Notation used throughout the thesis

Unless otherwise stated $\log x$ will always stand for $\log_2 x$. As usual $\mathbb{N}, \mathbb{Z}, \mathbb{R}, \mathbb{C}$ represent the positive integers, all integers, the reals, and the complex plain respectively.

$\mathbb{F}_p$ is a Galois field of size p, i.e. a finite field with p elements. $\mathbb{Z}_n$ is the additive group of integers modulo n. We only deal with commutative rings with units. For a ring R we denote by R^* the group of invertible elements in R. We use $\mathbb{F}_p^*$ and $\mathbb{Z}_p^*$ to denote the multiplicative group of integers modulo p.

For a finite set S we let $|S|$ denote the number of elements in S.

Chapter 2

Lower Bounds for Algebraic Algorithms

In this chapter we define and study *algebraic algorithms*. An algebraic algorithm is a weak model of computation. It is a natural one to study since many number theoretic problems can be solved using algebraic algorithms. The concepts defined in this chapter are necessary for understanding the results of the next chapter.

The main result of this chapter is a criteria for when a function can not be computed by an algebraic algorithm. Roughly speaking the result shows that if a function $f : \mathbb{C} \to \mathbb{C}$ has many roots in low algebraic extensions of the rationals then it can not be computed by a polynomial time algebraic algorithm. The validity of this criteria relies on a standard cryptographic assumption, namely that the problem of factoring integers can not be solved in polynomial time.

This hardness result for algebraic algorithms has many applications. First, it can be used to show that certain natural polynomials are hard to evaluate. Interestingly enough, the criteria has some number theoretic consequences. We derive bounds on the number of periods of a recurrence relation $T_n(x) = f(T_{n-1}(x))$ for some polynomial f (a point x is called a period of f if $T_n(x) = x$ for some n). Obtaining uniform bounds on the number of periods of such recurrences is a long standing open problem. We also show that certain Diophantine problems can not have an infinite

$\tau(13) \leq 6$:	1	$\tau(78) \leq 6$:	1
	$1+1=2$		$1+1=2$
	$2+1=3$		$2+1=3$
	$2+3=5$		$3*3=9$
	$3*5=15$		$9*9=81$
	$15-2=13$		$81-3=78$

Figure 1: Straight line computations

number of solutions, unless factoring is easy. These applications are examples of how a cryptographic assumption can be used to derive purely number theoretic facts.

2.1 Straight line complexity

We begin by defining the straight line complexity of integers and polynomials. The straight line complexity of an integer measures how many arithmetic operations are necessary to build up the integer from 1. Similarly, the straight line complexity of a polynomial measures how many arithmetic operations are necessary to evaluate the polynomial at a given point.

More precisely, a *straight line computation* for an integer N is a sequence of rational numbers $a_1, a_2, \ldots, a_n$ where $a_1 = 1$, $a_n = N$ and for all $i > 1$ the a_i satisfy $a_i = a_j \circ a_k$ where $\circ$ is one of $+ - * \div$ and $j, k < i$. We say that n is the length of the computation. Examples of a few straight line computations are shown in Figure 1. The *straight line complexity* of an integer N, denoted by $\tau(N)$, is defined as the length of the shortest straight line computation for N. Table 1 lists the straight line complexity of all natural numbers less than 200. The straight line complexity of a rational number is defined analogously.

Usually division in not allowed in straight line computations. One can easily see that division can reduce the complexity of various integers. For instance 2^{2^n-1} can

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
0		1	2	3	3	4	4	5	4	4	5	5	5	6	5	5	4	5	5	6
20	5	6	6	6	5	5	6	5	6	6	6	6	5	6	6	6	5	6	6	6
40	6	7	6	7	7	6	7	7	6	6	6	7	7	7	6	7	6	7	7	7
60	6	7	6	6	5	6	6	7	6	7	7	7	6	7	7	6	7	7	6	6
80	6	5	6	6	6	7	7	7	7	7	6	7	7	7	7	7	6	7	7	6
100	6	7	7	7	7	7	7	7	6	7	7	7	7	8	7	7	7	7	7	7
120	7	6	7	7	7	6	7	7	6	7	7	8	7	8	8	7	7	8	7	8
140	7	7	7	7	6	7	7	7	7	8	7	8	7	7	7	8	7	8	7	7
160	7	7	6	7	7	7	7	8	7	7	8	7	8	8	8	7	7	8	8	8
180	7	8	7	8	8	9	8	8	7	7	7	7	6	7	7	7	6	7	7	8

Table 1: Straight line complexity of integers less than 200

be computed as $2^{2^n}/2$ in $n+3$ steps. Without division this would take $2n$ steps. All our results will allow the straight line computation to use division.

Lemma 2.1.1 *For any integer $N > 1$ we have* $\log\log N \leq \tau(N) \leq 2\log N$.

Proof The left part of the inequality follows from the observation that the largest number which can be created in n steps is 2^{2^n}. The right part of the inequality follows by considering the binary representation of N. The program first constructs all power of two up to N and then sums the appropriate ones to obtain N. □

The problem of constructing families of integers for which $\tau(N)$ is large is still open. Even proving that $\tau(N!) > (\log N)^c$ for any constant c and sufficiently large N is still open. An old folk theorem states that if integers can not be factored in polynomial time then the above statement is true. We will come back to this theorem in the next section.

One can define the straight line complexity of a polynomial in a way similar to that used for integers. Let K be a field and $f \in K[X_1, \ldots, X_n]$ be a multivariate polynomial with coefficients in K. A straight line computation for f is a sequence

of rational functions $g_1, \ldots, g_k \in K(X_1, \ldots, X_n)$ where each g_i satisfies one of the following: (1) g_i is the constant 1 or is a variable X_i. (2) $g_i = g_j \circ g_k$ where $\circ$ is one of $+ - * \div$ and $j, k < i$. We say that n is the length of the computation. As before, the *straight line complexity* of the polynomial f, denoted by $\tau(f)$ is the length of the shortest straight line computation for f. The straight line complexity of a rational function is defined analogously.

The above definition needs some clarification when dealing with extension fields. Consider the field $K = \mathbb{Q}[\sqrt{2}]$. The linear polynomial $P(X) = X - \sqrt{2} \in K[X]$ can not be constructed using a straight line computations as defined above. The irrationality of $\sqrt{2}$ implies that it can not be constructed by arithmetic operations on integers. Therefore, in this context we can only discuss the straight line complexity of polynomials over *prime fields*. A prime field is a field which has no non-trivial subfields. Any prime field is isomorphic to $\mathbb{F}_p$ for some prime p (hence the name) or to $\mathbb{Q}$. Given a field K, its unique prime field is the field generated by 1. The straight line complexity is only defined for rational functions over the prime field of K. For the applications we have in mind it is crucial that we only allow the computations to use rational functions over prime fields.

The following lemma is useful when dealing with straight line computations involving divisions.

Lemma 2.1.2 *Let K be a prime field and $P(X)/Q(X)$ be a rational function, $P, Q \in K[X]$, $\gcd(P, Q) = 1$. Suppose $\tau(P/Q) = r$. Then there exists a non-zero polynomial $T(X)$ in $K[X]$ so that $\tau(TP) \leq 4r$ and $\tau(TQ) \leq 4r$.*

Proof Let $P_1/Q_1,\ P_2/Q_2,\ \ldots, P_r/Q_r$ be the sequence of rational functions encountered during the straight line computation for P/Q. The idea is to define a new computation which looks like $T_1P_1, T_1Q_1, T_2P_2, T_2Q_2, \ldots, T_rP_r, T_rQ_r$ where the T_i are some polynomials in $K[X]$. Addition, subtraction, multiplication and division in the original computation can be carried out in at most four steps in the new computation. For instance, to add the two rational functions P_1/Q_1 and P_2/Q_2 one computes the numerator in three steps as $(P_1T_1)(Q_2T_2) + (P_2T_2)(Q_1T_1)$ and the denominator in one

step as $(T_1Q_1)(T_2Q_2)$. The T polynomial associated with the sum is simply T_1T_2. Multiplication and division can be carried out in two steps. Hence, the total length of the computation increases by at most a factor of 4 proving that $\tau(T_rP_r) \leq 4r$ and $\tau(T_rQ_r) \leq 4r$. □

Corollary 2.1.3 *Let N be an integer with $\tau(N) = r$. Then there exists a non-zero integer T such that $\tau(TN) \leq 4r$ and TN has a straight line computation of length $4r$ which requires no divisions.*

Proof Let $\frac{p_1}{q_1}, \frac{p_2}{q_2}, \ldots, \frac{p_r}{q_r}$ be the rational numbers encountered during the computation of N. We transform this computation into a division free computation just as in Lemma 2.1.2. The final two elements in the computation will be two integers TN and T whose ratio is N. The integer TN is the required integer. □

The straight line complexity of integers and polynomials is a natural quantity to study. We discuss some results regarding these measures at the end of the chapter.

2.2 Algebraic algorithms

Straight line programs are a very weak model for evaluating functions. There is no reason why an algorithm should not be allowed to change its behavior according certain intermediate values. An *algebraic computation tree* enables the algorithm to test a certain predicate at every time step and modify its behavior accordingly.

Let K be a number field and L its prime subfield. An algebraic computation tree is a finite binary tree (each vertex has zero or two children) where each vertex is labeled with some polynomial in $L[X_1, \ldots, X_n]$. The polynomial associated with vertex v is denoted by p_v. The tree defines a function $f : K^n \to K$. To evaluate the function f on inputs $x_1, \ldots, x_n \in K$ one moves along a path from the root to a leaf of the tree. At every internal node v one tests the condition $p_v(x_1, \ldots, x_n) = 0$. If the condition holds the next node on the path is the right son of the current node,

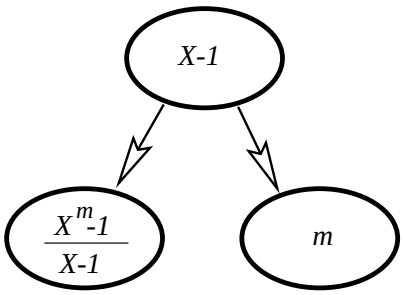

Figure 2: An algebraic computation tree

otherwise it is the left son. When a leaf l is finally reached, the value $p_l(x_1, \ldots, x_n)$ is declared to be the output of the function.

As an example consider the function $p(x) = 1 + x + x^2 + \ldots + x^{m-1}$. An algebraic computation tree which evaluates this function is a tree containing three nodes: a root and two leaves (see Figure 2). The root is labeled with $X - 1$, the right leaf with m and the left leaf with $\frac{X^m-1}{X-1}$. The right leaf is used when evaluating $p(1)$, i.e. when the condition $X = 1$ is satisfied. The left leaf is used for all other inputs.

Notice that our definition of algebraic computation trees only enables one to test equality at every internal node. For computation trees defined over the reals sometime [BO83] one allows internal nodes to test whether $p_v(x_1, \ldots, x_n)$ is bigger, equal or less than 0. Many fields, such as Galois fields (finite fields), do not have a natural order defined on their elements. Therefore these tests make no sense in our context. We only allow equality tests which make sense in any field.

Let $f : K^n \to K$ be a function computed by a computation tree $\mathcal{T}$. So far there was no mention as to how the polynomials associated with nodes of $\mathcal{T}$ are constructed. We enforce a uniformity requirement on the trees by saying that all these polynomials can be constructed by a Turing Machine. This is stated in the next definition. By a path in the tree we mean a word over the alphabet $\{left, right\}$.

Definition 2.2.1 *Let K be a field and $\mathcal{T}$ an algebraic computation tree over K. We say that a Turing Machine M* constructs *the tree $\mathcal{T}$ if given a path leading from the root to a node v the machine M outputs a straight line program for the polynomial*

p_v. *We say that the tree is* constructible *in time T if there exists a Turing Machine which constructs the tree and runs in time at most T for all inputs.*

There are two points that need to be clarified in the above definition. First, the representation used to describe the straight line program output by the machine M must be explained. Here the straight line program is output as triples of indices and operations. The structure of a triplet is (index-op1,op,index-op2). For instance, the obvious straight line program for the polynomial $(x+1)(x+2)$ is represented as

$$1,\ x,\ (1,+,2),\ (1,+,1),\ (2,+,4),\ (3,*,5)$$

Here the tuple $(1,+,2)$ means that the value at index 1 should be added to the value at index 2 resulting in the polynomial $x+1$. This representation has the property that a straight line program of length L is represented by a string of length $O(L \log L)$. Notice that all polynomials appearing in a tree which is constructible in time T have straight line complexity at most T.

The second point that needs to be clarified has to do with the problem of division by zero. Since a straight line program may use division it is possible that when a polynomial $p_v(X_1, \ldots, X_n)$ is evaluated a division by zero will be encountered. To exclude this possibility we introduce an extra requirement of the constructing machine M. We require that for all nodes v the straight line program computing p_v produced by the machine M does not encounter a division by zero for all inputs that *reach* the node v. For instance, the tree described in Figure 2 is constructible in time $O(\log m)$ since computing x^m takes $O(\log m)$ steps.

Definition 2.2.2 *Let $T > 0$ be some integer. A function $f : K^n \to K$ is computable in time T by an* algebraic algorithm *(over K) if there exists a T-time constructible algebraic computation tree computing f.*

When K is a finite field with q elements one can readily see that any function $f : K^n \to K$ which can be computed by an algebraic algorithm in time T can be computed by a Turing Machine in time at most $O(T^2 \log^2 q)$. This follows from two

simple observations. First, any tree which is constructible in time T can have depth at most T. Otherwise the machine constructing the tree will not have enough time to read its input. Second, all polynomials in the tree have straight line complexity at most T. Therefore T^2 arithmetic operations are needed to evaluate the tree and each operation takes time at most $O(\log^2 q)$. It therefore makes sense to define the following class of functions:

Definition 2.2.3 *Let $f(q, x_1, \ldots, x_n)$ be a function which for any fixed q is regarded as a function $f_q : \mathbb{F}_q^n \to \mathbb{F}_q$ (for some canonical representation of $\mathbb{F}_q$). Let $T : \mathbb{N} \to \mathbb{N}$ be a function. We say that f can be computed by an algebraic algorithm in time $T()$ if the following two conditions hold: (1) f can be computed by a Turing Machine in time $T(\log q)$ and (2) for all q the function f can be computed by an algebraic algorithm (over $\mathbb{F}_q$) in time $T(\log q)$.*

We define $\mathbb{P}_{alg}$ to be the set of functions $f(q, x_1, \ldots, x_n)$ for which there exists a constant c such that f can be computed by an algebraic algorithm in time $(\log q)^c$. For instance, as we show below, the square root function $f(q, x, y) = \sqrt{x} \pmod q$ is in[1] $\mathbb{P}_{alg}$. Condition (1) of the definition above ensures that $\mathbb{P}_{alg} \subseteq \mathbb{P}$. Here $\mathbb{P}$ is the set of functions of the form $f(q, x_1, \ldots, x_n)$ as in the definition above which are computable by a deterministic polynomial time Turing Machine. It is natural to ask whether $\mathbb{P}_{alg} = \mathbb{P}$.

Open problem: Is $\mathbb{P}_{alg} = \mathbb{P}$?

We will come back to this question in the next chapter and prove a somewhat weaker result. As we shall see, this problem is closely related to the black box field problem which we study in the next chapter.

Many number theoretic algorithms can be modeled as algebraic algorithms. Examples are Shank's algorithm for computing square roots in finite fields [Coh91, pp. 32–34] and Schoof's seminal algorithm for counting the number of points of an elliptic curve over a finite field [Sch85]. To familiarize the reader with the notion of

[1] To make the square root algorithm deterministic it must be given a quadratic non-residue modulo p. This value is supplied by the extra parameter y.

an algebraic algorithm we briefly describe Shank's square root algorithm and explain why it is an algebraic algorithm. Let $\mathbb{F}_p$ be a finite field where p is some prime. Given $x \in \mathbb{F}_p$ we wish to construct $\sqrt{x} \bmod p$. Write $p - 1 = 2^k \cdot l$ where l is odd. Let g be a quadratic non-residue modulo p (i.e. g does not have a square root modulo p). Since $g^{\frac{p-1}{2}} \equiv -1 \pmod{p}$ it follows that g^l generates the group of points in $\mathbb{F}_p^*$ whose order is a power of 2. Since x^l is such a point there exists a t such that $x^l = (g^l)^t$. Furthermore, t must be even since x is a quadratic residue. We now have

$$\sqrt{x} = \frac{x^{\frac{l+1}{2}}}{(g^l)^{t/2}} \pmod{p}$$

One can verify this equality by squaring both sides. This expression is the one used in the leaves of the tree. The internal nodes of the tree are used to compute t. To find t we determine the discrete log of x^l base g^l. Since the group in question has order 2^k this can easily be done using an algorithm due to Pohlig and Hellman [PH78]. The algorithm recovers the bits of t one by one starting from the least significant. Suppose it is already known that $t \equiv t_r \pmod{2^r}$ where $r < k$. Then

$$\left(\frac{x}{g^{t_r}}\right)^{\frac{p-1}{2^{r+1}}} \equiv 1 \pmod{p} \implies t \equiv t_r \pmod{2^{r+1}}$$

$$\left(\frac{x}{g^{t_r}}\right)^{\frac{p-1}{2^{r+1}}} \not\equiv 1 \pmod{p} \implies t \equiv t_r + 2^r \pmod{2^{r+1}}$$

Initially $r = 0$ and $t_r = 0$. The tests above can be carried out in the internal nodes of the tree. To find a quadratic non-residue g one picks a random element of $\mathbb{F}_p$. A random element will be a non-residue with probability one half. Since the computation tree corresponding to Shank's algorithm is clearly constructible in polynomial time we see that the square root function can be computed by a probabilistic algebraic algorithm (we leave it to the imagination of the reader to give the formal definition of probabilistic algebraic algorithms).

2.3 The Hardness of factoring

One of the main results of this chapter is a criteria for when functions can not be computed efficiently by algebraic algorithms. The validity of our criteria will depend

on the assumption that there is no polynomial time factoring algorithm. In this section we formally state this assumption.

We denote by $\mathbb{Z}_{(2)}(n)$ the set of n-bit integers which are a product of two large primes. More precisely,

$$\mathbb{Z}_{(2)}(n) = \{N \mid N < 2^n \ , \ N = pq \ , \ p > q > 2^{n/4} \ , \ p, q \, \text{prime}\}$$

For completeness we recall that a circuit is a collection of logical gates such as disjunction, conjunction and negation which form the nodes of a directed acyclic graph. A circuit with n inputs and m outputs computes some function $f : \{0,1\}^n \to \{0,1\}^m$. The size of a circuit is the number of logical gates in it. A family of circuits $\{\mathcal{C}_n\}_{n=1}^{\infty}$ is said to be of polynomial size if there exists a constant c such that the size of the circuit $\mathcal{C}_n$ is at most n^c for all n.

Hardness of factoring assumption For any constant c and any family of polynomial size circuits $\{\mathcal{C}_n\}_{n=1}^{\infty}$ we have that for large enough n

$$\Pr\left[1 < \gcd(\mathcal{C}_n(x), x) < N\right] < \frac{1}{n^c}$$

where x is chosen uniformly at random from $\mathbb{Z}_{(2)}(n)$. Here $\mathcal{C}_n(x)$ is the output of the circuit $\mathcal{C}_n$ on input x. The output of the circuit is regarded as the binary representation of an integer.

In words, the hardness of factoring means that there is no polynomial time algorithm which can factor a polynomial fraction of the integers in $\mathbb{Z}_{(2)}(n)$. Notice that this is a non-uniform assumption meaning that a sizeable fraction of the integers can not be factored even when the algorithm is given a polynomial number of advice bits which depend only on n. The non-uniformity is crucial for the applications we have in mind.

The hardness of factoring is an assumption which is the basis of most of modern cryptography. It has been used numerous times and sadly enough it appears to be true. Factoring algorithms have been getting better over the years [LL90], however they are still very far from being polynomial time. The fastest known factoring

algorithm is the number field sieve [LL94] whose conjectured expected running time is $e^{(1+o(1))\sqrt[3]{n \log^2 n}}$ for an n bit integer. The prospects of giving a complete proof of the hardness of factoring seem pretty dim at the present. Proving any kind of non-trivial lower bound on computations is one of the hardest and deepest problem in mathematics today.

The hardness of factoring can be immediately applied to obtain a criteria for when an integer must have high straight line complexity. The proof of the next theorem relies on the prime number theorem which says that if $\pi(x)$ is the number of primes less than x then $\lim_{x\to\infty} \frac{\pi(x)}{x \ln x} = 1$. For an integer N we denote by $d(N,k)$ the fraction of distinct prime divisors of N smaller than 2^k:

$$d(N,k) \stackrel{\text{def}}{=} \frac{\left|\{p < 2^k \mid p \text{ prime divisor of } N\}\right|}{\pi(2^k)}$$

For example, $d(12,3) = d(15,3) = 2/4$.

Theorem 2.3.1 *Let $c, d > 0$ be arbitrary integers. Let $N_1, N_2, \ldots$ be a sequence of non-zero integers such that $d(N_k, k) > 1/k^c$ for all k. Then $\tau(N_k) \geq k^d$ for sufficiently large k, unless factoring is easy.*

Proof Assume towards a contradiction that $\tau(N_k) < k^d$ infinitely often. We show how the straight line program for N_k can be used to factor integers in $\mathbb{Z}_{(2)}(m)$ for some $m \geq k$. Define n to be the largest integer satisfying $d(N_k, n) \geq \frac{1}{n^c}$. Then

$$n \geq k \quad \text{and} \quad d(N_k, n+1) < \frac{1}{(n+1)^c}$$

Set $m = 2n + 1$. We propose the following simple algorithm for factoring integers $M = pq$ in $\mathbb{Z}_{(2)}(m)$:

1. Compute $T = N_k \bmod M$ using the straight line program for N_k.
2. Compute $\gcd(M, T)$.

If p divides N_k, but q doesn't then $\gcd(T, M)$ will give a nontrivial divisor of M. Notice that $\gcd(T, M)$ can be computed in polynomial time in $\log M$. To compute

$N_k \bmod M$ we use the straight line program for N_k. Recall that by assumption $\tau(N_k) < k^d \leq n^d$. Hence, by evaluating all the arithmetic operations in the straight line program modulo M one can construct $N_k \bmod M$ in time $O(n^{d+2})$.

The algorithm above runs in polynomial time. We show that it outputs a factor of $M \in \mathbb{Z}_{(2)}(m)$ for $1/n^{c+2}$ of the elements in $\mathbb{Z}_{(2)}(m)$. Two events may cause the algorithm to fail to factor $M = pq$. First, the straight line computation for $N_k \bmod M$ may encounter a division by an element which is zero modulo M. This will prevent the algorithm from evaluating $N_k \bmod M$. Our solution relies on Corollary 2.1.3. The corollary implies that there is an integer T_k such that $\tau(T_k N_k) < 4n^d$ and the straight line computation for $T_k N_k$ involves no divisions. Clearly $d(T_k N_k, k) \geq d(N_k, k)$. Therefore we may use $T_k N_k$ instead of N_k. As a result division by a non-invertible element modulo M is not a problem.

More seriously, the algorithm will fail to factor M if either none or both of p and q divide M. We show that this doesn't happen too often. By definition of n at most $\frac{1}{(n+1)^c}$ of the $(n+1)$-bit primes divide N_k. Hence, by the prime number theorem, the number of $n+1$-bit primes *not* dividing N_k is at least $\frac{1}{2}2^n/n$. Furthermore, at least $\frac{1}{n^c}$ of the primes less than 2^n divide N_k. Again, the prime number theorem shows that for large enough n the number of primes p in the range $2^{n/2} < p < 2^n$ dividing N_k is at least $\frac{1}{2}2^n/n^{c+1}$. Hence, there are at least

$$\frac{1}{2}\frac{2^n}{n} \cdot \frac{1}{2}\frac{2^n}{n^{c+1}} = \frac{1}{4}\frac{2^{2n}}{n^{c+2}} = \frac{1}{2}\frac{2^m}{n^{c+2}}$$

numbers in $\mathbb{Z}_{(2)}(m)$ for which the above algorithm will output a nontrivial factor. This contradicts the hardness of factoring. □

Notice that the proof of Theorem 2.3.1 makes use of the non-uniformity in the hardness of factoring assumption: nowhere in the proof did we insist that there is a polynomial time algorithm which given k outputs the straight line program for N_k. Consequently, the resulting factoring algorithm is non-uniform: for every k the algorithm must be given the straight line program for N_k as advice bits.

Theorem 2.3.1 has several interesting applications. For instance, it can be used

to show that for large enough k we must have $\tau(k!) > k^c$ for all $c > 0$. Similarly, the theorem can be used to generate very general number theoretic statements. For instance, the following corollary is immediate.

Corollary 2.3.2 *For any fixed integer a there are at most a finite number of integers $m, n > 0$ satisfying*

$$|n! - a^m| < n^{\log n}$$

unless factoring is easy.

Proof Suppose the corollary is false. Then infinitely often there exists an integer B with $|B| < n^{\log n}$ such that

$$\tau(n!) \leq \tau(a^m) + \tau(B) \leq O(\log^2 n)$$

contradicting the hardness of factoring. □

Corollary 2.3.2 seems like a non trivial number theoretic fact. The interested reader may try to prove the corollary without relying on the hardness of factoring. The corollary can be strengthened in many ways. For instance, to show that one can even allow polynomial relations we state the following corollary:

Corollary 2.3.3 *Let $P(X, Y) \in \mathbb{Q}[X, Y]$ be some polynomial which is non-trivial in X. Then there is at most a finite number of pairs $n, m \in \mathbb{Z}$ satisfying:*

$$|P(n!, 2^m)| < n^{10}$$

Proof Suppose $P(X, Y)$ is a polynomial for which there is an infinite number of pairs n, m as above. For large enough n we must have $m < n^2$. Let B be the integer with $|B| < n^{10}$ satisfying $P(n!, 2^m) = B$. Such an integer is assumed to exist for an infinite number of pairs n, m. Define the polynomial $P'(X, Y) = P(X, Y) - B$ and write P' as

$$P'(X, Y) = g_d(Y)X^d + g_{d-1}(Y)X^{d-1} + \ldots + g_0(Y)$$

Without loss of generality we may assume that $g_0(Y)$ is a non-zero polynomial. Otherwise one may divide through by an appropriate power of X. Notice that whenever $P'(X,Y) = 0$ it must be the case the X divides $g_0(Y)$. It follows that $n!$ must divide $g_0(2^m)$ whenever n, m is a pair satisfying the relation above. Furthermore, $\tau(g_0(2^m)) < O(\log n)$ since $B < n^{10}$, $m < n^2$ and $P(X,Y)$ is a fixed polynomial. The infinite sequence of integers $g_0(2^m)$ contradicts the hardness of factoring. □

2.4 A hardness criteria in characteristic 0

We use the hardness of factoring to derive a general criteria for when a family of functions can not be computed by polynomial time algebraic algorithms. This result generalizes a result which appeared in [Lip94]. Our results apply to any number field, i.e. any algebraic extension of the rationals [Mar77].

Definition 2.4.1 *Let $\{f_n\}_{n=1}^{\infty}$ be a family of functions where $f_n : \mathbb{C} \to \mathbb{C}$. We say that the family $\{f_n\}$ can be computed by a polynomial time algebraic algorithm if there exists a constant c such that for every n the function f_n can be computed by an algebraic algorithm in time n^c.*

We define the *height* of a rational number $\frac{p}{q}$ $(\gcd(p,q) = 1)$ to be $|p| + |q|$. Given an algebraic number $x \in \bar{\mathbb{Q}}$ (i.e. a root of some polynomial in $\mathbb{Q}[X]$) the *norm* of x, denoted by $\mathcal{N}(x)$ is the product of all the conjugates of x over $\mathbb{Q}$. The norm of x is always a rational number. Given an integer $B > 0$ and a function $f : K \to K$ where K is a number field, define the bounded set of zeros of f to be

$$Z(f, K, B) = \{x \in K \mid f(x) = 0 \text{ and } \text{height}(\mathcal{N}(x)) < B\}$$

Definition 2.4.2 *Let $\{f_n\}_{n=1}^{\infty}$ be a family of functions $f_n : \mathbb{C} \to \mathbb{C}$. We say that $\{f_n\}_{n=1}^{\infty}$ is a c-sequence if there exists a number field K such that*

1. *$|Z(f_n, K, 2^{n^c})| > 2^n/n^c$ infinitely often.*

2. *For all n, the function f_n has infinite support in K (i.e. the set $\{x \in K \mid f_n(x) \neq 0\}$ is infinite).*

The last condition is there to prevent the functions from being almost everywhere zero in K. We can now state the main theorem of this section. The theorem says that if a functions has many zeroes in low degree number fields then it can not be computed by an algebraic algorithm in polynomial time.

Theorem 2.4.1 (Hardness criteria) *Let $\{f_n\}_{n=1}^{\infty}$ be a c-sequence for some constant $c > 0$. Then $\{f_n\}_{n=1}^{\infty}$ can not be computed by a polynomial time algebraic algorithm unless factoring is easy.*

The rest of this section is devoted to proving Theorem 2.4.1. We discuss applications of this theorem in Section 2.5.

2.4.1 Preliminaries

Before giving the proof of Theorem 2.4.1 we need to state some useful facts. The first fact is known as Tchebotarev's density theorem. Given a polynomial $f \in \mathbb{Q}[X]$ and a prime p we say that f has a root modulo p if there exists a $y \in \mathbb{Z}$ such that $f(y) \equiv 0 \pmod{p}$. Define $\pi_f(x)$ to be the number of primes $p < x$ such that f has a root modulo p.

Theorem 2.4.2 (Tchebotarev's density) *Let $f \in \mathbb{Q}[X]$ be an irreducible polynomial of degree n. Then*

$$\lim_{x \to \infty} \frac{\pi_f(x)}{\pi(x)} \geq \frac{1}{n}$$

Equality holds when the number field $K = Q[X]/(f)$ is Galois.

The theorem can be viewed as a generalization of Dirichlet's theorem regarding the density of primes in arithmetic progressions [Dav80, Ch. 1]. Consider the polynomial $x^q - 1/x - 1$ for some prime q. The polynomial has a root modulo p if and only if $p \equiv 1 \pmod{q}$. Tchebotarev's density theorem shows that the density of such primes is $\frac{1}{q-1}$

which is the same result obtained from Dirichlet's theorem. For a proof of the theorem and some more details see [Neu86, pp. 132–134]. Lagarias and Odlyzko [LO77] prove an effective version of the density theorem. Our results can be slightly strengthened using the effective version of the theorem, however we do not pursue these issues here.

The second fact has to do with how number fields inject into finite fields. Let $K = \mathbb{Q}[X]/(f)$ be a number field. We represent the elements of K as polynomials in $\mathbb{Q}[X]$ of degree less than n where $n = \deg(f)$. Let p be a prime such that f has a non-zero root in $\mathbb{F}_p$ (i.e. there exists $\alpha \in \mathbb{Z}$ satisfying $f(\alpha) \equiv 0 \pmod{p}$). There is a natural map which sends most of the elements in K to $\mathbb{F}_p$: an element $g \in K$ is mapped to $g(\alpha) \bmod p$. We run into problems for polynomials $g \in K$ such that $g(\alpha)$ is a rational number whose denominator is divisible by p. We say that the map fails on such elements $g \in K$.

Fortunately the language of ideals provides a convenient method for classifying elements on which the map fails. Let $\mathcal{O}_K$ be the ring of integers of K. The fact that f has a non-zero root in $\mathbb{F}_p$ means that there exists a prime ideal $\mathfrak{P} \subseteq \mathcal{O}_K$ of norm p (the norm of an ideal $\mathfrak{P}$ is the cardinality of $\mathcal{O}_K/\mathfrak{P}$). The fact that $\mathcal{O}_K/\mathfrak{P}$ is isomorphic to $\mathbb{F}_p$ means that there is a natural map $\psi_p : \mathcal{O}_k \to \mathbb{F}_p$. Let $\hat{K}_p$ be the localization of $\mathcal{O}_K$ at the ideal $\mathfrak{P}$. The ring $\hat{K}_p$ contains elements of the form $\frac{a}{b}$ where $a \in \mathcal{O}_K$ and $b \in \mathcal{O}_K \setminus \mathfrak{P}$. The map ψ_p extends to all of $\hat{K}_p$ by defining $\psi_p(\frac{a}{b}) = \psi_p(a)/\psi_p(b)$. Observe that $\psi_p(b) \neq 0$ since $b \notin \mathfrak{P}$. The elements in $K \setminus \hat{K}_p$ are exactly those on which the map fails. The elements in $\mathfrak{P}' = \hat{K}_p \cdot \mathfrak{P}$ are exactly those which the map sends to 0 ($\ker(\psi_p) = \mathfrak{P}'$).

The following lemma shows that the norm of x provides a necessary condition for testing if x is mapped to 0 mod p or if the map fails on x.

Lemma 2.4.3 *Let $K = \mathbb{Q}[X]/(f)$ be a number field and let p be a prime such that f has a non-zero root in $\mathbb{F}_p$. Consider an element $x \in K$ whose norm is $\mathcal{N}(x) = \frac{a}{b}$, $\gcd(a, b) = 1$. Then $\psi_p(x) = 0$ implies that p divides a. Similarly, $x \notin \hat{K}_p$ implies that p divides b.*

Proof We first prove the lemma for the case $x \in \mathcal{O}_K$. Let $\mathfrak{I}$ be the ideal generated by x, i.e. $\mathfrak{I} = x\mathcal{O}_K$. Suppose $\psi_p(x) = 0$. Then, by definition $\mathfrak{I} \subseteq \mathfrak{P}$. This implies

that the norm of $\mathfrak{P}$ denoted by $\mathcal{N}(\mathfrak{P})$ divides $\mathcal{N}(\mathfrak{I})$. Hence, p divides $\mathcal{N}(\mathfrak{I})$. Since $\mathcal{N}(\mathfrak{I})$ is a power of $\mathcal{N}(x)$ it follows that p divides $\mathcal{N}(x)$.

When $x \in \hat{K}_p$ there exists an integer t relatively prime to p such that $tx \in \mathcal{O}_K$. Then the above paragraph proves the result for tx and hence for x.

When $x \in K \setminus \hat{K}_p$ there exists an integer t divisible by p such that $tx \in \mathcal{O}_K$ but $tx \notin \mathfrak{P}$. If $\mathcal{N}(tx) = a \in \mathbb{Z}$ then p does not divide a and therefore $\mathcal{N}(x)$ satisfies the conditions of the lemma. □

As an example one can readily verify that the lemma holds when $K = \mathbb{Q}$. To see that the condition in the above lemma is necessary but not sufficient consider the number field $K = \mathbb{Q}[i]$ where $i^2 = -1$. One can define a map $\psi_5 : \mathbb{Q}[i] \to \mathbb{F}_5$ by mapping i to $2 \in \mathbb{F}_5$. Then $\psi_5(2 - i) = 0$ but $\psi_5(2 + i) \neq 0 \pmod 5$ even though $\mathcal{N}(2 + i) = 5$.

The third fact is of a more combinatorial nature. For a set S of rational numbers and a prime p we denote by S_p the set

$$S_p = \left\{ \frac{a}{b} \bmod p \;\middle|\; \frac{a}{b} \in S \text{ and } b \not\equiv 0 \pmod p \right\}$$

The lemma says that for a non-negligible number of primes p the set S_p is not too small.

Lemma 2.4.4 *Let S be a finite set of rationals and let $\mathcal{P}$ be a finite set of primes. Suppose that all $x \in S$ satisfy height$(x) < B$ where $|\mathcal{P}| > 4 \log B$. Then at least a fourth of the primes in $\mathcal{P}$ satisfy*

$$|S_p| \geq \frac{\frac{1}{8}|\mathcal{P}|}{4 \log B + \frac{|\mathcal{P}|}{|S|}}$$

Proof For each prime p we define the undirected graph $G_p = (V_p, E_p)$. The vertices in V_p are the elements $\frac{a}{b} \in S$ for which p does not divide b. The edge set E_p is defined by

$$\{x, y\} \in E_p \quad \Longleftrightarrow \quad x \equiv y \pmod p$$

Notice that the cardinality of S_p is simply the number of connected components in G_p.

Since all $x \in S$ satisfy height$(x) < B$ we know that the denominators of all elements in S are divisible by at most $\log B$ prime factors. We say that a prime in $\mathcal{P}$ is bad if it divides more than half the denominators in S. Suppose that more than half the primes in $\mathcal{P}$ are bad. Then the total number of prime factors dividing all denominators in S is at least $\frac{1}{4}|\mathcal{P}| \cdot |S|$. By assumption, this is bigger than $|S| \log B$ which is impossible. Hence, at least half the primes in $\mathcal{P}$ are not bad. Let $\mathcal{P}'$ be this subset of primes.

Suppose $p \in \mathcal{P}'$ is a prime such that $|S_p| \leq T$. Then the number of connected components in G_p is at most T. Furthermore, since $p \in \mathcal{P}'$, the number of vertices in G_p is at least $\frac{1}{2}|S|$. Notice that since E_p is an equivalence relation on V_p the connected components of G_p are all complete graphs. We give a lower bound on the number of edges in G_p. Let r be the number of components in G_p and $s_1, \ldots, s_r$ be the sizes of the components. Then $\frac{1}{2}|S| \leq \sum_{i=1}^{r} s_i \leq |S|$. The number of edges in G_p is

$$\sum_{i=1}^{r} \binom{s_i}{2} \geq \frac{1}{2r}(\sum_{i=1}^{r} s_i)^2 - \frac{1}{2}\sum_{i=1}^{r} s_i \geq \frac{|S|^2}{8r} - \frac{|S|}{2} \geq \frac{|S|^2}{8T} - \frac{|S|}{2}$$

The first inequality follows from the fact that for integers $x_1, \ldots, x_r$ whose sum is m the Cauchy-Schwartz inequality implies that

$$\sum_{i=1}^{r} \binom{s_i}{2} = \frac{1}{2}\sum_{i=1}^{r} s_i^2 - \frac{m}{2} \geq \frac{m^2}{2r} - \frac{m}{2}$$

Since all elements $x \in S$ satisfy height$(x) < B$ we know that all pairs $(x, y) \in S$ satisfy height$(x - y) < B^2$. Since B^2 has at most $2 \log B$ factors, the edge (x, y) can appear in the graph G_p for at most $2 \log B$ primes p. Therefore, the total number of edges in all graphs G_p, $p \in \mathcal{P}'$ is at most

$$\binom{|S|}{2} 2 \log B \leq |S|^2 \log B$$

On the other hand, suppose that at least half the primes $p \in \mathcal{P}'$ satisfy $|S_p| \leq T$. Then the total number of edges in all graphs G_p, $p \in \mathcal{P}'$ is at least

$$\frac{1}{2}|\mathcal{P}'| \left[\frac{|S|^2}{8T} - \frac{|S|}{2} \right]$$

We therefore obtain the inequality

$$\frac{1}{2}|\mathcal{P}'|\left[\frac{|S|^2}{8T}-\frac{|S|}{2}\right] \leq |S|^2 \log B$$

which leads to

$$T \geq \frac{\frac{1}{4}|\mathcal{P}'|}{4\log B + \frac{|\mathcal{P}'|}{|S|}} \geq \frac{\frac{1}{8}|\mathcal{P}|}{4\log B + \frac{|\mathcal{P}|}{|S|}}$$

Hence, at least half the primes in $\mathcal{P}'$ must satisfy $|S_p| \geq T$ for the above value of T. Since $\mathcal{P}'$ contains at least half the primes in $\mathcal{P}$ the lemma is proved. □

Let $K = \mathbb{Q}[X]/(f)$ be a number field. Define $\mathcal{P}_f$ to be the set of primes p for which f has a non-zero root in $\mathbb{F}_p$. For each $p \in \mathcal{P}_f$ define the map $\psi_p : \hat{K}_p \to \mathbb{F}_p$ as in the discussion preceding Lemma 2.4.3. Given a set $S \subseteq K$ and a prime p define

$$S_p = \left\{\psi_p(x) \mid x \in S,\ x \in \hat{K}_p\right\}$$

We use Lemmas 2.4.4 and 2.4.3 to prove the following corollary:

Corollary 2.4.5 *Choose integer constants n, c. Let $\mathcal{P}$ be a set of size at least $2^n/n^c$ containing primes in $\mathcal{P}_f$ which are less than 2^n. Let $S \subseteq K$ be a set of size $|S| > 2^n/n^c$ where height$(\mathcal{N}(x)) \leq 2^{n^c}$ for all $x \in S$. Then at least a fourth of the primes in $\mathcal{P}$ satisfy*

$$|S_p| \geq \frac{1}{40} \cdot \frac{2^n}{n^{2c}}$$

Proof Define $S' = \{\mathcal{N}(x) \mid x \in K\} \subseteq \mathbb{Q}$. For $p \in \mathcal{P}$ let S'_p be defined as in Lemma 2.4.4. We claim that

$$\forall p \in \mathcal{P} \ : \quad |S_p| \geq |S'_p|$$

To see this define the map $h : S'_p \to S_p$ as follows: given $a \in S_p$ there exists $x_a \in S$ such that $a = \mathcal{N}(x_a) \bmod p$. Set $h(a) = \psi_p(x_a) \in S_p$. The following two observations show that the map h is well defined. First $\psi_p(x_a)$ is well defined since $x_a \in \hat{K}_p$. This follows from Lemma 2.4.3 since $\mathcal{N}(x_a) \in S'_p$ implies that the denominator of $\mathcal{N}(x_a)$ is not divisible by p. Second, if $x_1, x_2 \in S$ both satisfy $\mathcal{N}(x_1) \equiv \mathcal{N}(x_2) \equiv a \pmod p$

then $\psi_p(x_1) = \psi_p(x_2)$. This again follows from Lemma 2.4.3. Hence the value of $h(a)$ does not depend on which preimage of a in S is chosen. This shows that the map h is well defined. Next we show that it is injective (one-to-one). If $h(a) = h(b)$ then $\psi_p(x_a) = \psi_p(x_b)$, but then $\mathcal{N}(x_a) \equiv \mathcal{N}(x_b) \pmod{p}$ by Lemma 2.4.3. Hence, $a = b$. We have thus shown that S'_p injects into S_p and therefore $|S_p| \geq |S'_p|$.

Applying Lemma 2.4.4 to the set S'_p and $\mathcal{P}$ will complete the proof of the corollary: at least a quarter of the primes in $\mathcal{P}$ satisfy

$$|S_p| \geq |S'_p| \geq \frac{\frac{1}{8}|\mathcal{P}|}{4\log 2^{n^c} + \frac{|\mathcal{P}|}{|S|}} \geq \frac{\frac{1}{8}|\mathcal{P}|}{4n^c + n^c} \geq \frac{1}{40}\frac{2^n}{n^{2c}}$$

□

2.4.2 Proof of the hardness criteria

We now turn to the proof of Theorem 2.4.1. Let $\{f_n\}_{n=1}^{\infty}$ be a c-sequence which can be computed by an algebraic algorithm. We show that this assumption implies that factoring is easy. This is done in two steps. First we construct a set of polynomials with low straight line complexity which form a c-sequence. This set of polynomials is then used to construct a set of integers with low straight line complexity and many integer factors. These integers satisfy the hypothesis of Theorem 2.3.1 implying that factoring is easy. The following two lemmas explain these two steps.

Lemma 2.4.6 *Let $\{f_n\}_{n=1}^{\infty}$ be a c-sequence of functions which can be computed by an algebraic algorithm. Then there exists a c-sequence of univariate polynomials $\{g_n\}_{n=1}^{\infty}$ in $\mathbb{Z}[X]$ satisfying $\tau(g_n) < n^d$ for some constant d.*

Proof The fact that $\{f_n\}_{n=1}^{\infty}$ can be computed by an algebraic algorithm implies that there is a constant d such that for all n the function f_n can be computed by an n^d-time constructible algebraic computation tree. Let $\mathcal{T}_n$ be the tree computing f_n. Let $v_1, v_2, \ldots, v_r$ be the nodes encountered on the path starting at the root and always branching left (v_1 is the root, v_r is a leaf). This is the path taken when all internal

nodes v along the computation path satisfy $p_v(x) \neq 0$. Here p_v is the polynomial associated with the node v. Define the rational function g'_n as:

$$g'_n(x) = \prod_{i=1}^{r} p_{v_i}(x)$$

Notice that p_{v_i} is a polynomial in $\mathbb{Q}[X]$ for $i < r$ however p_{v_r} may be a rational function. Therefore g'_n is a rational function in $\mathbb{Q}(X)$. Since the tree $\mathcal{T}_n$ is constructible in time n^d it follows that $\tau(g'_n) < n^{2d}$. Let $g'_n = P_n/Q_n$ where $P_n, Q_n \in \mathbb{Z}[X]$ and $\gcd(P_n, Q_n) = 1$. By Lemma 2.1.2 there exist polynomials $T_n \in \mathbb{Z}[X]$ such that $\tau(T_n P_n) \leq 4n^{2d}$. We define $g_n = T_n P_n$ and argue that $\{g_n\}_{n=1}^{\infty}$ is a c-sequence. This will prove the lemma.

Since $\{f_n\}_{n=1}^{\infty}$ is a c-sequence there exists a number field K such that $Z(f_n, K, 2^{n^c}) \geq 2^n/n^c$. We show that the same holds for g_n. Observe that if on input $x \in K$ the computation reaches a leaf $l \neq v_r$ then it must be the case that for some i we have $p_{v_i}(x) = 0$ (otherwise the subtree containing l would never be entered). Hence, if $f_n(x) = 0$ for some $x \in K$ then $p_{v_i}(x) = 0$ for some $i = 1, \ldots, r$ and therefore $g_n(x) = 0$. It follows that the zeroes of g_n satisfy

$$Z(f_n, K, 2^{n^c}) \subseteq Z(g_n, K, 2^{n^c})$$

implying that the cardinality of $Z(g_n, K, 2^{n^c})$ is at least 2^{n^c} as required.

Finally we argue that for all n the polynomials g_n have infinite support in K. Since g_n is a polynomial, it follows that if it has an infinite number of zeroes it must be identically zero. If g_n is zero then one of the p_{v_i} must be zero. If an internal node in an algebraic computation tree is identically zero then that node may be removed from the tree. Hence, without loss of generality we may assume that for $i = 1, \ldots, r-1$ all of the p_i are non-zero. Now, if p_r (the polynomial at the leaf) is zero then one can readily see that the function f_n does not have an infinite support in K. This follows from the fact that all but a finite number of inputs must reach the leaf v_r. Hence, if $p_r \equiv 0$ the function f_n will be zero for all but a finite number of inputs contradicting the fact that $\{f_n\}_{n=1}^{\infty}$ is a c-sequence. Hence, g_n has infinite support in K. $\square$

Lemma 2.4.7 *Let $\{g_n\}_{n=1}^{\infty}$ be a c-sequence of univariate polynomials in $\mathbb{Z}[X]$ satisfying $\tau(g_n) < n^c$. Then there exists a set of non-zero integers $\{y_n\}_{n=1}^{\infty}$ such that $\tau(y_n) \leq 3n^c$ and infinitely often $d(y_n, n) > \frac{1}{n^{3c+2}}$.*

Proof Since $\{g_n\}_{n=1}^{\infty}$ is a c-sequence there exists a number field K such that infinitely often $Z(g_n, K, 2^{n^c}) > 2^n/n^c$. Let $g_n \in \mathbb{Z}[X]$ be such a polynomial and $K = \mathbb{Q}[X]/(f)$. We show that evaluating $g_n(x)$ at a random point $x \in \mathbb{Z}$ is likely to produce an integer with many small prime factors and low straight line complexity.

Define $\mathcal{P}$ to be the set of primes $p < 2^n$ for which f has a non-zero root modulo p. By Tchebotarev's density theorem (Theorem 2.4.2) and the prime number theorem we know that $|\mathcal{P}| \geq \frac{1}{2}2^n/n^{c+1}$ for large enough n. For $p \in \mathcal{P}$ define

$$Z_p = \left\{\psi_p(x) \mid x \in Z(g_n, K, 2^{n^c})\right\} \subseteq \mathbb{F}_p$$

where $\psi_p : (\hat{K}_n)_p \to \mathbb{F}_p$ is the map defined in the discussion preceding Lemma 2.4.3. Applying Corollary 2.4.5 to $Z(g_n, K, 2^{n^c})$ and $\mathcal{P}$ shows that at least a fourth of the primes in $\mathcal{P}$ satisfy

$$|Z_p| \geq \frac{1}{40} \cdot \frac{2^n}{n^{2c}}$$

Denote this set of primes by $\mathcal{P}'$. Then $|\mathcal{P}'| \geq \frac{1}{8}2^n/n^{c+1}$. For a prime p we denote by $\hat{g}_n$ the polynomial g_n reduced modulo p (i.e. $\hat{g}_n \in \mathbb{F}_p[X]$). Observe that any integer M such that $M \bmod p$ is in Z_p satisfies

$$g_n(M) \equiv \hat{g}_n(M \bmod p) = \hat{g}_n(\psi_p(x)) = \psi_p(g_n(x)) = 0 \pmod p$$

where $x \in Z(g_n, K, 2^{n^c})$ satisfies $\psi_p(x) = M \bmod p$. Therefore, given a prime $p \in \mathcal{P}'$ the number of positive integers $M < 2^{2n^c}$ such that p divides $g_n(M)$ is at least $2^{2n^c}/40n^{2c}$. It follows that the number of pairs (M, p) where $M < 2^{2n^c}$, $p \in \mathcal{P}'$ and p divides $g_n(M)$ is at least

$$2^{2n^c} \cdot \frac{1}{8}\frac{2^n}{n^{c+1}} \cdot \frac{1}{40}\frac{1}{n^{2c}} > \frac{2^{2n^c}2^n}{\frac{1}{2}n^{3c+2}}$$

Let S be the set of positive integers M less than 2^{2n^c} for which $d(g_n(M), n) \geq 1/n^{3c+2}$ (i.e. at least $1/n^{3c+2}$ of the prime less than 2^n divide $g_n(M)$). Then the set S

contains more than $2^{2n^c}/n^{3c+2}$ elements. To see this observe that if $|S| < 2^{2n^c}/n^{3c+2}$ then the number of pairs (M, p) defined above is at most

$$|S| \cdot 2^n + \left(2^{2n^c} - |S|\right) \frac{2^n}{n^{3c+2}} \leq \frac{2^{2n^c} 2^n}{n^{3c+2}} + \frac{2^{2n^c} 2^n}{n^{3c+2}} \leq 2\frac{2^{2n^c} 2^n}{n^{3c+2}}$$

contradicting the lower bound given above.

Even though all M in S satisfy $d(g_n(M), n) \geq 1/n^{3c+3}$ there are two problems which might prevent a specific M from being used. First, it may happen that $g_n(M) = 0$. Second it may be the case that the straight line program for g_n encounters a division by zero on input M. Since $\tau(g_n) < n^c$ the degree of g_n is at most 2^{n^c} and therefore it has at most 2^{n^c} roots. Further, during the evaluation of $g_n(M)$ a division by zero is encountered only if M is a root of one of the denominators in the straight line computation for g_n. Since the degree of the denominators is bounded by 2^{n^c} a division by zero will be encountered for at most $n^c 2^{n^c}$ inputs. Therefore there are at most

$$2^{n^c} + n^c 2^{n^c}$$

elements in S which are inappropriate for our purposes. For large enough n this value is less than the size of S which is at least $2^{2n^c}/n^{3c+2}$. Therefore, there must exist an $M_n \in S$ for which $g_n(M_n) \neq 0$ and the straight line computation for g_n does not encounter a division by zero on input M_n. Notice that since $M_n < 2^{2n^c}$ we have that $\tau(M_n) < 2n^c$. This integer M_n satisfies:

$$\tau(g_n(M_n)) \leq 2n^c + n^c = 3n^c \quad ; \quad d(g_n(M_n, n) \geq \frac{1}{n^{3c+2}} \quad ; \quad g(M_n) \neq 0$$

The resulting sequence $\{g(M_n)\}_{n=1}^{\infty}$ is the one we set out to construct. This completes the proof of the lemma. □

Theorem 2.4.1 is an immediate consequence of these two lemmas and Theorem 2.3.1.

2.5 Applications

We now describe several applications of Theorem 2.4.1. All the results in this sections assume the hardness of factoring. We will not mention this assumption in the theorems explicitly.

Theorem 2.4.1 can be used to deduce lower bounds on the algebraic complexity of many polynomials. Here we give two examples.

Theorem 2.5.1 *The following polynomials in $\mathbb{Z}[X]$ can not be computed by an algebraic algorithm running in time $(\log n)^{O(1)}$:*

$$(x-1)(x-2)\cdots(x-n)$$
$$(x^2-2)(x^2-8)\cdots(x^2-2n^2)$$

2.5.1 Recurrence relations

Recurrence relations can usually be computed by short straight line programs. Given a rational function f in $\mathbb{Q}(X)$ we define the recurrence relation $T_n(x)$ by

$$T_n(x) = f(T_{n-1}(x)) \quad \text{where} \quad T_0(x) = x$$

Theorem 2.4.1 has two applications for such general relations. The first is an immediate corollary. The second is an interesting application of the hardness of factoring to an open problem regarding periods of rational functions.

Theorem 2.5.2 *Let c be some integer constant and K a number field. The function $T_n(x)$ can have at most $2^n/n^c$ roots of norm less than 2^{n^c} in K for large enough n.*

Let $T_n(x)$ be a recurrence relation as above. We say that x is R-preperiodic if there exists $r < R$ such that $T_r(x) = T_R(x)$. In other words, x is R-preperiodic if the sequence

$$x, T_1(x), T_2(x), \ldots$$

becomes periodic after at most R steps. We say that x is preperiodic if it is preperiodic for some R. Northcott [Nor50] proved in 1950 that a fixed recurrence relation T_n has

only finitely many preperiodic points in $\mathbb{Q}$. Moreover, these values can be computed effectively given the rational function defining T_n. Northcott's theorem holds over any fixed number field as well.

It is much harder to obtain a *uniform* bound on the number of preperiodic points (i.e. a bound independent of T_n itself). For some recent results regarding this problem see [FPS95] and the references therein. Theorem 2.4.1 can be used to derive some sort of "uniform" bound on the number of R-preperiodic points. We denote by $P(T_n, K, R)$ the number of R-preperiodic points of T_n of norm at most 2^R in the number field K:

$$P(T_n, K, R) = \left| \left\{ x \in K \;\middle|\; x \text{ is } R\text{-preperiodic and height}(\mathcal{N}(x)) < 2^R \right\} \right|$$

Theorem 2.5.3 *Let K be a fixed number field and let c be some constant. For large enough n, all recurrence relations $T_n(x) = f(T_{n-1}(x))$ with $\tau(f) < n^c$ satisfy*

$$P(T_n, K, n^c) < 2^n$$

Proof Set $R = n^c$ and let $T_n(x) = f(T_{n-1}(x))$ be a recurrence relation where $\tau(f) < n^c$. Consider the function $g(x) = \prod_{r=0}^{R-1}(T_R(x) - T_r(x))$. Observe that $\tau(g) < n^{3c}$. Clearly if $x \in K$ is an R-preperiodic points of T_n then $g(x) = 0$. Hence, if $P(T_n, K, n^c) > 2^n$ infinitely often then by Theorem 2.4.1 we obtain a contradiction to the hardness of factoring. □

We note that the bound provided by Theorem 2.5.3 is much better than the bound obtained by simply considering the degree of the polynomial $g(x)$ defined in the proof.

A closely related problem is that of obtaining a uniform bound on the number of torsion points on Abelian varieties. A bound is considered uniform if it only depends on the dimension of the variety and the degree of the number field in question. The existence of such a bound is known as the *Torsion Conjecture*. See [Sil92] for a survey article on this topic.

The hardness criteria of Theorem 2.4.1 can be applied to this problem as well. One can use the theorem to obtain bounds on the number of torsion points on an elliptic curve over a number field. More generally, one can use the same technique

to obtain bounds on the number of torsion points on the Jacobian variety associated with certain hyper-elliptic curves. These results are not as strong as one would like and we therefore chose to omit the details.

2.5.2 Chromatic Polynomials

As a final application we turn to the *chromatic polynomial* of a graph G. Recall that this is a polynomial $P(G;x)$ that counts the number of possible ways to color the graph G with x colors. We can prove the following theorem:

Theorem 2.5.4 *Let G_n be a sequence of n-vertex graphs. Suppose G_n requires at least $n/\log^c n$ colors for some $c > 0$. Then the polynomial $P(G_n;x)$ can not be evaluated in time $\log^{\Omega(1)}$ by an algebraic algorithm.*

Proof Since the graphs can not be colored by $n/\log^c n$ colors the value of the chromatic must be zero for all integers $1, 2, \ldots, n/\log^c n$. □

Read and Tutte [RT88] point out that chromatic polynomials are often "laborious" to compute. In a sense this theorem explains why: if they were easy to compute for graphs with large chromatic number, then factoring would be easy.

2.6 History/Related work

The straight line complexity of integers and polynomials has been studied by several authors. Using a counting argument, Demelo and Svaiter [MS96] show that almost all integers have large straight line complexity. More precisely, they show that for all $\epsilon > 0$ the set of integers N for which

$$\tau(N) < \frac{\log N}{(\log\log N)^{1+\epsilon}}$$

has density zero. The straight line complexity of integers is used by Blum, Shub and Smale [BSS89] in their work on the problem $P \neq NP$ in the algebraic settings.

A closely related complexity measure for integers is the *addition chain complexity* [Str90]. An addition chain for an integer N is a straight line computation for N which only uses addition and subtraction. Multiplication and division is not allowed. The addition chain complexity of an integer is the shortest addition chain computing that integer. See Knuth [Knu81, pp. 465–466] for some properties of the addition chain complexity of integers. Computing the addition chain complexity of an integer was shown to be NP-hard by [DLS81].

Addition chains arise naturally when one tries to exponentiate numbers. For instance $g^{15} \pmod p$ is computed as

$$g \cdot g^2 \cdot g^4 \cdot g^8 \cdot g^{12} \cdot g^{14} \cdot g^{15} \pmod p$$

However, the same result can be obtained more efficiently as

$$g \cdot g^2 \cdot g^3 \cdot g^6 \cdot g^{12} \cdot g^{15} \pmod p$$

Hence, one way of speeding up computations that involve exponentiations (e.g. RSA, Diffie-Hellman protocol) is to find good addition chains for the exponents. Bod and Coster [BC90] describe an approximation algorithm for determining the addition chain complexity of a given integer.

Throughout this chapter we always assumed that the computation trees at hand were uniform. In other words, the trees could always be constructed in polynomial time by a Turing Machine. It made no difference if the tree had exponential size as long as it was constructible. Proving lower bounds on uniform computations is virtually impossible. We therefore had to rely on an unproven assumption, namely the hardness of factoring. We note that for non-uniform computation trees there are many results proving absolute lower bounds which do not depend on unproven assumptions. A classic result shows that an algebraic decision tree which determines if a collection of n integers contains an integer more than once must have depth at least $\Omega(n \log n)$(see [BO83]).

2.7 Summary

The objective of this chapter was to define and study algebraic algorithms. We noted that several classic number theoretic algorithms can be modeled as algebraic algorithms. In characteristic zero we presented a criteria for when a function can not be computed by a polynomial time algebraic algorithm. The validity of this criteria rested on a classic cryptographic assumption: most integers can not be factored in polynomial time.

Our hardness criteria has several number theoretic consequences. These examples illustrate that the hardness of factoring can be used to derive number theoretic facts. The idea of using the hardness of factoring to "prove" main-stream number theoretic conjectures seems like a method worth pursuing. The hardness of factoring is a very deep statement and it is therefore likely to have have many interesting number theoretic consequences.

A curious point is that we did not really use the full power of the hardness of factoring assumption. The factoring algorithm implied by Theorem 2.4.1 only uses simple algebraic operations. It may certainly be possible that the hardness of factoring can be proved in such a restricted model of computation. As we have demonstrated, such a lower bound result would have interesting consequences. The problem of proving a lower bound in the more general circuit model seems much harder.

Chapter 3

Algorithms for Black-Box Fields

An algebraic structure is often defined as a set of operators acting on some universe. Usually there is no reference as to how the elements in the universe are represented. One can design algorithms for such an abstract algebraic structure by providing the algorithm with oracles for the various operators. We refer to such a representation of an algebraic structure as a *black box representation*. The most widely studied structure given in this fashion is the black box group [BS84].

In this chapter we study *fields* given in a black box representation. We refer to such fields as black box fields, or BBF for short. The definition of black box fields will be given in Section 3.1. For now we give a high level description. Let K be a BBF. Intuitively speaking, the elements of K are represented as arbitrary binary strings. For an element $x \in K$ we denote by $[x]$ the binary string representing the element x. We refer to $[x]$ as the *black box representation* of x. A black box field algorithm has access to oracles that given $[x]$ and $[y]$ compute the black box representation of the sum $[x+y]$ and product $[xy]$. Similarly, there is an oracle that given $[x], [y]$ will output "true" if and only if $x = y$. Finally, the algorithm is provided with an oracle that given $x \in K$ will output $[x]$.

We will be most interested in the following problem: let p be a prime and $K = \mathbb{F}_p$ be a finite field given as a BBF. Find an algorithm that takes as input the black box representation of a field element $[\alpha]$ and outputs an integer $0 \leq a < p$ such that

$a \equiv \alpha \pmod{p}$. We refer to this problem as the black box field problem, or BBFP for short. A trivial algorithm for this problem is to test all elements of $\mathbb{F}_p$ one by one by using the equality test oracle. The running time of this algorithm is $O(p)$. We are interested in finding algorithms for BBFP whose running time is substantially less than p.

Our main results will be randomized algorithms for BBFP whose expected running time is sub-exponential in $\log p$. Our algorithms are based on a technique due to Maurer [Mau94]. The existence of sub-exponential algorithms for BBFP is surprising when contrasted with a result of Nechaev [Nec94] and Shoup [Sho95]. They considered the equivalent problem to BBFP over groups, i.e. where elements can be added, but not multiplied. They show that for the group $G = \mathbb{Z}/p\mathbb{Z}$ the best algorithm for finding a hidden element must take time $\Omega(\sqrt{|G|})$.

Our sub-exponential algorithms for BBFP have several cryptographic consequences. The first application shows that any *algebraically homomorphic* cryptosystem can be broken in sub-exponential time. Such cryptosystems are desirable since they enable non-interactive two-player secure function evaluation [AF90]. These concepts will be defined in Section 3.3.1. One may view this result as a general cryptanalytic tool: to show that a cryptosystem can be broken in sub-exponential time it suffices to show that it is algebraically homomorphic.

An important motivation for studying the black box field problem is that algorithms for BBFP can be used to prove the security of the Diffie-Hellman secret key exchange protocol [DH76]. In Section 3.3.2 we will show that efficient algorithms for BBFP imply that breaking the Diffie-Hellman protocol is equivalent to computing discrete log. Proving the equivalence of breaking the Diffie-Hellman protocol and computing discrete-log is one of the oldest problems in public key cryptography. The existence of a sub-exponential algorithm for BBFP has the following consequence: let G be a group in which the discrete log problem can not be solved in sub-exponential time. Then in the group G the Diffie-Hellman protocol can not be broken in sub-exponential time. The group generated by the points of an elliptic curve over a finite field is an example of a group for which there is no known sub-exponential algorithm

for computing discrete log[1] (this is the main motivation for using elliptic curve cryptosystems [Kob87b, Mil85]). Hence, our results show that if computing discrete log in the group of points of an elliptic curve is hard then the Diffie-Hellman protocol in such groups is secure. The existence of such a reduction demonstrates another advantage of elliptic curve cryptosystems.

Finally in Section 3.4 we consider an equivalent of BBFP over the rationals. We show that solving BBFP over the rationals is as hard as factoring integers. The proof relies on the hardness criteria of the previous chapter. This negative result suggests that performing computations over rational black box fields is much harder than over finite black box fields.

3.1 Black-box fields

A black box field is an abstract algebraic construct motivated by cryptographic applications. We begin by giving a precise definition of black box fields (BBF) and the black box field problem (BBFP).

Definition 3.1.1 *A* black box field *is a six-tuple:* (p, n, h, F, G, T) *where* p *is a prime and* n *is a positive integer representing the encoding length. The functions* h, F, G, T *are defined as follows:*

1. *The function* $h : \{0,1\}^n \to \mathbb{F}_p$ *associates a field element with every* n*-bit binary string. The function* h *is surjective, i.e. every field element is represented by at least one binary string.*

2. *The functions* $F, G : \{0,1\}^n \times \{0,1\}^n \to \{0,1\}^n$ *perform addition and multiplication. They satisfy the following relations:* $h(F(x,y)) = h(x) + h(y)$ *and* $h(G(x,y)) = h(x)h(y)$.

3. *The function* $T : \{0,1\}^n \times \{0,1\}^n \to \{$`true,false`$\}$ *tests equality of two black box elements:* $T(x,y) =$ `true` *if and only if* $h(x) = h(y)$.

[1] A sub-exponential algorithm is known for the rare event when the curve is supersingular [MV91].

Notice that an element $x \in \mathbb{F}_p$ can be represented by many different n-bit binary strings since $h^{-1}(x)$ is a set of arbitrary cardinality. Throughout the paper we will use $[x]$ to denote some binary string representing the field element x, i.e. $h([x]) = x$. The functions F and G compute $[x+y]$ and $[xy]$ given $[x]$ and $[y]$. This is consistent with the notation used in the introduction. We usually refrain from mentioning the functions h, F, G, T explicitly. When we say that $\mathbb{F}_p$ is given as a black box field we assume that these functions have already been agreed upon. As an abuse of notation we occasionally write $[x] \in \mathbb{F}_p$ which is to be understood as saying that $[x]$ is a black box representation of the field element $x \in \mathbb{F}_p$.

In Section 3.3 we give examples where black box fields arise naturally. To familiarize the reader with the concept of a black box field we present a few simple algorithms for such fields. The first is computing the inverse: given $[x] \in \mathbb{F}_p$ we wish to compute $[x^{-1}]$. This can easily be done by observing that $[x^{-1}] = [x^{p-2}]$. Using repeated squaring this requires $O(\log p)$ applications of the multiplication function. Since in a black box field one can add, subtract, multiply, divide and test equality we obtain the following fact.

Fact 3.1.1 *Let $f : \mathbb{F}_p^n \to \mathbb{F}_p$ be a function computable by an algebraic algorithm in time T. Then the function*

$$f'\Big([x_1], \ldots, [x_n]\Big) = [f(x_1, \ldots, x_n)]$$

is computable in time $O(T \cdot \log p)$ in a black box field.

The extra $\log p$ factor comes from the fact that division takes $O(\log p)$ time. For example one can compute $[\sqrt{x}]$ given $[x]$ in polynomial time in $\log p$. This uses Shanks algorithm which, as we noted in the previous chapter, can be modeled as a polynomial time algebraic algorithm. Hence, when the algorithm is applied to $[x]$ it outputs $[\sqrt{x}]$. As a final example we observe that it is possible to manipulate polynomials over black box fields. A polynomial $p(x) \in \mathbb{F}_p[x]$ over a black box field is a formal polynomial of the form:

$$p(x) = [a_n]x^n + [a_{n-1}]x^{n-1} + \ldots + [a_1]x + [a_0] \ .$$

One can readily verify that two such polynomials $p(x), q(x)$ can be added, subtracted, multiplied and even divided in polynomial time. Consequently it is possible to do computations over algebraic extensions of black box fields.

Definition 3.1.2 *Let (p, n, h, F, G, T) be a black box field for some prime p. We denote the map sending x to some $[x]$ by $[]$. The* black box field problem *is the following: find an algorithm A that given p and oracles for $F, G, T, []$ and an element $[\alpha] \in \mathbb{F}_p$ finds α explicitly. Formally, $A^{F,G,T,[]}([\alpha]) = a$ where $a \equiv \alpha \pmod{p}$. The algorithm is said to run in polynomial time if it runs in time $\log^{O(1)} p$. The algorithm is sub-exponential if it runs in sub-exponential time in $\log p$.*

The main goal of this paper is to provide algorithms for solving the black box field problem. We first show that it is possible to obtain a small number of bits that uniquely define elements of the finite field $\mathbb{F}_p$. This will show that the black box field problem is information theoretically easy.

Conjecture 3.1.2 *Let p be a prime and set $k = \lceil 2\log^2 p \rceil$. For an element $x \in \mathbb{F}_p$ define the signature of x as the vector:*

$$sig(x) = \left(\left(\frac{x}{p}\right), \left(\frac{x+1}{p}\right), \ldots, \left(\frac{x+k}{p}\right) \right) \ .$$

where $\left(\frac{x}{p}\right)$ is the Legendre symbol of x over p. Then we conjecture[2] *that for sufficiently large p, any two distinct elements x, y in $\mathbb{F}_p$ satisfy $sig(x) \neq sig(y)$.*

Problems similar to Conjecture 3.1.2 were studied by Davenport [Dav33]. The identity $\left(\frac{x}{p}\right) \equiv x^{\frac{p-1}{2}} \pmod{p}$ shows that given $[x]$ one can compute $\mathrm{sig}(x)$ in a black box field using $O(\log^3 p)$ applications of the oracles. Assuming the conjecture is true, $\mathrm{sig}(x)$ provides enough information to recover x. Unfortunately, there is no known polynomial time algorithm for finding x given $\mathrm{sig}(x)$. In fact, Damgård [Dam88] suggested using this sequence as a pseudo-random sequence.

[2] *Notice that the conjecture implies that $sig(1) \neq sig(2)$. This means that there must exist an $1 \leq a \leq k+2$ such that a is a quadratic non-residue modulo p. This is known to hold under the generalized Riemann hypothesis when $k > 2\log^2 p$ (see [Bac90]). That explains our particular choice of k in the conjecture.*

The argument above shows that one can not hope to obtain an information theoretic lower bound on the number of oracle calls needed to solve BBFP. Assuming conjecture 3.1.2 holds, a polynomial number of oracle calls are sufficient to completely constrain the hidden element $[x]$. This situation is substantially different than the one which occurs in black box groups. As was mentioned in the introduction, to solve the equivalent problem on the group $\mathbb{Z}/p\mathbb{Z}$ one must make at least $\Omega(\sqrt{p})$ oracle queries to obtain enough information about the hidden element.

3.2 Algorithms for black-box fields

In this section we describe two sub-exponential algorithms for BBFP. The results will depend on an assumption which is often used in computational number theory:

Smoothness assumption

Let $L_\alpha(p)$ be the function $L_\alpha(p) = \exp\left(\log^\alpha p \, \log\log^{1-\alpha} p\right)$. For an integer x let $d(x)$ be the largest prime divisor of x. If $d(x) < \alpha$ we say that x is α-smooth. The assumption we need is that integers chosen uniformly in the range $[p+1-2\sqrt{p}, p+1+2\sqrt{p}]$ satisfy

$$\Pr_x[d(x) < L_\alpha(p)] > \frac{1}{L_{1-\alpha}(p)^{1-\alpha+o(1)}}$$

for any fixed $\alpha > 0$. When the integer x is chosen in the range $[1, p]$ this assumption is known to be true (see [CEP83, DeB66]). The assumption when $\alpha = \frac{1}{2}$ is necessary for the running time analysis of the elliptic curve factoring method of Lenstra [Len87].

3.2.1 A sub-exponential algorithm for BBFP

This section is devoted to the proof of the following theorem:

Theorem 3.2.1 *Let K be a finite field of size p given as a black box field. Under the smoothness assumption, BBFP can be solved in expected time*

$$L_{\frac{1}{2}}(p)^{2+o(1)} = \exp\left((2+o(1))\sqrt{\log p \, \log\log p}\right) \quad .$$

and $O(\log p)$ space.

The proof of Theorem 3.2.1 uses a technique similar to the one used by Maurer [Mau94]. Before proving Theorem 3.2.1 we state some simple facts regarding elliptic curves. We denote by $E_{a,b}(p)$ the set of points $(x, y) \in \mathbb{F}_p^2$ on the curve $y^2 = x^3 + ax + b$ plus a point O called the point at infinity. It is well known [Sil86] that there is a natural Abelian group structure defined on the points in $E_{a,b}(p)$. Thus, $E_{a,b}(p)$ forms an Abelian group. Given two points $P_1, P_2 \in E_{a,b}(p)$ we denote their sum in this group by $P_1 + P_2$.

Fact 3.2.2 *Let $E_{a,b}$ be an elliptic curve over the black box field $\mathbb{F}_p$. Let $P_1 = (x_1, y_1)$ and $P_2 = (x_2, y_2)$ be two points on $E_{a,b}$. Let $P_3 = P_1 + P_2 = (x_3, y_3)$. Then given $[x_1], [y_1], [x_2], [y_2]$ it is possible to compute $[x_3], [y_3]$ in polynomial time in $\log p$.*

Proof The values x_3, y_3 can be expressed as algebraic expressions of x_1, y_1, x_2, y_2, a, b. See [Sil86, pp. 58–59] for a list of these expressions. Thus, $[x_3], [y_3]$ can be computed from $[x_1], [y_1], [x_2], [y_2]$ using the addition and multiplication oracles. □

Throughout the section we adopt the following notation: for a point $P = (x, y) \in E_{a,b}(p)$ we denote by $[P]$ the point P whose coordinates are given as black box elements, i.e. $[P] = ([x], [y])$. Fact 3.2.2 shows that given points $[P_1] = ([x_1], [y_1])$ and $[P_2] = ([x_2], [y_2])$ it is possible to construct $[P_1 + P_2] = ([x_3], [y_3])$ in polynomial time. Similarly, using repeated doubling one can construct the point $[kP]$ given a point $[P]$ and an integer k.

Fact 3.2.3 *Let $E_{a,b}$ be an elliptic curve over $\mathbb{F}_p$. Then the group $E_{a,b}$ can be generated by two points. Furthermore, two random points P_1, P_2 generate the group $E_{a,b}$ with probability at least $\Omega(1/\log^2 p)$.*

Proof The structure of the group $E_{a,b}$ is known to be $E_{a,b} \simeq \mathbb{Z}_n \times \mathbb{Z}_m$ for some m, n where $m|n$ (see [Sil86]). Thus $E_{a,b}$ can be generated by two points. The number of pairs which generate $\mathbb{Z}_n \times \mathbb{Z}_m$ is lower bounded by $\Omega(\varphi(nm)^2)$. The result now follows since always $\varphi(x)/x > \Omega(1/\log x)$. □

Proof of Theorem 3.2.1 The basic idea is to generate random elliptic curves $E_{a,b}$ over $\mathbb{F}_p$ until a curve with a smooth order is found. It is well known that the number of points on a random elliptic curve over $\mathbb{F}_p$ is approximately uniformly distributed in the range $[p+1-2\sqrt{p},\ p+1+2\sqrt{p}]$ (see [Len87]). Hence, by the smoothness assumption, after an expected $\exp\left((\frac{1}{2}+o(1))\sqrt{\log p\ \log\log p}\right)$ tries we will have generated a curve $E_{a,b}$ such that the largest prime divisor of $|E_{a,b}|$ is less than $\exp\left(\sqrt{\log p\ \log\log p}\right)$.

Let $E_{a,b}$ be an elliptic curve over $\mathbb{F}_p$ for which the largest prime divisor of $|E_{a,b}|$ is less than d for some $d > 0$. The number of points on the curve $E_{a,b}$ can be found in polynomial time using Schoof's algorithm [Sch85]. We show that using this curve it is possible to solve the black box field problem over $\mathbb{F}_p$ in time $O(d^2 \log p)$. This will prove the theorem. Note that since the prime factors of $|E_{a,b}|$ are small, they can be found in the required time bound.

By Fact 3.2.3 we can find a pair of points P_1, P_2 which generate the group $E_{a,b}$ by picking them at random. If it so happens that P_1, P_2 do not generate $E_{a,b}$ then the algorithm will fail and we will know that P_1, P_2 were poorly chosen. Fact 3.2.3 shows that in an expected polynomial number of attempts the pair P_1, P_2 will work.

Given two points $P_1 = (x_1, y_1), P_2 = (x_2, y_2)$ which generate $E_{a,b}$ we show how to solve the black box field problem. Let $[x]$ be a black box representation of some field element $x \in \mathbb{F}_p$. Given $[x]$ our goal is to construct x explicitly, i.e. find an integer w such that $w \equiv x \pmod{p}$. This can be done in three steps:

1. Embed $[x]$ in the curve $E_{a,b}$, i.e. find $[y] \in \mathbb{F}_p$ so that the point $[P] = ([x], [y])$ is on $E_{a,b}$.

2. Find integers α and β such that $[P] = [\alpha P_1 + \beta P_2]$. Such integers exist since P_1, P_2 generate $E_{a,b}$.

3. Calculate $\alpha P_1 + \beta P_2$ explicitly. The x coordinate of the resulting point, which is an explicit field element, is the required value.

We now explain how to carry out steps 1 and 2. In step 1 we are looking for $[y] \in \mathbb{F}_p$ such that $y^2 = x^3 + ax + b$. Given $[x]$ one can clearly construct $[x^3 + ax + b]$. The required $[y]$ can be found by taking the square root of the element $[x^3 + ax + b]$.

This is possible since in Section 3.1 we saw that given $[z] \in \mathbb{F}_p$ there is an algorithm for constructing $[\sqrt{z}]$ in random polynomial time. If the above element does not have a square root in $\mathbb{F}_p$ we can run through the entire computation using $[\hat{x}] = [x + r]$ for some randomly chosen r. After a constant number of tries it will be possible to embed $[\hat{x}]$ in the curve.

Next we explain how to find α and β as required in step 2. Recall that $E_{a,b} \simeq \mathbb{Z}_n \times \mathbb{Z}_m$ where $n = q_1^{\gamma_1} \cdots q_r^{\gamma_r}$, $m|n$ and all the primes q_i are less than d. To find α, β we use a simple generalization of the Pohlig-Hellman algorithm [PH78] for discrete log in groups of smooth order. For simplicity we assume that all the γ_i are equal to 1. The Pohlig-Hellman algorithm generalizes to the case where $\gamma_i > 1$ as well. The method for finding α, β is to construct $\alpha, \beta \pmod{q_i}$ for all $i = 1, \ldots, r$ and then use chinese remaindering to find α, β.

Let q_i be a prime dividing n and let $k = n/q_i$. Notice that the points kP_1, kP_2 have order at most q_i. Therefore, since the point kP is in the group generated by kP_1, kP_2 there must exist integers $0 \leq \alpha_i, \beta_i < q_i$ such that

$$[kP] = [\alpha_i kP_1 + \beta_i kP_2]$$

These integers can be found in time $O(q_i^2)$ by trying all possible pairs α_i, β_i in the range $[0, q_i]$. Testing if the above condition holds is possible using the equality testing oracle.

As was stated above, the $(\alpha_i, \beta_i) \pmod{q_i}$ can be combined using chinese remaindering to obtain integers α, β such that $P = \alpha P_1 + \beta P_2$. Since $|E_{a,b}|$ has at most $O(\log p)$ prime factors and they are all less than d the running time of this procedure is at most $O(d^2 \log p)$. This completes the proof of the theorem. □

3.2.2 A two step algorithm

We improve the algorithm of the previous section by improving the brute force search step. The improvement relies on a certain additional assumption. Namely, we assume that the Diffie-Hellman function over the group of points of an elliptic curve can be

computed by an $L_{1/3}(p)$ *algebraic* algorithm. Though this assumption is likely to be false (there are no sub-exponential algebraic algorithms for computing discrete log) the theorem is quite useful. It provides a more efficient reduction from computing discrete-log to breaking Diffie-Hellman (Theorem 3.3.4 part (2)).

Theorem 3.2.4 *Suppose $DH_g(P_1, P_2)$ can be computed in time $L_{1/3}(p)$ by an* algebraic *algorithm where $P_1 = (x_1, y_1)$, $P_2 = (x_2, y_2)$ and g are points on an elliptic curve $E_{a,b}(p)$. Then under the smoothness assumption, BBFP in a finite field of size p can be solved in expected time*

$$L_{\frac{1}{3}}(p)^{2+o(1)} = \exp\left((2+o(1))\sqrt[3]{\log p \ \log\log^2 p}\right) \ .$$

Proof We improve the algorithm of Theorem 3.2.1 by using a two step algorithm. Let $\mathbb{F}_p$ be a black box field and let $[x] \in \mathbb{F}_p$. The idea is to generate an elliptic curve $E_{a,b}$ over $\mathbb{F}_p$ with n points such that the largest prime divisor of n is $L_{2/3}(p)$. By the smoothness assumption this can be done in expected $L_{1/3}(p)^{1/3+o(1)}$ time. An important point is that the prime factors of n can be found using the elliptic curve factoring algorithm [Len87] in time $L_{1/3}(p)^{1+o(1)}$.

As in the proof of Theorem 3.2.1 we can find, in expected polynomial time, two points P_1, P_2 that generate $E_{a,b}$. Similarly we can find a $[y] \in \mathbb{F}_p$ such that the point $[P] = ([x], [y])$ is in $E_{a,b}$. Recall that it may be necessary to shift $[x]$ by a random $r \in \mathbb{F}_p$ to make sure that $[y]$ exists. We now wish to find integers α, β such that $[P] = [\alpha P_1 + \beta P_2]$. Let q be a prime divisor of n and let $k = n/q$. We show how to find a pair α_q, β_q satisfying:

$$[kP] = [\alpha_q k P_1 + \beta_q k P_2]$$

for each prime divisor q of n. As in the proof of Theorem 3.2.1 this is sufficient for recovering the point P and finding x.

When $q < L_{1/3}(p)^{1/2}$ the values α_q, β_q can be found by a brute force search in time $L_{1/3}(p)$. When $q > L_{1/3}(p)^{1/2}$ we may assume that q^2 does not divide n. This follows from the fact that the number of integers less than p that are divisible by a square of a prime bigger than $L_{1/3}(p)^{1/2}$ is negligible. Hence, if n is divisible by q^2 we can look

for a different curve. Within an expected constant number of tries a curve satisfying the above requirement will be found.

Let G_q be the group of points of order q in $E_{a,b}$. Since q^2 does not divide n we know that G_q is a cyclic group of size q. Furthermore, since q is prime, at least one of kP_1 or kP_2 must generate this group. Henceforth we assume that kP_1 generates G_q. We wish to find an integer α_q such that $[kP] = [\alpha_q \cdot kP_1]$.

To find α_q we use the algorithm recursively. Let $R = kP_1$. Then R generates G_q. We show that the group G_q may be regarded as the black box field $\mathbb{F}_q$. An element $u \in \mathbb{F}_q$ is represented in the following way: let $uR = (x, y) \in E_{a,b}$. Then $[u]$ is defined to be $[u] = ([x], [y])$.

The axioms of black box fields are easily seen to hold using this representation. The only axiom that needs some explanation is the one stating that two black box elements can be multiplied. Given two elements $[u] = uR = ([x_1], [y_1])$ and $[v] = vR = ([x_2], [y_2])$ we wish to form $[uv] = uvR = ([x_3], [y_3])$. By assumption, the function $\mathrm{DH}_R(uR, vR) = uvR$ is computable in time $L_{1/3}(p)$ by an *algebraic* algorithm. Consequently, Fact 3.1.1 implies that given $[x_1], [y_1], [x_2], [y_2]$ it is possible to construct $[uv] = ([x_3], [y_3])$ in time $L_{1/3}(p) \log p$.

The idea is now to solve the black box field problem in the smaller field $\mathbb{F}_q$. Since $kP \in G_q$ the point $[kP]$ is a valid black box representation of an element in $\mathbb{F}_q$. We can apply the algorithm of Theorem 3.2.1 to solve BBFP in $\mathbb{F}_q$. The output of the algorithm is an explicit element $\alpha_q \in \mathbb{F}_q$ satisfying $kP = \alpha_q R$ as required.

Since $q < L_{2/3}(p)$ solving BBFP in $\mathbb{F}_q$ using Theorem 3.2.1 takes time $L_{1/2}(q)^{2+o(1)} < L_{1/3}(p)^{2+o(1)}$. Thus, the entire running time of the algorithm is bounded by $L_{1/3}(p)^{2+o(1)}$. □

Theorem 3.2.4 is a rare example where the elliptic curve method produces an algorithm with a running time of $L_{1/3}(p)$. Such running times are usually associated with the number field sieve [LL94].

3.3 Applications

The black box field problem arises naturally in cryptography. In this section we discuss two applications of algorithms for BBFP to cryptography.

3.3.1 Algebraically homomorphic encryption schemes

Informally a cryptosystem is algebraically homomorphic if given the encryption of two plain-texts x, y one can construct the encryption of the plain-texts $x + y$ and xy in polynomial time. This is captured in the following definition.

Definition 3.3.1 *Let d, e, k be positive integers denoting the plain-text length, cipher-text length and key-length respectively. Let $\mathbb{Z}_n$ be the ring of integers modulo n where $\lfloor \log_2 n \rfloor = d$. The ring $\mathbb{Z}_n$ constitutes the set of plain-texts. Let (E, D) be an encryption scheme, i.e.*

$$D : \mathbb{Z}_n \times \{0,1\}^k \to \{0,1\}^e \quad \textit{and} \quad E : \{0,1\}^e \times \{0,1\}^k \to \mathbb{Z}_n$$

where E, D are deterministic polynomial time computable functions. Furthermore, $D(E(x, K_e), K_d) = x$ for some key pair (K_e, K_d) generated by a probabilistic polynomial time key generation algorithm. The encryption scheme is said to be algebraically homomorphic *if there exist two probabilistic expected polynomial time algorithms $A, M : \{0,1\}^e \times \{0,1\}^e \to \{0,1\}^e$ such that for all $x, y \in \mathbb{Z}_n$ and encryption keys K_e:*

$$A(E(x, K_e), E(y, K_e)) = E(x + y, K_e) \quad \textit{and} \quad M(E(x, K_e), E(y, K_e)) = E(xy, K_e)$$

As an example we note that for the RSA cryptosystem, given the encryption of two plain-texts x and y one can easily construct the encryption of xy by simply multiplying the two given cipher-texts. RSA is not known to be algebraically homomorphic even though it supports one of the required operations.

Algebraically homomorphic encryption schemes have several applications which make them desirable. Most importantly, they enable two players two perform non-interactive secure function evaluation. See [AF90] for the appropriate definitions.

The existence of such functions was posed as an open question in [FM89, pp. 6–7]. Unfortunately our sub-exponential algorithm for BBFP shows that any algebraically homomorphic encryption scheme can be broken in sub-exponential time.

Theorem 3.3.1 *Suppose that BBFP in a finite field of size p can be solved in time $T_{BBF}(p)$. Then any algebraically homomorphic encryption scheme (D, E) over a plain-text ring of size n can be broken in expected time*

$$O\left(T_{BBF}(n) + \exp((1+o(1))\sqrt[3]{\log n \log^2 \log n}\right)$$

Proof To simplify the exposition we assume that n is square free. This restriction can be easily lifted using methods of Pohlig and Hellman [PH78] which will be discussed in the next section(Lemma 3.3.2). Since one can factor integers in expected $\exp((1+o(1))\sqrt[3]{\log n \log^2 \log n})$ time (see [LL94]) it is possible to factor the plain-text ring into a direct product of finite fields: $\mathbb{Z}_n = \prod_{i=1}^{s} \mathbb{F}_{p_i}$ where the p_i are distinct primes.

Let K_e, K_d be some encryption/decryption key pair. Given $E(x, K_e)$ we wish to find x in the required time bound. For each p_i we define the black box field (p_i, e, h, A, M, T) as follows: $h(x) = D(x, K_d) \pmod{p_i}$. Notice that for $a \in \mathbb{F}_{p_i}$ the string $[a]$ can be any string w satisfying $w = E(a', K_e)$ where $a \equiv a' \pmod{p_i}$. The addition and multiplication functions A, M are simply the corresponding functions used in Definition 3.3.1. Hence A, M can be computed in expected polynomial time. To implement the equality testing oracle T observe that $a' \equiv b' \pmod{p_i}$ if and only if $(a' - b') * n/p_i \equiv 0 \pmod{n}$. Thus, for $a, b \in \mathbb{F}_{p_i}$ given $[a] = E(a', K_e)$ and $[b] = E(b', K_e)$ testing if $a = b$ is done by testing if $E((a' - b') * n/p_i, K_e) = E(0, K_e)$. The string $E((a'-b')*n/p_i, K_e)$ can be computed from $[a], [b]$ in expected polynomial time. We have thus shown that (p_i, e, h, A, M, T) is a black box field and the functions A, M, T can be computed in probabilistic polynomial time.

Given $E(x, K_e)$ we use the algorithm for BBFP in each of the black box fields $\mathbb{F}_{p_i}$ to recover $x \mod p_i$. Using chinese remaindering we can now recover x. The total expected running time of this procedure is $\sum_{i=0}^{s} O(T_{BBF}(p_i)) < O(T_{BBF}(n))$. □

Theorem 3.3.1 can be generalized to work for more general finite commutative rings. We do not pursue these generalizations here. An immediate corollary of Theorem 3.3.1 is that the sub-exponential algorithm for BBFP (Theorem 3.2.1) enables one to break any algebraically homomorphic encryption scheme in sub-exponential time. This fact may also be viewed as a general tool for cryptanalysis. A cryptanalyst which is faced with a new cryptosystem might try to prove that it is algebraically homomorphic. If he succeeds then our techniques immediately give a method for breaking the system[3].

3.3.2 The Diffie-Hellman protocol

The Diffie-Hellman secret key exchange protocol [DH76] is one of the oldest public key protocols. The protocol enables two parties to perform a secret key exchange. We briefly describe the protocol using an arbitrary finite cyclic groups G. One can take as the group G the multiplicative group $\mathbb{Z}_p^*$ of residues modulo a prime p. This group contains $p-1$ elements.

Say Alice and Bob wish to perform a secret key exchange. They agree ahead of time on some generator $g \in G$ of the group G. The generator g is made public. Both Alice and Bob secretly pick random integers $0 < a, b < |G|$. Alice sends to Bob the value g^a and Bob sends to Alice the value g^b. Both Alice and Bob can now compute the value $g^{ab} = (g^a)^b = (g^b)^a$ which is used as their secret key.

A passive eavesdropper, Eve, who listens in on the conversation hears g^a and g^b. To discover the secret key, Eve has to compute g^{ab}. Thus, we define the Diffie-Hellman function as:

$$\mathrm{DH}_g(g^a, g^b) = g^{ab} \ .$$

Note that the cyclic group G is implicit in this notation. In their original paper, Diffie and Hellman claimed that for $G = \mathbb{Z}_p^*$ computing the function $\mathrm{DH}_g(x, y)$ is hard. Many other types of groups have been suggested by various authors. Examples include

[3]This does not hold for probabilistic encryption schemes. The reason is that given two cipher-texts one can not test if they are the encryption of the same plain-text. As a result, the equality testing oracle can not be implemented.

the multiplicative group of residues modulo a composite number [McC88, MY91], elliptic curves over finite fields [Kob87b, Mil85], the Jacobian of a hyperelliptic curve over a finite field [Kob88] and the class group of imaginary quadratic fields [BW88]. In all these groups the function $\mathrm{DH}_g(x, y)$ is believed to be hard to compute.

To stress the importance of demonstrating the hardness of the Diffie-Hellman function we note that many other cryptographic protocols rely on it for their security. Some examples are the ElGamal public key cryptosystem [EG85], Shamir's three pass protocol [Kob87a, pp. 96–97], and the Bellare-Micali non-interactive oblivious transfer scheme [BM84]. See [SS95] for proofs of the equivalence of breaking these protocols and computing the Diffie-Hellman function.

A long standing open problem in cryptography is the question of whether computing $\mathrm{DH}_g(x, y)$ is as hard as computing discrete-log for the group G. The discrete log-function is defined as $\mathrm{Dlog}_g(g^a) = a$ where a is an integer in the range 0 to $|G|-1$. It is not difficult to see that an oracle for $\mathrm{Dlog}_g(z)$ enables one to compute $\mathrm{DH}_g(x, y)$ in polynomial time. The hard question is whether the converse holds: given an oracle for computing $\mathrm{DH}_g(x, y)$ can one compute $\mathrm{Dlog}_g(z)$ in polynomial time (in $\log |G|$)? Surprisingly, there have been very few results on this problem [dB88, Mau94, Wol95]. Recently Maurer [Mau94] obtained a beautiful result showing that given a polynomial number of advice bits which depend only on $|G|$ one can compute $\mathrm{Dlog}_g(x)$ in polynomial time given an oracle for $\mathrm{DH}_g(x, y)$. Unfortunately, computing these advice bits takes exponential time.

We show how algorithms for BBFP can be used to reduce $\mathrm{Dlog}_g(z)$ to $\mathrm{DH}_g(x, y)$. First we state an observation due to Pohlig and Hellman [PH78] which shows that the ability to compute discrete log in groups of prime order is sufficient for computing discrete log in arbitrary groups.

Lemma 3.3.2 *Let G be a finite cyclic group for which the factorization of $|G|$ is known. Suppose that for any element $g \in G$ of prime order p one can compute $Dlog_g(x)$ in time $T(p)$. Then one can compute $Dlog_h(x)$ for any element $h \in G$ in time $T(|G|)\log^{O(1)}|G|$.*

Proof Let $|G| = \prod_{j=1}^{r} p_j^{\alpha_j}$ where p_j are distinct primes. We wish to compute

$\mathrm{Dlog}_g(x)$ for an element $x = g^a$ of G. The plan is to determine $a \bmod p_j^{\alpha_j}$ for all j. The chinese remainder theorem will then enable us to reconstruct a.

Let p be some prime divisor of $|G|$ and p^α be the highest power of p that divides $|G|$. Suppose we already know $b = a \bmod p^k$ for some $0 \leq k < \alpha$ (initially $b = k = 0$). We show how to determine $a \bmod p^{k+1}$.

First, define $h = g^{|G|/p}$. Note that the element h has prime order and hence generates a subgroup of G of prime order. Therefore, by assumption, we can compute $\mathrm{Dlog}_h(y)$ in time $T(p)$ (for y in the group generated by h). To find $a \bmod p^{k+1}$ compute the following value: $y = (xg^{-b})^{|G|/p^{k+1}}$.

Since $a - b$ is divisible by p^k the element y can be expressed as follows:

$$y = (xg^{-b})^{|G|/p^{k+1}} = g^{(a-b)|G|/p^{k+1}} = g^{\frac{|G|}{p}\frac{a-b}{p^k}} = h^{(a-b)/p^k} \ .$$

This shows that y is in the subgroup generated by h. As was stated above we can now compute $t = \mathrm{Dlog}_h(y)$. Clearly $t = (a-b)/p^k \bmod p$. This means that $a \equiv b + tp^k \pmod{p^{k+1}}$.

Thus, given $b = a \bmod p^k$ we were able to compute $b' = a \bmod p^{k+1}$. Iterating this procedure will eventually produce $a \bmod p^\alpha$. This shows that $a \bmod p_j^{\alpha_j}$ can be found for all j. Consequently a can be found by chinese remaindering. The total running time is $\sum_i \alpha_i T(p_i) \log^{O(1)} p \leq T(|G|) \log^{O(1)} |G|$. The inequality follows from the fact that T must satisfy $T(x) + T(y) \leq T(xy)$ since $T(p) > \Omega(\log p)$. □

The main connection between BBFP and the security of the Diffie-Hellman protocol is explained in the following theorem. The theorem shows that if BBFP can be solved quickly then an algorithm for breaking Diffie-Hellman will give an algorithm for computing discrete-log.

Theorem 3.3.3 *Suppose that BBFP over the field $\mathbb{F}_p$ can be solved in time $T_{BBF}(p)$. Let G be some finite cyclic group for which the factorization of $|G|$ is known. Suppose that $DH_g(x, y)$ can be evaluated in time $T_{DH}(|G|)$ for any $g \in G$ of prime order. Then for any $g \in G$ the function $Dlog_g(x)$ can be computed in time*

$$T_{BBF}(|G|) \cdot T_{DH}(|G|) \cdot \log^{O(1)} |G|$$

Proof Let g be a generator of G. We wish to compute $\mathrm{Dlog}_g(x)$ for some $x = g^a$. First we show that when G has prime order p the theorem is immediate. The ability to evaluate $\mathrm{DH}_g(x, y)$ transforms G into the black box field $\mathbb{F}_p$. An elements $b \in \mathbb{F}_p$ is represented as $[b] \doteq g^b$. Two elements can be added $[b + c] = g^b g^c$ and multiplied $[bc] = \mathrm{DH}_g(g^b, g^c)$. Observe that $[a] = x$. Hence, the algorithm for BBFP will output a on input x. By assumption, the black box field problem can be solved in time $T_{BBF}(|G|)$. The algorithm for solving BBFP will make at most $T_{BBF}(|G|)$ calls to the multiplication oracle. Each call requires a calculation of the Diffie-Hellman function which takes time $T_{DH}(|G|)$. Therefore, the total running time is $O(T_{BBF}(|G|) \cdot T_{DH}(|G|))$. Thus, the theorem is proved when G has prime order.

When $|G|$ is not prime we make use of Lemma 3.3.2. Let p be a prime dividing $|G|$ and let $h = g^{|G|/p}$. The group generated by h has prime order. By assumption, $\mathrm{DH}_h(x, y)$ can be evaluated in time $T_{DH}(p)$. Hence, by the previous paragraph, $\mathrm{Dlog}_h(y)$ can be computed in time $T_{BBF}(p) \cdot T_{DH}(p)$. Lemma 3.3.2 can now be applied to show that $\mathrm{Dlog}_g(x)$ can be computed in time $T_{BBF}(|G|) \cdot T_{DH}(|G|) \cdot \log^{O(1)} |G|$ for any $g \in G$. □

We note that when $|G|$ is square free the theorem can be strengthened to obtain a more direct reduction from computing discrete-log to breaking Diffie-Hellman. Namely, for a fixed g an algorithm for $\mathrm{DH}_g(x, y)$ can be converted to an algorithm for $Dlog_g(x)$.

Security of the Diffie-Hellman protocol

Theorem 3.3.3 shows that algorithms for BBFP give a reduction from computing discrete-log to Breaking the Diffie-Hellman protocol. In general we apply Theorem 3.3.3 in the following way: suppose discrete-log can not be computed in time $T_{DL}(|G|)$ in a group G. Then the Diffie-Hellman protocol can not be broken in time $\tilde{O}(T_{DL}(|G|)/T_{BBF}(|G|))$. Thus, the faster we can solve BBFP the more secure the Diffie-Hellman protocol is.

Since the algorithms for BBFP take sub-exponential time the above approach

can only be used in groups for which there is no sub-exponential time algorithm for discrete log. There are several groups which are suspected to have such a property. The groups are all constructed using a prime modulus p.

1. Let g be an element of $\mathbb{Z}_p^*$ which generates a subgroup of order $L_{1/3}(p)$. Let H_g be this subgroup. Discrete log in H_g is not known to be solvable in sub-exponential time in $|H_g|$. Notice that the best general discrete log algorithm in $\mathbb{Z}_p^*$ takes time $O(|H_g|)$. The best known discrete log algorithm for such groups is Shank's baby-step-giant-step [Coh91, pp.] which takes $O(\sqrt{|H_g|})$ time. We therefore say that H_g is a group for which there is no known sub-exponential algorithm for discrete log. The digital signature standard [Nat91] relies on the exponential time hardness of discrete log in H_g for its security.

2. Let $E_{a,b}$ be the group of points on an elliptic curve modulo p. The best algorithm for discrete log in such groups is Shank's baby-step-giant-step algorithm. These groups are another example where there is no known sub-exponential algorithm for discrete-log.

3. Let J be the Jacobian of a hyper-elliptic curve of genus g. When the genus g is fixed, there is no known sub-exponential algorithm for discrete log in J. Adleman, DeMarrais and Huang [ADH94] show that when the genus is at least $\log p$ the index calculus method can be adapted to give an $L_{\frac{1}{2}}(p^{2g+1})^{1+o(1)}$ algorithm for discrete log in J.

4. Finally we mention that all sub-exponential algorithms for group $\mathbb{Z}_p^*$ rely on the index calculus method and therefore require sub-exponential space. There is no known sup-exponential time and polynomial space algorithm for computing discrete log in $\mathbb{Z}_p^*$.

The algorithms for BBFP can be used to derive hardness results for breaking the Diffie-Hellman protocol for the groups mentioned above. These results are summarized in the following theorem.

Theorem 3.3.4 *Let p be a prime. Under the smoothness assumption we obtain the following results:*

1. *Let G be one of the groups $E_{a,b}, J, H_g$ over $\mathbb{F}_p$ and set $s = |G|$. If discrete log can not be computed in time $L_{1/2}(s)^{3+o(1)}$ in G then the Diffie-Hellman protocol can not be broken in time $L_{\frac{1}{2}}(s)$ in G.*

2. *If discrete log can not be computed in $L_{1/3}(p)^{3+o(1)}$ time in some group $E_{a,b}$ over $\mathbb{F}_p$ then there exists some group $E_{a,b}$ in which the Diffie-Hellman protocol can not be broken by an* algebraic *algorithm in time $L_{1/3}(p)$.*

3. *If discrete log can not be computed in time $L_{\frac{1}{2}}(p)^{3+o(1)}$ and polynomial space in the group $\mathbb{Z}_p^*$ then the Diffie-Hellman protocol can not be broken in time $L_{\frac{1}{2}}(p)$ and polynomial space in $\mathbb{Z}_p^*$.*

Proof First, we may assume that the factorization of $|G|$ is known. This factorization can be found using the number field sieve algorithm whose conjectured running time is $L_{1/3}(p)^{1+o(1)}$ [LL94]. Part (1) follows by combining Theorem 3.3.3 and Theorem 3.2.1. Part (2) follows by combining Theorem 3.3.3 and Theorem 3.2.4. Part (3) follows by observing that the algorithm of Theorem 3.2.1 and the reduction of Theorem 3.3.3 both require only polynomial space. □

Note that part (2) of the theorem relies on a weaker assumption than part (1). Unfortunately the conclusion is also weaker since we can only prove that $\mathrm{DH}_g(x, y)$ can not be computed by an *algebraic* algorithm.

3.3.3 The power of algebraic algorithms

Recall that in Section 2.2.3 (page 15) we defined the class of functions $\mathbb{P}_{alg}$ to be those functions over finite fields which can be computed by an algebraic algorithm. We posed the question $\mathbb{P} = \mathbb{P}_{alg}$ as an open problem. Here we show that the algorithms for BBFP can resolve a weaker question.

Define SUBEXP_{alg} to be the class of functions over finite fields which can be computed by a randomized algebraic algorithm running in sub-exponential time. More precisely SUBEXP_{alg} is the set of functions $f(p, x_1, \ldots, x_n)$ which can be computed by a randomized algebraic algorithm in time $L_{1/2}(\log p)^{2+o(1)}$. The class SUBEXP is defined analogously without the restriction that the algorithm computing f is algebraic.

Theorem 3.3.5 *$SUBEXP = SUBEXP_{alg}$*

Proof It suffices to prove $\text{SUBEXP} \subseteq \text{SUBEXP}_{alg}$. Let $f(p, x_1, \ldots, x_n)$ be a function computable in sub-exponential time by a randomized Turning Machine M. Recall that for a fixed prime p the function f is regarded as a function $f_p : \mathbb{F}_p^n \to \mathbb{F}_p$.

Observe that the sub-exponential algorithm for BBFP may be regarded as an algebraic computation tree of sub-exponential depth. To see this observe that an algorithm solving BBFP may only apply elementary arithmetic operations and equality tests to its input and can therefore be modeled as an algebraic computation tree. For a fixed prime p this tree has p leaves labeled with the elements of $\mathbb{F}_p$.

To compute the function f_p by an algebraic algorithm we first run the BBFP algorithm on each of the n inputs. At this point the machine constructing the algebraic computation tree knows the n inputs explicitly. It can then apply the machine M to these inputs to compute the answer. □

3.4 Black-box fields in characteristic 0

We now turn our attention to black box fields over the field of rational numbers $\mathbb{Q}$. We are provided with a black box representation of the rationals which supports the axioms of Definition 3.1.1. The black box field problem over $\mathbb{Q}$ can be stated as follows: given a black box representation of an integer $[x]$ between 1 and N find the integer x. The algorithm is said to be polynomial if it runs in time polynomial in $\log N$. We prove the following negative result:

Theorem 3.4.1 *BBFP over $\mathbb{Q}$ can not be solved in polynomial time, unless factoring is easy.*

Proof In a black box field the only operations allowed are additions, multiplications and comparisons. Thus, a computation in a black box field can be viewed as an algebraic computation tree. That is, given an input x_0 the computation proceeds as follows: at every internal node v in the tree some polynomial $f_v(x_0)$ is evaluated. If $f_v(x_0) = 0$ the computation branches to the left child of v otherwise the computation branched to the right child. The leaves of the tree are labeled with the output which is some constant between 1 and N.

Assume towards a contradiction that for any N there is an algorithm which solves BBFP over $\mathbb{Q}$ in time $\log^{O(1)} N$. That is, given the black box representation of an integer $0 < [x] \leq N$ the algorithm will find x. The algorithm defines an algebraic computation tree with N leaves. Since the algorithm is polynomial, the depth of the tree is at most $\log^d N$ for some $d > 0$. Furthermore, the straight line complexity of every polynomial f_v in the tree must satisfy $\tau(f_v) < \log^e N$ for some $e > 0$.

Consider the path $v_0, \ldots, v_m$ from the root to a leaf in the tree which corresponds to the input x which satisfies

$$f_{v_0}(x) \neq 0 \quad ; \quad f_{v_1}(x) \neq 0 \quad ; \quad \ldots \quad ; \quad f_{v_m}(x) \neq 0 \ .$$

That is $v_0, \ldots, v_m$ is the path from root to leaf in which every v_i is the right child of v_{i-1}. For $i = 0, \ldots, m$ let Z_{v_i} be the set of integer roots of f_{v_i} in the range $[1, N]$. An input x will not reach the leaf v_m if it is contained in some Z_{v_i}. Since only one integer in the range $[1, N]$ is allowed to reach the leaf v_m we know that $|\bigcup_{i=1}^{m} Z_{v_i}| = N - 1$. Hence, there must exist an i such that $|Z_{v_i}| > (N-1)/m$. Since $m < \log^d N$ we conclude that the polynomial f_{v_i} has at least $N/\log^d N$ integer roots.

Applying this argument for $N = 2^k$ where $k = 1, 2, \ldots$ we obtain a sequence of polynomials f_k where each f_k has at least $2^k/k^d$ integer roots and $\tau(f_k) < k^e$. Using Theorem 2.4.1 (page 22) this sequence contradicts the hardness of factoring. Hence, there can not exist a polynomial time algorithm for solving the black box field problem over $\mathbb{Q}$ unless factoring is easy. $\square$

The proof of Theorem 3.4.1 is based on the fact that an algorithm for BBFP over $\mathbb{Q}$ implies the existence of a family of polynomials with exponentially many roots. Naively one could argue that when these polynomials are evaluated at a point which is not a root the result requires exponential space. This does *not* imply a hardness result for BBFP over $\mathbb{Q}$ since the black box oracles are the ones who manipulate these exponentially long integers. They do this in constant time.

The theorem shows that the black box field problem over $\mathbb{Q}$ is as as hard as factoring. Hence, cryptographers who wish to construct algebraically homomorphic cryptosystems may try to construct them over the field of rational numbers.

3.5 Conclusions and open problems

we defined a new problem which we call the black box field problem or BBFP for short. We demonstrated several applications of BBFP to cryptography: (1) efficient algorithms for BBFP provide reductions from breaking the Diffie-Hellman protocol to computing discrete-logs. (2) algorithms for BBFP can be used as a cryptanalytic tool to break algebraically homomorphic cryptosystems. These applications demonstrate the importance of this problem.

We described two sub-exponential algorithms for solving BBFP. The first solves BBFP in time $L_{1/2}(p)$. The second solves BBFP in time $L_{1/3}(p)$ under certain (strong) assumptions. These algorithms were used to show that over elliptic curves the hardness of computing discrete-log implies the security of the Diffie-Hellman protocol. These results demonstrate an advantage of elliptic-curve cryptosystems over conventional ones. In addition we noted that for small sub-groups of $\mathbb{Z}_p^*$, such as the ones used by DSS [Nat91], our results show that the hardness of discrete-log implies the security of the Diffie-Hellman protocol. We have also shown that assuming factoring integers is hard, BBFP can not be solved in polynomial time over the rationals.

The problem of solving BBFP over a finite field in polynomial time is still open. We briefly describe a promising approach. Schoof's algorithm [Sch85] for counting

the number of points on elliptic curves over $\mathbb{F}_p$ can be made to work over black box fields. Given a curve $y^2 = x^3 + [a]x + [b]$ over $\mathbb{F}_p$ the algorithm will output an explicit integer which is the number of $\mathbb{F}_p$-points on the curve. Given a black box element $[j]$ we construct the curve $y^2 = x^3 + [a]x + [b]$ whose j-invariant is $[j]$. The number of points on this curve provides a lot of information about j. Currently it is not known how to reconstruct j from this information. Progress in this direction will be of great interest and is likely to yield a polynomial time algorithm for BBFP.

Chapter 4

Diffie-Hellman Bit Security

Cryptographic protocols for secure communications (e.g. SSL) are usually composed of two steps. The first involves a secret key exchange to set up a session key. This is usually done using a public key system such as RSA or the Diffie-Hellman protocol. The second step encrypts the message plain-text using the session key and a block cipher (e.g. DES, IDEA). Unlike public key systems, block ciphers are fast and can handle large volumes of data.

Using the above mechanism one has to agree before hand on how the key generated by the public key system is converted to a session key used by the block cipher. For instance, if the Diffie-Hellman system uses a prime of length 1024 bits, one may use the 128 most significant bits of g^{ab} as a key for the IDEA block cipher. This conversion may be insecure: an attacker, who may not be able to compute the whole g^{ab}, may nevertheless succeed in computing the part of the bits of g^{ab} and crack the session. Hence it is important to know if the most significant bits (MSB) of g^{ab} are secure from an adversary who knows both g^a and g^b. Despite their long history, the security of the MSB's has not been shown for Diffie-Hellman keys.

A number of cryptographic schemes that have been proposed in the literature and used are related to or based on the Diffie-Hellman function $\mathrm{DH}_g(g^a, g^b) = g^{ab}$. These schemes depend on the "hidden" nature of g^{ab}. For examples, we mention ElGamal's public key cryptosystem [EG85], Shamir's Message Passing scheme [Kob87a],

Bellare-Micali Non-interactive Oblivious Transfer [BM89] and Okamoto Conference Key sharing scheme [Oka88]. We define these systems in the next section.

In this chapter, we study the security of the MSB's of Diffie-Hellman key exchange schemes and the related schemes mentioned above. As usual we define an abstract problem, solve it and show how its solution proves the security of the MSB's of Diffie-Hellman key exchange. Our solution is based on lattice basis reduction techniques using the LLL algorithm.

4.1 Discovering a number given an MSB oracle

Throughout the chapter we use $x \bmod p$ to denote the unique integer a in the range $[0, p-1]$ satisfying $x \equiv a \pmod{p}$. We will often refer to the function $\mathrm{MSB}_k(x)$ where $0 \leq x < p$. Given a prime p the function $\mathrm{MSB}_k(x)$ is defined to be the integer t such that $(t-1) \cdot p/2^k \leq x < t \cdot p/2^k$. For example, $\mathrm{MSB}_1(x)$ is either 0 or 1 depending on whether x is smaller than or greater than $p/2$. For convenience we sometimes assume that $\mathrm{MSB}_k(x)$ is an integer z satisfying $|x - z| < p/2^{k+1}$.

To study the security of parts of the Diffie-Hellman secret we suggest studying the following abstract problem:

Hidden Number Problem: Fix p and k. Let be $\mathcal{O}_{\alpha,g}(x)$ be an oracle that on input x computes the k most significant bits of $\alpha g^x \bmod p$:

$$\mathcal{O}_{\alpha,g}(x) = \mathrm{MSB}_k(\alpha \cdot g^x \bmod p)$$

The task is to compute the hidden number α modulo p, in expected polynomial time in $\log p$, given access to the oracle $\mathcal{O}_{\alpha,g}(x)$. Clearly we wish to solve this problem with a small value of k as possible.

One could query the oracle at chosen values for x and we assume that the oracle always gives the correct answer (the problem is still open when the oracle is occasionally allowed to give wrong answers). Note that the way the oracle is queried imposes a severe restriction. Namely, the multiplier for the hidden α is an element of $\mathbb{Z}_p^*$ for

which the querying party knows the discrete logarithm to the base g. This restriction is quite crucial for the applications at hand.

Since the restriction above is difficult to deal with we usually avoid it by making the problem a bit harder yet easier to work with. Observe that if x is picked at random in the range $[0, p-1]$ then g^x is a random element of $\mathbb{Z}_p^*$. We therefore obtain an oracle

$$\mathcal{O}_\alpha(t) = \mathrm{MSB}_k(\alpha \cdot t \bmod p)$$

which can only be evaluated at randomly chosen inputs $t \in \mathbb{Z}_p^*$. The sampling version of the hidden number problem is: given $\mathcal{O}_\alpha(t_i)$ where all t_i are chosen uniformly and independently at random in $\mathbb{Z}_p^*$, find α. This version of the problem is the one we study (the only exception is Section 4.3 where the original hidden number oracle is used).

If one is allowed to query the oracle $\mathcal{O}_\alpha(t)$ at *chosen* values of t then recovering α is much simpler. The study of this unrestricted problem is closely related to proving the bit security of the RSA cryptosystem. Alexi, Chor, Goldreich and Schnorr [ACGS88] completely solve this problem. They show that when chosen queries are allowed, the hidden number problem can be solved when $k = 1$ even when the oracle is noisy: namely, the oracle computes the MSB only on $1/2 + \epsilon, \epsilon > 0$ fraction of x correctly. However for their method to work it is essential to be able to query the oracle at arbitrary points.

The above discussion exposes a scenario which is often encountered in learning theory. When one is given the ability to query an oracle at chosen inputs the learning problem is easy. However, given only random samples the problem is much harder. In this chapter we present some algorithms for solving the hidden number problem given random samples.

4.2 Using Square Root $\log p$ Bits

Solutions to the hidden number problem can be used to prove that computing a part of the Diffie-Hellman secret is as hard as computing the entire secret. We state our

first result regarding the hidden number problem and show how it applies to the problems at hand. The following theorem shows that the sampling version of the hidden number problem can be solved using $k = 3\sqrt{\log p}$ bits.

Theorem 4.2.1 *Let α be some integer in the range $[1, p-1]$. Let $\mathcal{O}$ be a function defined by $\mathcal{O}(t) = \mathrm{MSB}_k(\alpha t \bmod p)$ with $k = 3\lfloor\sqrt{\log_2 p}\rfloor$. There exists a deterministic polynomial time algorithm A such that*

$$\Pr_{t_1,\ldots,t_d}\left[A\Big(t_1,\ldots,t_d,\mathcal{O}(t_1),\ldots,\mathcal{O}(t_d)\Big) = \alpha\right] \geq \frac{1}{2}$$

Where $d = \sqrt{\log p}$ and $t_1,\ldots,t_d$ are chosen uniformly and independently at random from $\mathbb{Z}_p^*$.

The proof of the theorem is given in the next section. Here we discuss some applications of the theorem. We first derive the following result regarding the hardness of MSB's of a Diffie-Hellman secret.

Theorem 4.2.2 *Set $k = 3\sqrt{\log p}$. Given an efficient algorithm to compute $\mathrm{MSB}_k(g^{ab})$ from g^a, g^b, there is an algorithm to efficiently (in expected polynomial time) compute g^{ab} itself.*

Proof Let A be an efficient algorithm computing $\mathrm{MSB}_k(g^{ab})$ given g^a, g^b. Given g^a, g^b we wish to compute g^{ab}. Set $\alpha = g^{ab} \bmod p$ and $h = g^b$. Let $\mathcal{O}(x)$ be the function $\mathcal{O}(x) = \mathrm{MSB}_k(\alpha \cdot h^x \bmod p)$. Then

$$\mathcal{O}(x) = \mathrm{MSB}_k(\alpha \cdot h^x) = \mathrm{MSB}_k(g^{ab} \cdot (g^b)^x) = \mathrm{MSB}_k(\mathrm{DH}_g(g^{a+x}, g^b)) = A(g^{a+x}, g^b)$$

Hence, the algorithm A can be used to evaluate the function $\mathcal{O}(x)$ for arbitrary x. We obtain a hidden number problem which can be solved in expected polynomial time according to Theorem 4.2.1. This shows that $\alpha = g^{ab}$ can be found if the algorithm A exists. □

The theorem shows that the $3\sqrt{\log p}$ most significant bits of $DH_g(g^a, g^b)$ are secure, if breaking the Diffie-Hellman protocol is hard. For example, in the case of a 1024

bit prime, the first 96 bits are hard to compute. This is of some practical interest as it shows that the 128 MSB's of the Diffie-Hellman secret may be securely used as the session key (used by the block cipher).

Theorem 4.2.1 is general enough so as to apply to many other cryptographic schemes based on the Diffie-Hellman protocol. We first define these schemes and then prove some results regarding their bit security. Below, we assume that the generator g and the prime p have been agreed upon by Bob and Alice and all the required inverses exist.

ElGamal Public Key Encryption Bob picks a random x and publishes $y = g^x$ as his public key. To send a message m to Bob, Alice picks a random r and sends g^r, my^r. Bob can decode the message by computing $my^r/(g^r)^x$. To break the scheme one has to compute the function $EL_g(g^x, g^r, mg^{xr}) = m$.

Shamir Message Passing Scheme : To send a message m Alice picks a random r, and sends $y = m^r$ to Bob. Bob picks a random s and sends $z = y^s$ back to Alice. Alice sends $w = z^{r^{-1}}$ to Bob who computes $m = w^{s^{-1}}$. Here r^{-1}, s^{-1} denote inverses modulo $p - 1$. To break the scheme one computes $SH(g^{rs}, g^r, g^s) = g$.

Okamato Conference Key Sharing . Bob picks r at random and sends to Alice $x = g^r$. Alice picks a random s and sends $y = x^s$ back. Bob computes $y^{r^{-1}} = g^s$ which is the conference key they use. Since the conference key is determined by Alice's bits alone she can distribute the same key to all members of the conference. To break the scheme one computes the function $OK_g(g^{rs}, g^r) = g^s$.

The equivalence of the above functions to Diffie-Hellman was studied in [SS95]. To apply Theorem 4.2.1 to these schemes we first derive several relations satisfied by the associated functions.

Lemma 4.2.3 *The functions defined above satisfy the following relations:*

$$\begin{aligned} EL_g(g^{x+r}, g^y, mg^{xy}) &= m \cdot (g^{-y})^r \\ SH(g^{xy}, g^{x+rxy}, g^y) &= g \cdot (g^y)^r \qquad (when \ \ \gcd(yr+1, p-1) = 1\ ) \\ OK_g(g^{y(x+r)}, g^y) &= g^x \cdot g^r \end{aligned}$$

Proof These relations are derived as follows:

$$EL_g(g^{x+r}, g^y, mg^{xy}) = El_g(g^{x+r}, g^y, (mg^{-yr}) \cdot g^{y(x+r)}) = m \cdot (g^{-y})^r$$

To prove the second relation define $h = g^{yr+1}$. Then

$$SH(g^{xy}, g^{x+rxy}, g^y) = SH(h^{\frac{xy}{yr+1}}, h^{\frac{x+rxy}{yr+1}}, h^{\frac{y}{yr+1}}) = h = g \cdot (g^y)^r$$

The middle equality follows since $\frac{x+rxy}{yr+1} \cdot \frac{y}{yr+1} = \frac{xy}{yr+1}$ and hence h must be the value of the Shamir function on those three inputs. The third relation follows since

$$OK_g(g^{y(x+r)}, g^y) = g^{x+r} = g^x \cdot g^r$$

□

Lemma 4.2.3 provides the relations necessary to convert an oracle for recovering the MSB of the schemes above into a hidden number oracle. As a result Theorem 4.2.1 can be used to prove the security of the MSB's of these schemes.

Theorem 4.2.4 *Set* $k = 3\sqrt{\log p}$.

1. *Given an efficient algorithm to compute* $\mathrm{MSB}_k(m)$ *of the secret message* m *in the ElGamal public-key system there is an algorithm to efficiently (in expected polynomial time) compute the entire message.*

2. *Given an efficient algorithm to compute* $\mathrm{MSB}_k(m)$ *of the secret message* m *in Shamir's message passing scheme there is an algorithm to efficiently (in expected polynomial time) compute the entire message.*

3. *Given an efficient algorithm to compute* $\mathrm{MSB}_k(g^s)$ *of the conference key* g^s *in the Okamoto system there is an algorithm to efficiently (in expected polynomial time) compute the entire key.*

Proof We prove statement (2) of the theorem. Statements (1) and (3) follow using a similar argument. Let A be an efficient algorithm computing $\mathrm{MSB}_k(m)$ given m^x, m^y, m^{xy}. Set $g = m^y \bmod p$ and define $\mathcal{O}(r)$ to be the function $\mathcal{O}(r) = \mathrm{MSB}_k(m \cdot g^r \bmod p)$. Then

$$\mathcal{O}(r) = \mathrm{MSB}_k(m \cdot (m^y)^r) = \mathrm{MSB}_k\Big(SH(m^{xy}, m^{x+rxy}, m^y)\Big) = A(m^{xy}, m^{x+rxy}, m^y)$$

The third equality follows by Lemma 4.2.3. Notice that m^{x+rxy} can be computed as $m^x \cdot (m^{xy})^r$. It follows that given m^x, m^y, m^{xy} the algorithm A can be used to evaluate the function $\mathcal{O}(r)$ for arbitrary r. Hence, we obtain a hidden number problem which can be solved in expected polynomial time according to Theorem 4.2.1. This shows that the secret message m can be found if the algorithm A exists. □

A weaker version of Theorem 4.2.1 is developed in [FHK*88] for the purpose of breaking a truncated linear congruential pseudo random number generator. Our value of $k = 3\sqrt{\log p}$ is much better than the value obtained in Theorem 3.4 of [FHK*88]. The improved performance is due to the fact that we query the oracle at random points.

4.2.1 Proof of the square root $\log p$ bit theorem

We now turn to the proof of Theorem 4.2.1. For notational convenience we assume that the prime p and the generator g of $\mathbb{Z}_p^*$ are fixed. Throughout this section we let $n = \log p$.

The proof relies on rounding techniques in lattices. We therefore first briefly recall the notion of a lattice and cite some relevant results. A (full rank) lattice L is defined to be the set of points

$$L = \left\{ y \ : \ y = \sum_{i=1}^{d} t_i b_i \ , \ t_i \in \mathbb{Z} \right\}$$

where the b_i are linearly independent vectors in $\mathbb{R}^d$. The set $\{b_i\}_{i=1}^d$ is called the basis of the lattice and d is the dimension of the lattice. We denote the L_2 norm of a vector $v \in \mathbb{R}^d$ by $\| v \|$.

An important result due to Babai [Bab86] shows how given a lattice L and a point v one can find a lattice point which is approximately the closest to v. Using the lattice basis reduction algorithm of Lenstra, Lenstra and Lovasz [LLL82] he proves the following.

Theorem 4.2.5 (Babai) *Let L be a lattice of dimension d. Given a point $v \in \mathbb{R}^d$ there exists a polynomial time algorithm which finds a lattice point $w \in L$ such that*

$$\| v - w \| \leq 2^{\frac{d}{2}-1} \min\{\| v - b \| : \ b \in L\}$$

Proof of Theorem 4.2.1 We show a polynomial time algorithm for recovering the hidden number α. Set $d = \sqrt{\log p}$. We are given randomly chosen integers $t_1, \ldots, t_d$ and corresponding integers $a_1, \ldots, a_d$ such that for all $i = 1, \ldots, d$:

$$|(\alpha t_i \bmod p) - a_i| < p/2^k \tag{1}$$

where $k = 3\lfloor\sqrt{\log p}\rfloor$. To find the hidden α we construct the following lattice (the rows represent the basis vectors):

$$L = \begin{pmatrix} p & 0 & 0 & \ldots & 0 & 0 \\ 0 & p & 0 & \ldots & 0 & 0 \\ & \vdots & & & 0 & \vdots \\ 0 & 0 & 0 & \ldots & p & 0 \\ t_1 & t_2 & t_3 & \ldots & t_d & 1/p \end{pmatrix}$$

The dimension of this lattice is $d + 1$. We refer to the first d vectors in the basis as p-vectors. Notice that multiplying the bottom vector by α and subtracting the appropriate number of p-vectors produces a lattice vector

$$v_\alpha = (r_1, \ldots, r_d, \ \alpha/p)$$

where $|r_i - a_i| < p/2^k$ for all $i \leq d$. Define u to be the vector $u = (a_1, \ldots, a_d, 0)$. Then v_α is a lattice point whose distance from u is at most $\sqrt{d+1}\, p/2^k$. In other words,

$$\min\left\{ \| u - w \| \;\middle|\; w \in L \right\} \leq \sqrt{d+1}\, p/2^k$$

The following theorem shows that with high probability all lattice points which are this close to u have a special structure. Here the probability is over the random choice of $t_1, \ldots, t_d$.

Theorem 4.2.6 (Uniqueness Theorem) *Set $d = \lfloor \sqrt{\log p} \rfloor$, $k = 3\lfloor \sqrt{\log p} \rfloor$ and let α be a fixed integer in the range $[1, p-1]$. Choose integers $t_1, \ldots, t_d$ uniformly and independently at random in the range $[1, p-1]$. Let L be the lattice constructed as above and $u = (a_1, \ldots, a_d, 0)$ be a vector satisfying condition (1). Then with probability at least $\frac{1}{2}$ we have:*

$$\forall v \in L: \quad \| u - v \| < \frac{p}{2^{k-d}} \implies$$
$$v = (t_1\beta \bmod p, \ldots, t_d\beta \bmod p,\ \beta/p) \text{ where } \alpha \equiv \beta \pmod p$$

Proof Let β, γ be two integers. Define the *modular distance* between β and γ as

$$\mathrm{dist}_p(\beta, \gamma) = \min_{b \in \mathbb{Z}} |\beta - \gamma - bp|$$

For example, $\mathrm{dist}_p(1, p) = 1$. Suppose $\beta \neq \gamma \pmod p$ and they are both integers in the range $[1, p-1]$. Define

$$A = \Pr_t \left[\mathrm{dist}_p(\beta t, \gamma t) > 2p/2^{k-d} \right]$$

where t is an integer chosen uniformly at random in $[1, p-1]$. Then

$$A = \Pr_t \left[\frac{2p}{2^{k-d}} < (\beta - \gamma)t \bmod p < p - \frac{2p}{2^{k-d}} \right] = \frac{\left\lfloor p - \frac{2p}{2^{k-d}} \right\rfloor - \left\lceil \frac{2p}{2^{k-d}} \right\rceil}{p-1} \geq 1 - \frac{5}{2^{k-d}}$$

This follows since for every $x \in [\frac{2p}{2^{k-d}}, p - \frac{2p}{2^{k-d}}]$ there exists a t such that $(\beta - \gamma)t = x \pmod p$. In general, a lattice point v has the form

$$v = (\beta t_1 - b_1 p,\ \beta t_2 - b_2 p, \ldots, \beta t_d - b_d p,\ \beta/p)$$

for some integers $\beta, b_1, \ldots, b_d$. Suppose $\| v - u \| < p/2^{k-d}$. We show that with probability at least $\frac{1}{2}$ the vector v satisfies $\beta \equiv \alpha \pmod{p}$ and $\beta t_i - b_i p \in [0, p]$ for all i. Observe that if $\beta = \alpha \pmod{p}$, then $\beta t_i - b_i p \in [0, p]$ for all i. Otherwise at least one of the components of $v - u$ is bigger in absolute value than $p/2^{k-d}$.

Now, suppose $\beta \neq \alpha$. Then

$$\Pr\left[\| v - u \| > p/2^{k-d}\right] \geq \Pr\left[\exists i \ : \ \mathrm{dist}_p(t_i\beta, a_i) > p/2^{k-d}\right] \geq$$
$$\Pr\left[\exists i \ : \ \mathrm{dist}_p(t_i\beta, t_i\alpha) > 2p/2^{k-d}\right] = 1 - (1 - A)^d \geq 1 - \left(\frac{5}{2^{k-d}}\right)^d$$

Since $\beta \neq \alpha \pmod{p}$ there are exactly $p - 1$ values of $\beta \bmod p$ to consider. Hence, the probability there exists a lattice point contradicting the statement of the theorem is at most

$$(p-1) \cdot \left(\frac{5}{2^{k-d}}\right)^d < \frac{1}{2}$$

The last inequality follows from the fact that $d(k - d - \log_2 5) > \log p$. This completes the proof of the theorem. □

We now apply Theorem 4.2.5 to the lattice L and the vector u. According to the theorem we can find in polynomial time a lattice vector $w \in L$ such that

$$\| u - w \| \leq 2^{\frac{d+1}{2}-1} \min\{\| u - b \| : \ b \in L\} \leq 2^{\frac{d+1}{2}-1} \cdot \sqrt{d+1}\, p/2^k < p/2^{k-d}$$

By Theorem 4.2.6, with probability $\geq 1/2$, we have that $w = v_\alpha = (r_1, \ldots, r_d, \alpha/p)$. The last component of this vector contains the hidden element α. This completes the proof of Theorem 4.2.1. □

The technique above can be used to solve a more general problem. Let α, β be two integers in the range $[1, p-1]$. Let $\mathcal{O}(t)$ be the function defined by

$$\mathcal{O}(t) = \mathrm{MSB}_k(\alpha t + \beta \bmod p)$$

where $k = 3\sqrt{\log p}$. Then one can recover α, β in expected polynomial time from random samples of the function $\mathcal{O}(t)$. Given random integers $t_1, \ldots, t_d$ in the range

$[0, p-1]$ construct the lattice:

$$L = \begin{pmatrix} p & 0 & 0 & \dots & 0 & 0 & 0 \\ 0 & p & 0 & \dots & 0 & 0 & 0 \\ & \vdots & & & 0 & \vdots & 0 \\ 0 & 0 & 0 & \dots & p & 0 & 0 \\ t_1 & t_2 & t_3 & \dots & t_d & 1/p & 0 \\ 1 & 1 & 1 & 1 & 1 & 0 & 1/p \end{pmatrix}$$

This lattice has dimension $d+2$. Let b_{d+1} be the second to last vector in the lattice and b_{d+2} be the last vector. The vector $\alpha \cdot b_{d+1} + \beta \cdot b_{d+2}$ minus an appropriate number of p-vectors will be very close to the vector constructed from the oracles answers. Hence, Theorem 4.2.5 can be applied to recover α, β. The uniqueness theorem (Theorem 4.2.6) can be generalized to handle this more general problem as well.

4.3 The case of a small generator

The original formulation of the hidden number problem asks one to discover a hidden number α from an oracle which outputs the k most significant bits of $\alpha \cdot g^r \bmod p$. In the previous section we proved that setting $k = 3\sqrt{\log p}$ suffices. In this sections we show that when the generator g is small this result can be improved. For instance, when $g = 2$ only the most significant bit is needed, i.e. $k = 1$ suffices. This result leads us to suggest a new variant of the Diffie-Hellman protocol.

For a generator g of $\mathbb{Z}_p^*$ we define the significant bits function $SB_g(x \bmod p)$ to be an integer t such that $(t-1)p/g \leq x < tp/g$. Clearly $t \in [0, g-1]$ and therefore the function SB_g returns at most $\log_2 g$ bits of information. Notice that when $g = 2$ the function $SB_2(x)$ is the same as the $\mathrm{MSB}_1(x)$ function used in Section 4.1.

Theorem 4.3.1 *Let α be some integer in the range $[1, p-1]$. Let $\mathcal{O}$ be a function defined by $\mathcal{O}(x) = SB_g(\alpha g^x \bmod p)$ for some generator g of $\mathbb{Z}_p^*$. Then there exists an algorithm which given access to an oracle computing the function $\mathcal{O}$ can find α in polynomial time in $\log p$.*

Proof Let U and L be upper and lower bounds on α, i.e. $L \leq \alpha < U$. Initially we set $L = 0$ and $U = p$. The algorithm will iteratively decrease the gap between U and L until $U - L < 1$ in which case α is found. Throughout the algorithm we maintain that at the r'th iteration $L = (t-1) \cdot \frac{p}{g^r}$ and $U = t \cdot \frac{p}{g^r}$ for some integer $t \in [1, g^r]$. Initially $r = 0$.

Consider the r'th iteration. Then $U = t \cdot \frac{p}{g^r}$ and $L = (t-1) \cdot \frac{p}{g^r}$ for some integer t. Since $L \leq \alpha < U$ we have

$$0 \leq \alpha g^r - pt < p$$

The algorithm will now query the oracle at the point $x = r$. By definition, the oracle returns a number z such that

$$(z-1)\frac{p}{g} \leq \alpha g^r \bmod p < z\frac{p}{g}$$

Since $\alpha g^r \bmod p = \alpha g^r - pt$ we can rewrite the above inequality as:

$$\frac{p(z-1) + ptg}{g^{r+1}} < \alpha < \frac{pz + ptg}{g^{r+1}}$$

We now take these lower and upper bounds to be the L and U used in the next iteration. Observe that $U - L = p/g^{r+1}$. This shows that the gap between U and L decreased as expected completing the proof of the theorem. □

Theorem 4.3.1 shows that the hidden number problem can be solved using $k = \log g$ most significant bits when the generator g is used. For small values of g this improves on the result of the previous section. Notice that unlike the algorithm of the previous section, the algorithm above relies on the ability to query the oracle at chosen inputs.

We give an alternate proof of a (slightly weaker) version of this theorem in Section 4.4.2. The proof there relies on lattice rounding techniques and uses the standard $\mathrm{MSB}_k(x)$ oracle rather than the $SB_g(x)$ oracle used here. The proof in Section 4.4.2 is more powerful and is hence more likely to generalize to arbitrary g.

4.3.1 A variant of Diffie-Hellman and its bit security

Theorem 4.3.1 suggests a new variant of the Diffie-Hellman protocol. This new variant is motivated by the following corollary which states that the most significant bit of $\mathrm{DH}_g(g^x, 2)$ is as hard to compute as all of $\mathrm{DH}_g(g^x, 2)$. For clarity we observe that

$$\mathrm{DH}_g(g^x, 2) = \mathrm{DH}_g(g^x, g^{\log_g 2}) = g^{x \log_g 2} = 2^x \pmod{p}$$

Theorem 4.3.2 *Given an efficient algorithm A to compute $\mathrm{MSB}_1(\mathrm{DH}_g(g^x, 2))$ from g, g^x there is an algorithm to efficiently (in polynomial time) compute $\mathrm{DH}_g(g^x, 2)$ itself.*

Proof Set $\alpha = \mathrm{DH}_g(g^x, 2) = 2^x$. We have already seen (Theorem 4.2.2) that the algorithm A can be used to define a hidden number oracle

$$\mathcal{O}(r) = \mathrm{MSB}_1(\mathrm{DH}_g(g^{x+r}, 2)) = \mathrm{MSB}_1(\alpha \cdot 2^r \bmod p)$$

Hence, by Theorem 4.3.1 α can be found in polynomial time. □

The corollary suggests a Diffie-Hellman protocol where Alice always sends the value 2 to Bob. The idea is for Alice to pick a random generator g for which she knows a value x satisfying $g^x = 2 \bmod p$. More precisely suppose Alice and Bob have already agreed on a prime p. To perform secret key exchange they use the following protocol which we call the **Modified Diffie-Hellman Protocol:**

1. Alice picks a random number x in the range $[1, p-1]$ with $\gcd(x, p-1) = 1$. She computes $g = 2^x \pmod{p}$ and sends g to Bob.

2. Bob picks a random number y in the range $[1, p-1]$ and sends g^y to Alice.

The secret they both agree on is $\alpha = \mathrm{DH}_g(g^y, 2) = 2^y \pmod{p}$. Clearly Bob can compute this value. Alice can compute this value since

$$2^y = g^{yx^{-1}} \pmod{p}$$

where x^{-1} is the inverse of x modulo $p-1$.

An adversary who wishes to discover the secret shared by Alice and Bob observes g, g^y and must compute the value $\mathrm{DH}_g(g^y, 2)$. Corollary 4.3.2 shows that computing the MSB of the secret shared by Alice and Bob is as hard as computing the entire secret. Thus this new scheme has the advantage of provable bit security. We summarize this in the following theorem:

Theorem 4.3.3 *Given an oracle that computes the most significant bit of the modified Diffie-Hellman secret there exists a polynomial time algorithm that computes the entire secret.*

A drawback of this new variant is that it relies on the hardness of $DH_g(g^y, 2)$ for its security. This is a special case of the Diffie-Hellman function which could potentially be easier to break. The standard heuristic way of arguing about the security of the Diffie-Hellman protocol is to argue that the corresponding discrete log problem is hard. In our case the corresponding discrete log problem is that of computing discrete log of 2 base g. One can easily show that computing discrete log of 2 base g is as hard as computing discrete log of any x base g (observe that $\log_g x = \log_g 2 / \log_x 2$). Thus, the standard heuristic discrete log argument supports the security of this variant.

4.4 A nonuniform result

We show that there exist a polynomial number of advice bits depending only on p and g which enable one to solve the hidden number problem in polynomial time using an oracle which returns only $2 \log \log p$ bits. This improves the $3\sqrt{\log p}$ bound of Theorem 4.2.1.

Theorem 4.4.1 *Let p be a prime and g a generator of $\mathbb{Z}_p^*$. Set $m = \lceil 2 \log \log p \rceil$. For a hidden α define the function $\mathcal{O}(r) = \mathrm{MSB}_m(\alpha \cdot g^r \bmod p)$. Then there exists a polynomial time algorithm which takes a polynomial number of advice bits depending only on p, g and recovers α given an oracle for the function $\mathcal{O}(r)$.*

The proof relies on a new analysis of a lattice rounding technique to solve the hidden number problem. We intend to query the oracle $\mathcal{O}$ at uniformly and independently chosen inputs $r_1, \ldots, r_d$. The value of d will be specified later. Set $t_i = g^{r_i} \bmod p$ then the oracle $\mathcal{O}$ will output integers $a_1, \ldots, a_d, a_{d+1}$ such that

$$|(\alpha t_i \bmod p) - a_r| < R \qquad \text{and} \qquad |(\alpha \bmod p) - a_{d+1}| < R$$

To keep the discussion general we do not specify the value of R for now. Note that a_{d+1} is found by querying the oracle at $r = 0$. Our goal is to recover α from this information.

Let L be the lattice spanned by the rows of the matrix:

$$L = \begin{pmatrix} p & 0 & \ldots & 0 & 0 \\ 0 & p & \ldots & 0 & 0 \\ & \vdots & & & \vdots \\ 0 & 0 & \ldots & p & 0 \\ t_0 & t_1 & \ldots & t_d & 1 \end{pmatrix}$$

and set $u = (a_1, \ldots, a_d, a_{d+1})$. Define the lattice vector $v_\alpha \in L$ by

$$v_\alpha = (t_1\alpha \bmod p,\ t_2\alpha \bmod p, \ldots t_d\alpha \bmod p,\ \alpha)$$

Given u our algorithm will output the vector v_α. Clearly α can be recovered from the vector v_α. To construct the vector v_α from u we intend to use a different basis of the lattice L. Let $b_1, \ldots, b_{d+1}$ be some basis of L. Given u the algorithm performs two steps.

1. Write $u = \sum_{i=1}^{d+1} y_i b_i$ for some $y_i \in \mathbb{R}$.
2. Set $v = \sum_{i=1}^{d+1} \lfloor y_i \rceil b_i$ where $\lfloor y_i \rceil$ is the integer closest to y_i.

The question is when does this type of rounding produce the right answer, i.e. when does $v = v_\alpha$? The next lemma gives a tight criteria for when this occurs. We need the following notation: for a matrix W whose columns are the vectors $w_1, \ldots, w_d$ define

$$L_{1,\infty}(W) \stackrel{def}{=} \max_i L_1(w_i)$$

where $L_1(w_i)$ is the L_1 norm of w_i.

Lemma 4.4.2 *Let A be the matrix whose rows are the basis vectors b_i. If $L_{1,\infty}(A^{-1}) < 1/(2R)$ then the vector v constructed above satisfies $v = v_\alpha$.*

Proof Since the b_i form a basis of L we know that there exists a vector $x \in \mathbb{Z}^{d+1}$ such that $xA = v_\alpha$. Let $y \in \mathbb{R}^{d+1}$ be the vector such that $yA = u$. We need to prove that the theorem's hypothesis implies $\lfloor y_i \rceil = x_i$ for every coordinate i. Here x_i, y_i are the i'th coordinates of the vectors x, y respectively. Since x_i is an integer it suffices to prove that $|x_i - y_i| < 1/2$ for all i.

Set $W = A^{-1}$ and $W = (w_{i,j})$. Then

$$\left\{ \begin{array}{l} xA = v_\alpha \\ yA = u \end{array} \right. \implies (x-y)A = v_\alpha - u \implies x - y = (v_\alpha - u)A^{-1} \implies$$

$$|x_i - y_i| \leq \sum_{j=1}^{d+1} |w_{j,i}| \cdot |(v_\alpha)_j - u_j| \leq R \sum_{j=1}^{d+1} |w_{j,i}| \leq R \cdot L_{1,\infty}(W) < \frac{1}{2}$$

This completes the proof of the lemma. □

At this point we need to recall the notion of the dual lattice. Let L be a lattice spanned by the vectors $b_1, \ldots, b_d \in \mathbb{R}^d$. The dual lattice of L, denoted L^*, is the set of vectors

$$L^* = \left\{ w \in \mathbb{R}^d \quad \text{s.t.} \quad \forall u \in L \ : \ (u, w) \in \mathbb{Z} \right\}$$

One can readily show that if A is the matrix whose rows are $b_1, \ldots, b_d$ then the dual lattice is spanned by the columns of A^{-1}. Using this terminology Lemma 4.4.2 implies that we must find a basis of the dual lattice whose $L_{1,\infty}$ norm is as small as possible.

By inverting (and transposing) the matrix L we obtain that the dual lattice of our lattice L is spanned by the rows of the matrix:

$$L^* = \frac{1}{p} \begin{pmatrix} 1 & 0 & \ldots & 0 & -t_1 \\ 0 & 1 & \ldots & 0 & -t_2 \\ & \vdots & & & \vdots \\ 0 & 0 & \ldots & 1 & -t_d \\ 0 & 0 & \ldots & 0 & p \end{pmatrix}$$

We prove that with high probability (over the choice of the random t_i) the lattice L^* has a basis with low $L_{1,\infty}$ norm.

Theorem 4.4.3 *Let p be a prime and $d > 4 + \log p + \log\log p$. Let $t_1, \ldots, t_d$ be integers chosen uniformly and independently at random in the range $[0, p-1]$. Consider the corresponding lattice L^*. Then with probability $1/2$ there exists a basis of this lattice with $L_{1,\infty} < 3d\log p/p$.*

The proof relies on the following lemma.

Lemma 4.4.4 *Let $p, d, t_1, \ldots, t_d$ as above. Set $m = \lceil \log\log p \rceil + 4$ and $n = m + \lfloor \log p \rfloor$. By assumption $d > n$. Then with probability $1/2$ for all $k = m+1, m+2, \ldots, n$ there exist subsets $S_k \subseteq \{1, \ldots, k-1\}$ satisfying:*

$$2^{n-k} \leq t_k + \sum_{i \in S_k} t_i \bmod p < 2 \cdot 2^{n-k}$$

Proof Let $k \in [m, n]$ be an integer. For each non empty set $S \subseteq \{1, \ldots, k-1\}$ define the random variable $T_S = t_k + \sum_{i \in S} t_i \bmod p$. The set $\{T_S\}_{S \in \subseteq \{1,\ldots,k-1\}}$ is a collection of pair wise independent random variables each uniformly distributed in the range $[0, p-1]$. Similarly, define the indicator random variable X_S as

$$X_S = \begin{cases} 1 & \text{if} \quad 2^{n-k} \leq T_S < 2 \cdot 2^{n-k} \\ 0 & \text{otherwise} \end{cases}$$

For notational convenience we define $X_\emptyset$ to be the fixed value $1/2^{k-m+1}$. The set $\{X_S\}$ is also a collection of pair wise independent random variables with $E[X_S] = 2^{n-k}/p \geq 1/2^{k-m+1}$. We set $Z = \sum_S X_S$. Then by pair wise independence we obtain

$$E[Z] \geq 2^{k-1} \cdot \frac{1}{2^{k-m+1}} = 2^{m-2} \quad \text{and} \quad \mathrm{Var}[Z] = \sum_S \mathrm{Var}[X_S] < 2^{m-1}$$

Chebyshev's inequality now gives:

$$\Pr[Z < 1] \leq \Pr\left[|Z - E[Z]| > 2^{(m-3)/2}\sqrt{\mathrm{Var}(Z)}\right] < \frac{1}{2^{m-3}}$$

Hence, $Z \geq 1$ with probability at least $1 - \frac{1}{2^{m-3}}$. When $Z \geq 1$ there exists at least one set S_k satisfying the required condition. Therefore, the required S_k exists with probability at least $1 - \frac{1}{2^{m-3}}$. We conclude that the probability that S_k exists for all $k = m+1, m+2, \ldots, n$ satisfies

$$\Pr[\forall k \ : \ S_k \text{ exists}] \geq 1 - \frac{n-m}{2^{m-3}} > \frac{1}{2}$$

This concludes the proof of the lemma. □

Proof of Theorem 4.4.3 The proof follows by multiplying the matrix L^* above on the left by two unimodular matrices. Let n, m be defined as in Lemma 4.4.4. Since $d > n$ it follows that with probability at least $1/2$ there exist sets $S_{m+1}, S_{m+2}, \ldots, S_n$ satisfying the conditions of Lemma 4.4.4. For all $k \leq m$ and $k > n$ we define $S_k = \emptyset$. Define the $d+1 \times d+1$ matrix $U = (u_{i,j})$ by

$$u_{i,j} = \begin{cases} 1 & \text{if} \quad j \in S_i \ \text{ or } \ i = j \\ 0 & \text{otherwise} \end{cases}$$

All the elements on the diagonal of U are 1. Furthermore, since the sets S_k satisfy $S_k \subseteq \{1, \ldots, k-1\}$ for all k it follows that U is a lower diagonal matrix and hence unimodular. The rows of the matrix UL^* form a basis of the dual lattice L^*. Multiplying L^* by U on the left has the effect of replacing each basis vector b_k by the vector $b_k + \sum_{i \in S_k} b_i$.

After rearranging the rows of UL^* and reducing the elements in the last column modulo p the resulting basis has the following structure:

$$\frac{1}{p} \begin{pmatrix} & & u_1 \\ & & \vdots \\ & & u_{n-m} \\ & B & t_1 \\ & & \vdots \\ & & t_m \\ & & p \end{pmatrix}$$

where B is a $d+1 \times d$ matrix with $0,1$ entries. By the choice of the sets S_k we can arrange things so that the elements $u_1, \ldots, u_{n-m}$ satisfy $2^{k-1} \leq u_k < 2^k$. This property of the u_k enables one to write any integer x in the range $[1, 2^k]$ as $x = \sum_{i=1}^{k-1} x_i u_i$ where all $x_i = 0, 1, 2, 3$. Specifically, any integer x in the range $[0, p]$ can be written as $x = \sum_{i=1}^{n-m} x_i u_i$ where $x_i = 0, 1, 2, 3$.

We now transform the above basis into a basis with low $L_{1,\infty}$ norm. Let $b_1, \ldots, b_{d+1}$ be the basis vectors. Let $x^{(k)}$ be the right most entry in the vector b_k, e.g. $x^{(1)} = u_1$, $x^{(2)} = u_2$, $x^{(d+1)} = p$, etc. We replace the basis vector b_k by the vector $b_k - \sum_{i=1}^{k-1} x_i^{(k)} b_i$ where the $x_i^{(k)}$ satisfy $x^{(k)} = \sum_{i=1}^{k-1} x_i^{(k)} x^{(i)}$ with $x_i^{(k)} = 0, 1, 2, 3$. The unimodular matrix which performs this transformation is:

$$V = \begin{pmatrix} 1 & 0 & 0 & 0 & \ldots & 0 \\ -x_1^{(2)} & 1 & 0 & 0 & \ldots & 0 \\ -x_1^{(3)} & -x_2^{(3)} & 1 & 0 & \ldots & 0 \\ & \vdots & & & \vdots & \\ -x_1^{(d+1)} & -x_2^{(d+1)} & -x_3^{(d+1)} & & \ldots & 1 \end{pmatrix}$$

The resulting basis has the structure

$$\frac{1}{p} \begin{pmatrix} & & 1 \\ & & 0 \\ & B' & \vdots \\ & & 0 \end{pmatrix}$$

where B' is a $d+1 \times d$ matrix whose entries are all less than $3(n-m) < 3 \log p$ in absolute value. Thus, the $L_{1,\infty}$ norm of the resulting basis is at most $3d \log p/p$. This completes the proof of Theorem 4.4.3. □

Proof of Theorem 4.4.1 The non-uniform result of Theorem 4.4.1 readily follows by combining Lemma 4.4.2 and Theorem 4.4.3. Given a prime p and a generator g the advice bits is a set of integers $t_1, \ldots, t_d$ with $d = 2 \log p$ along with a basis of L^* whose $L_{1,\infty}$ norm is less than $3d \log p/p = 6 \log^2 p/p$. Theorem 4.4.3 shows that indeed there

exist integers $t_1, \ldots, t_d$ for which the required basis exists[1]. Furthermore, the advice bits include the discrete-log of $t_1, \ldots, t_d$ to the base g.

Since the $L_{1,\infty}$ norm of the given basis of L^* is less than $6\log^2 p/p$, Lemma 4.4.2 shows that the rounding algorithm will find the correct α if the oracle returns $2\log\log p + O(1)$ bits. More precisely, the value of R in Lemma 4.4.2 must satisfy $R < p/12\log^2 p$ implying that the oracle must return at least $2\log\log p + 4$ bits. Notice that since the discrete-log of t_i base g is known, the oracle can indeed be queried at the appropriate points to return the most significant bits of $\alpha \cdot t_i \bmod p$. This completes the proof of Theorem 4.4.1. □

We note that constructing a basis satisfying the conditions of Lemma 4.4.3 is likely to be hard. It requires one to solve certain instances of the modular subset sum problem. Impagliazzo and Naor [IN89] note that these instances are likely to be hard on average.

4.4.1 An application to the Okamoto conference scheme

Unfortunately the non-uniform result of Theorem 4.4.1 can not be applied to the Diffie-Hellman protocol. However it can be applied to the Okamoto conference key sharing protocol described in Section 4.2. As can easily be seen, this is a simple variation on the Diffie-Hellman protocol. We obtain the following theorem which is much better than Theorem 4.2.4(page 64) in terms of the number of bits needed.

Theorem 4.4.5 *There exist advice bits which depend only on the prime p and the generator g such that given an efficient algorithm to compute $2\log\log p$ MSB's of the conference key in the Okamoto system there is an algorithm to efficiently (in expected polynomial time) compute the entire key.*

Proof The proof is identical to the proof of Theorem 4.2.4 using Theorem 4.4.1 rather than Theorem 4.2.1. □

[1]Theorem 4.4.3 shows even more; it shows that for most tuples $(t_1, \ldots, t_d)$ such a basis is known to exist.

The theorem can not be applied to the Diffie-Hellman protocol or the other related schemes since the generator g in the associated hidden number problem is not fixed. For instance, the relation $\mathrm{DH}_g(g^{x+r}, g^y) = g^{xy} \cdot (g^y)^r$ shows that the generator $h = g^y$ in the associated hidden number problem is changed in every invocation of the protocol. For the Okamoto scheme the relation $OK_g(g^{y(x+r)}, g^y) = g^x \cdot g^r$ shows that the generator g in the associated hidden number problem is independent of the invocation.

4.4.2 An alternate proof of the small generator case

Theorem 4.3.1 is very useful when the generator g is small. Unfortunately the proof does not seem to generalize to the case of arbitrary generator. In this section we give a more powerful proof of a theorem slightly weaker than Theorem 4.3.1. This proof is likely to generalize to the case of arbitrary generator though at the present this is still an open problem.

Theorem 4.4.6 *For fixed p and $g, \alpha \in \mathbb{Z}_p^*$ define $\mathcal{O}(x)$ to be the function $\mathcal{O}(x) = \mathrm{MSB}_k(\alpha \cdot g^x \bmod p)$ where $k = \log g + \log\log p + 2$. There exists a polynomial time algorithm (in $\log p$) which given access to an oracle computing the function $\mathcal{O}(x)$ finds α.*

Proof Let L^* be the lattice spanned by the rows of the matrix:

$$A = \frac{1}{p} \begin{pmatrix} 1 & 0 & \dots & 0 & -g \\ 0 & 1 & \dots & 0 & -g^2 \\ & \vdots & & & \vdots \\ 0 & 0 & \dots & 1 & -g^d \\ 0 & 0 & \dots & 0 & p \end{pmatrix}$$

where $d > \log_g p$. Lemma 4.4.2 shows that to prove Theorem 4.4.6 it suffices to construct a basis of L^* with low $L_{1,\infty}$ norm. Write $p = \sum_{i=0}^{d} b_i g^i$ where all b_i satisfy $0 \le b_i < g$. This is possible since $d > \log_g p$. We multiply the basis A by a unimodular

matrix to obtain a new basis B as follows:

$$B = \begin{pmatrix} 1 & 0 & 0 & \dots & 0 & 0 & 0 \\ -g & 1 & 0 & \dots & 0 & 0 & 0 \\ 0 & -g & 1 & \dots & 0 & 0 & 0 \\ & \vdots & & & \vdots & & \\ 0 & 0 & 0 & \dots & -g & 1 & 0 \\ b_1 & b_2 & b_3 & \dots & b_{d-1} & b_d & 1 \end{pmatrix} \cdot A = \frac{1}{p} \begin{pmatrix} 1 & 0 & 0 & \dots & 0 & 0 & -g \\ -g & 1 & 0 & \dots & 0 & 0 & 0 \\ 0 & -g & 1 & \dots & 0 & 0 & 0 \\ & \vdots & & & \vdots & & \\ 0 & 0 & 0 & \dots & -g & 1 & 0 \\ b_1 & b_2 & b_3 & \dots & b_{d-1} & b_d & b_0 \end{pmatrix}$$

For this basis we have $L_{1,\infty}(B) = gd/p$. Therefore, taking $R = \frac{p}{3gd}$ in Lemma 4.4.2 guarantees that the rounding algorithm of Section 4.4 will recover α. Taking $R = \frac{p}{3dg}$ corresponds to an oracle which returns more than $\log 3dg$ of the most significant bits of αg^k. Since $\log 3dg \leq \log g + \log\log p + 2$ we see that the hidden number problem can be solved using this many bits. □

For very small g this is not as strong as Theorem 4.3.1 (due to the $\log\log p$ term). However, the above algorithm uses the general MSB oracle rather than the special SB oracle used in the algorithm of Theorem 4.3.1.

We are hopeful that the above proof can be generalized to show that a small number of bits suffices to solve the hidden number problem for an arbitrary generator g. To do so one must construct a basis of L^* with low $L_{1,\infty}$ norm for a general g.

4.5 Conclusions

Currently there are no results regarding the security of individual bits of the Diffie-Hellman shared secret. The results of this chapter make progress in this direction. We proved that $3\sqrt{\log p}$ of the most significant bits of the Diffie-Hellman secret as hard to compute as the entire secret. Further we defined a new variant of the Diffie-Hellman protocol for which a single bit is as hard to compute as the entire secret.

These results are proved by studying an abstract problem which we call the hidden number problem. Our results regarding the hidden number problem are general enough as to apply to many other schemes which are based on the Diffie-Hellman

protocol. We proved that $3\sqrt{\log p}$ of the most significant bits of a message encrypted using the ElGamal public key system or Shamir's three pass protocol are as hard to compute as the entire message. The same result holds for the secret key of the Okamoto conference scheme.

Finally in Section 4.4 we described a new lattice rounding technique. This technique was used to prove that $2 \log \log p$ bits are sufficient for solving the hidden number problem in a non-uniform model. Applying this result to the Okamoto conference scheme proved that in a non-uniform model $2 \log \log p$ bits of the conference key are as hard to compute as the entire key.

We note that the value $k = 3\sqrt{\log p}$ used in the theorems throughout the chapter can be asymptotically improved to $\epsilon\sqrt{\log p}$ for any fixed $\epsilon > 0$. This is done by combining results of Schnorr [Sch87] with Babai's closest lattice point algorithm. Schnorr's improvement to the LLL algorithm improves Babai's algorithm which in turn reduces the number of bits required in Theorem 4.2.1 to $\epsilon\sqrt{\log p}$. Consequently the algorithm of Theorem 4.2.1 becomes less efficient, but is still polynomial time for fixed ϵ.

There is still substantial progress to be made regarding Diffie-Hellman bit security and the hidden number problem. For one thing it would be nice to have an algorithm for solving the hidden number problem for an arbitrary generator using only the most significant bit (rather than $O(\sqrt{\log p})$ bits). The next step is to solve the problem given a $\frac{1}{2} + \epsilon$ noisy oracle. Such a result would completely solve the bit security of the Diffie-Hellman secret.

Chapter 5

A Group Theoretic Goldreich-Levin Theorem

Let $\mathcal{W}_n$ denote the linear space of function $f : \{0,1\}^n \to \mathbb{C}$ over $\mathbb{C}$. Two functions f, g in $\mathcal{W}_n$ can be added and multiplied by a complex number. Since a function $f : \{0,1\}^n \to \mathbb{C}$ can be regarded as a vector of dimension 2^n the space $\mathcal{W}_n$ is a linear space of dimension 2^n. There is a natural inner product defined on $\mathcal{W}_n$ by:

$$(f, g) = \frac{1}{2^n} \sum_{x \in \{0,1\}^n} f(x) \cdot \overline{g(x)}$$

where $\overline{z}$ denotes the complex conjugate of z. An *orthonormal basis* of $\mathcal{W}_n$ with respect to this inner product is a basis $\mathcal{B} = \{b_1, \ldots, b_{2^n}\}$ such that all $k \neq l$ satisfy $(b_k, b_l) = 0$ and $(b_k, b_k) = 1$. Any function f in $\mathcal{W}_n$ can be represented as

$$f(x) = \sum_{k=1}^{2^n} \hat{f}_k \cdot b_k(x)$$

where $\hat{f}_k$ are complex numbers.

In this chapter we study the *large coefficient problem*: given a basis $\mathcal{B}$ and a function f in $\mathcal{W}_n$ find the large coefficients of f on $\mathcal{B}$. In other words, given δ find all k such that $|\hat{f}_k| > \delta$. Our goal is to construct algorithms for this problem whose running time is polynomial in n and $\frac{1}{\delta}$. Since there are 2^n basis elements this is clearly not doable for an arbitrary basis $\mathcal{B}$. The main result of this chapter is a large

class of orthonormal bases of $\mathcal{W}_n$ for which the largest coefficient problem can be solved in polynomial time (in n/δ). These orthonormal bases are the ones arising from representations of finite groups.

The large coefficient problem is closely related to a theorem due to Goldreich and Levin [GL89]. As a result, our solution to the large coefficient problem can be used to generalize the Goldreich-Levin theorem. This generalization has applications in learning theory. For instance, we show that any group-decision-tree can be learned in polynomial time in the size of the tree. These concepts will be defined in Section 5.5.

5.1 Hard core bits

Pseudo random generators are at the hart of many cryptographic protocols. A central result is the Goldreich-Levin theorem which provides the tools for constructing pseudo random number generators from any one-way function. The key concept used in this construction is that of a *hard core* bit. We first define these concepts and then state the Goldreich-Levin theorem. Next we discuss our generalization of the theorem.

A polynomial time function f is called one-way if any probabilistic polynomial time algorithm can invert it with only a negligible probability. A polynomial time predicate b is called a *hard-core* of a function f if all expected polynomial time algorithms, given $f(x)$, can guess $b(x)$ with success probability which is only negligibly better than a half. More precisely:

Definition 5.1.1 *A polynomial time computable predicate $b : \{0,1\}^* \rightarrow \{0,1\}$ is called a* hard-core *of a function f if for every probabilistic polynomial time algorithm A*

$$\Pr[A(f(x)) = b(x)] < \frac{1}{2} + \frac{1}{n^c}$$

for all constants c and sufficiently large n. Here x is chosen uniformly and independently from $\{0,1\}^n$.

The first example of a hard-core bit was given by Blum and Micali [BM84] for the discrete log function. They showed that if discrete log modulo p can not be

computed in polynomial time then $b(x) = \mathrm{MSB}(x)$ is a hard-core predicate for the function $f(x) = g^x \pmod p$. To prove such a statement Blum and Micali show that discrete log can be computed in expected polynomial time given an oracle A satisfying $A(g^x \bmod p) = \mathrm{MSB}(x)$ for at least $\frac{1}{2} + \frac{1}{n^c}$ of the inputs. It follows that if $\mathrm{MSB}(x)$ was not a hard-core of $f(x) = g^x \pmod p$ then discrete log would be computable in expected polynomial time. We note that the least significant bit $\mathrm{LSB}(x)$ is also a hard core predicate of $f(x) = g^x \pmod p$ when the generator g is of prime order[1]

Hard core predicates are used to construct pseudo random number generators. We do not describe this construction (nor the definition of a pseudo random number generator) here. We refer to [Gol95] for the details.

The Goldreich-Levin theorem constructs a hard-core predicate for any one way permutation. Consequently, a pseudo random number generator can be constructed out of any one way permutation. Having stressed the importance of this theorem we can now finally state it. We define the inner product of two binary vectors u, v in $\{0,1\}^n$ by

$$< u, v >= (-1)^{\sum_{j=1}^{n} u_j \cdot v_j}$$

Theorem 5.1.1 (Goldreich-Levin) *Let $f : \{0,1\}^n \to \{0,1\}^n$ be a one way permutation. Define a new function $g(x,r) = (f(x), r)$ where $x, r \in \{0,1\}^n$. Then the inner product $b(x,r) =< x, r >$ is a hard-core predicate of g.*

This is the standard formulation of the theorem. Unfortunately this formulation hides the algorithmic aspect of the theorem. We therefore reword the theorem as follows:

Theorem 5.1.2 (Goldreich-Levin) *Let $f : \{0,1\}^n \to \{0,1\}^n$ be an arbitrary function. Given an oracle A satisfying*

$$\Pr_{x,r}[A(f(x), r) =< x, r >] > \frac{1}{2} + \frac{1}{n^c}$$

[1]When g has even order, the least significant bit of x can be easily deduced from g^x by computing the Legendre symbol of g^x. Therefore $\mathrm{LSB}(x)$ is not hard-core in this case.

there exists an expected polynomial time algorithm A' such that

$$\Pr_x[A'(f(x)) = x] > \frac{1}{n^d}$$

for some constant d depending only on c.

Let S be the set of $x \in \{0,1\}^n$ for which the above oracle A satisfies:

$$\Pr_r[A(g(x), r) =< x, r >] > \frac{1}{2} + \frac{1}{2n^c}$$

Here x is fixed and r is chosen uniformly at random in $\{0,1\}^n$. In the statement of the theorem both x and r were chosen uniformly and independently at random. By a standard averaging argument one can show that $|S| > \frac{2^n}{2n^c}$. Otherwise, the oracle A can not satisfy the assumption of Theorem 5.1.2. At the hart of the Goldreich-Levin theorem is the statement that the oracle A can be used to invert f on any input $x \in S$. We therefore obtain the final formulation of the Goldreich-Levin theorem which completely exposes its algorithmic nature.

Theorem 5.1.3 (Goldreich-Levin) *Let x be an element in $\{0,1\}^n$. Given an oracle A_1 satisfying*

$$\Pr_r\left[A_1(r) =< x, r >\right] > \frac{1}{2} + \frac{1}{n^c}$$

one can find x in expected polynomial time (in n).

The theorem shows that for a fixed x in S the oracle $A_1(r) = A(f(x), r)$ can be used to recover x. In other words, the oracle A enables one to invert the function f for any $x \in S$. Hence, Theorem 5.1.3 implies Theorem 5.1.2.

To explain the connection between the *large coefficient problem* and Theorem 5.1.3 we look at the theorem under a slightly different light. Given a binary vector x in $\{0,1\}^n$ the character associated with x is the parity function defined on $\{0,1\}^n$ by

$$\chi_x(r) =< x, r >= (-1)^{\sum_{j=1}^n x_i \cdot r_i}$$

It is well known that the set of functions $\{\chi_x\}_{x \in \{0,1\}^n}$ is an orthonormal basis of the linear space $\mathcal{W}_n$ where $\mathcal{W}_n$ is the space of all functions $f : \{0,1\}^n \to \mathbb{C}$. We refer

to this orthonormal basis as the *parity basis*. Assuming the oracle A_1 only obtains values in $1, -1$ we have the relation:

$$(2) \qquad \left|(A_1, \chi_x)\right| = \frac{1}{2^n}\left|\sum_r A_1(r)\cdot\chi_x(r)\right| = \left|1 - 2\Pr_r[A_1(r) = \chi_x(r)]\right| > \frac{2}{n^c}$$

In other words, the function $A_1 : \{0,1\}^n \to \{1,-1\}$ has a non-negligible projection on one of the basis elements in the parity basis. Finding this basis element is equivalent to finding x.

Viewed in this light, the Goldreich-Levin theorem simply states that the large coefficient problem can be solved when the orthonormal basis used is the parity basis. Therefore, it makes sense to try and construct other basis of $\mathcal{W}_n$ for which the large coefficient problem can be solved in expected polynomial time. Such bases will generalize the Goldreich-Levin theorem and provide new constructions for hard-core predicates.

5.2 Learning using orthonormal bases

The algorithm provided by the Goldreich-Levin theorem has many important applications in learning theory. This was first observed in an influential paper by Kushilevitz and Mansour [KM91]. They showed how the ability to find the large coefficients of a function on the parity basis can be used to learn the function. This approach became known as the Fourier method for learning and is a valuable tool in learning theory[LMN89, KM91, Man92, Bel92, BFJ*94, Jac94]. Its most powerful application is due to Jackson [Jac94] who solved the long standing open problem of learning polynomial size DNF formula in polynomial time using membership queries.

These applications to learning theory motivate the study of the *large coefficient problem* for bases other than the parity basis. As we shall see, any orthonormal basis of $\mathcal{W}_n$ for which the large coefficient problem is solvable, gives rise to a learning algorithm for a certain class of functions. It is therefore not surprising that our results give rise to new classes of functions which can be learned in polynomial time. We begin by defining the learning model.

Definition 5.2.1 *Let $\mathcal{F}$ be a a family of boolean functions over some domain G. We say that $\mathcal{F}$ is* learnable *if there exists an algorithm A which for every $f \in \mathcal{F}$ and $\epsilon, \delta > 0$ outputs an hypothesis $h = A(f, \epsilon, \delta)$ such that*

$$\Pr[error(f,h) \geq \epsilon] \leq \delta$$

where $error(f,h) = \Pr[f(x) \neq h(x)]$ and $\Pr$ is the uniform distribution on G. The algorithm is polynomial if its running time is polynomial in $n, \frac{1}{\epsilon}, \log \frac{1}{\delta}$ where $n = \log |G|$.

Next we explain how an arbitrary orthonormal basis can be used to learn a family of functions. The following definitions extend the ones in [KM91] to handle arbitrary orthonormal bases. Let G be some finite domain of size s. Usually G will be taken to be $\{0,1\}^n$. Let $\mathcal{W}$ be the vector space of all functions $f : G \to \mathbb{C}$. We have already seen that there is a natural inner product defined on $\mathcal{W}$, i.e.

$$(f,g) = \frac{1}{s} \sum_{x \in G} f(x)\overline{g(x)}$$

Let $\mathcal{B} = \{b_1, \ldots, b_s\}$ be an orthonormal basis of $\mathcal{W}$. Any function $f \in \mathcal{W}$ can be uniquely written as

$$f = \sum_{k=1}^{s} c_k b_k \quad \text{where} \quad c_k = (f, b_k).$$

Parseval's identity states that $\| f \|^2 = \sum |c_k|^2$. Define the L_1-norm of f with respect to $\mathcal{B}$ as

$$L_{\mathcal{B}}(f) = \sum_{k=1}^{s} |c_k|$$

Lemma 5.2.1 *Let $f : G \to \{-1, 1\}$, and $t > 0$. Define $g_t(x) = \sum_{|c_i|>t} c_i b_i(x)$ and set*

$$h_t(x) = \text{sign}(\text{real}(g_t(x)))$$

Then

$$\Pr[f(x) \neq h_t(x)] \leq t L_{\mathcal{B}}(f)$$

Proof Observe that when $f(x) \neq h_t(x)$ we have that

$$|f(x) - g_t(x)| \geq |f(x) - \text{real}(g_t(x))| \geq 1$$

Hence,

$$\Pr[f(x) \neq h_t(x)] \quad \leq \quad \| f - g_t \|^2 = \sum_{|c_k| \leq t} |c_k|^2 \leq t \sum_{k=1}^{s} |c_k| = t \cdot L_{\mathcal{B}}(f)$$

□

Setting $t = \epsilon / L_{\mathcal{B}}(f)$ implies that $\Pr[f(x) \neq h_t(x)] \leq \epsilon$. Hence, $h_t(x)$ is a valid output of a learning algorithm applied to f. By Parseval's identity,

$$1 = \| f \|^2 = \sum |c_k|^2$$

Therefore, there can be at most $(L_{\mathcal{B}}(f)/\epsilon)^2$ coefficients c_k larger than $\epsilon / L_{\mathcal{B}}(f)$. When $L_{\mathcal{B}}(f)$ is polynomial in $n = \log |G|$, the hypothesis description length will be polynomial in n and $\frac{1}{\epsilon}$ and may be generated by a polynomial time algorithm.

The above discussion shows that in order to learn a set functions $\mathcal{F}$ one has to find an orthonormal basis $\mathcal{B}$ of $\mathcal{W}$ that satisfies the following two properties:

1. Every $f \in \mathcal{F}$ has a polynomial $L_{\mathcal{B}}(f)$ norm.
2. There exists a polynomial time algorithm that finds the large projections of f on the basis $\mathcal{B}$.

A large portion of the work in learning theory related to Fourier analysis concentrates on finding $\mathcal{F}$ that satisfy property 1 for the basis made up of parity functions. Here we are concerned with finding bases that satisfy property 2. This will enable us to use these methods for learning functions which are not necessarily learnable using the parity basis.

5.3 Group Representations

The orthonormal bases we use arise from representations of finite group. Therefore, we must first recall some basic definitions from the theory of linear representations

of finite groups. For an introduction to this subject see [Ser77]. For a vector space V over $\mathbb{C}$ we let $GL(V)$ denote the group of invertible linear transformations of V under composition. From here on G and H will denote finite groups.

Definition 5.3.1 *A* representation *ρ of G is a homomorphism $\rho : G \to GL(V)$ where V is a finite dimensional vector space over $\mathbb{C}$. The dimension of V is called the degree of the representation and is denoted by d_ρ.*

Definition 5.3.2 *A representation $\rho : G \to GL(V)$ is said to be* irreducible *if there is no non trivial subspace of V which is invariant under $\rho(g)$ for all $g \in G$.*

Definition 5.3.3 *Two representations $\rho : G \to GL(V)$ and $\sigma : G \to GL(W)$ are equivalent if there is an isomorphism $f : V \to W$ such that all $g \in G$ satisfy $\rho(g) = f^{-1}\sigma(g)f$.*

Equivalent representations have the same degree and their images are conjugate subgroups of $GL(V)$. Naturally, we will be interested in representations up to equivalence. A finite group admits only a finite number of nonequivalent irreducible representations.

Theorem 5.3.1 *Let $\rho_1, \ldots, \rho_c$ be all the irreducible representations (up to equivalence) of a finite group G and let $d_1, \ldots, d_c$ denote their degrees. Then*

$$\sum_{k=1}^{c} d_k^2 = |G|$$

We note that a group G is Abelian if and only if all its irreducible representations have degree 1. We now give and example of the representations of a non Abelian group.

Example The quaternion group Q consists of 8 elements $\{\pm 1, \pm i, \pm j, \pm k\}$ and the group operation satisfies the relations: $-1 = i^2 = j^2 = k^2 = ijk$. This group is not

Abelian since $ij = -ji = k$. The five irreducible representations of G are:

$$\rho_1 \equiv 1$$

$$\rho_2(\pm 1) = \rho_2(\pm i) = 1 \quad ; \quad \rho_2(\pm j) = \rho_2(\pm k) = -1$$

$$\rho_3(\pm 1) = \rho_3(\pm j) = 1 \quad ; \quad \rho_3(\pm i) = \rho_3(\pm k) = -1$$

$$\rho_4(\pm 1) = \rho_4(\pm k) = 1 \quad ; \quad \rho_4(\pm i) = \rho_4(\pm j) = -1$$

$$\rho_5(\pm 1) = \begin{pmatrix} \pm 1 & 0 \\ 0 & \pm 1 \end{pmatrix} \quad ; \quad \rho_5(i) = \begin{pmatrix} 0 & -1 \\ 1 & 0 \end{pmatrix}$$

$$\rho_5(j) = \begin{pmatrix} 0 & -i \\ -i & 0 \end{pmatrix} \quad ; \quad \rho_5(k) = \begin{pmatrix} i & 0 \\ 0 & -i \end{pmatrix}$$

$$\rho_5(-i) = -\rho_5(i) \quad ; \quad \rho_5(-j) = -\rho_5(j) \quad ; \quad \rho_5(-k) = -\rho_5(k)$$

The square of the degrees of $\rho_1, \ldots \rho_5$ sum to 8. Hence, by Theorem 5.3.1, these are all the irreducible representation of the group Q.

Theorem 5.3.2 *Let $\rho : G \to GL(V)$ be any representation of G. There exists a basis of V for which the matrix representing $\rho(g)$ is unitary[2] for all $g \in G$. Further more, when G is a p-group[3] there exists a basis of V for which the matrix representing $\rho(g)$ is monomial[4] for all $g \in G$.*

See [Dor71, p. 77] for a proof of the second part of the theorem. The matrices representing ρ_5 above are monomial. From now on, whenever we speak of the matrix of a representation we will assume that some basis satisfying the assertion of Theorem 5.3.2 has been fixed. Given a representation ρ of G, we denote by $\rho_{k,l}(g)$ the element in the k'th row and l'th column of the unitary matrix representing $\rho(g)$. The next theorem describes the orthonormal basis of $\mathcal{W}$ that will be used by the learning algorithm.

---

[2] *A matrix P is unitary if $PP^* = I$ where $P^* = \overline{P}^t$.*

[3] *A p-group is a group whose order is a power of p.*

[4] *A monomial matrix is a unitary matrix in which all the elements are either 0 or a root of unity. Groups that can be represented using monomial matrices are called $\mathcal{M}$ groups. It can be shown that any nilpotent group is an $\mathcal{M}$ group (a group is called nilpotent if it is a direct product of p-groups). Specifically, any group of size 2^n is an $\mathcal{M}$ group.*

Theorem 5.3.3 *Let ρ, ψ be two irreducible representations of G and let d be the degree of ρ. Then*

$$(\rho_{k,l}, \psi_{r,s}) \stackrel{def}{=} \frac{1}{|G|} \sum_{g \in G} \rho_{k,l}(g) \cdot \overline{\psi_{r,s}(g)} = \begin{cases} \frac{1}{d} & \text{if } \rho \equiv \psi, k = r, l = s, \\ 0 & \text{otherwise.} \end{cases}$$

We define an orthonormal basis of $\mathcal{W}$ as follows. For an irreducible representations ρ of G and k, l between 1 and the degree of ρ define the function

$$b_{\rho,k,l}(x) \stackrel{def}{=} \sqrt{d_\rho}\ \rho_{k,l}(x)$$

Let $\mathcal{B}_G \subset \mathcal{W}$ be the set of functions $\{b_{\rho,k,l}(x)\}$ where ρ ranges over all the irreducible representations of G and k, l range between 1 and the degree of ρ. Theorem 5.3.3 tells us that $\mathcal{B}_G$ is an orthonormal set in $\mathcal{W}$. By theorem 5.3.1, the number of functions in $\mathcal{B}_G$ is $|G|$. Hence, $\mathcal{B}_G$ is an orthonormal basis of $\mathcal{W}$.

Example The orthonormal basis generated by the representations of the quaternion group Q is:

$$\begin{array}{cc} [1,1,1,1,1,1,1,1] & \sqrt{2}\ [1,-1,0,0,0,0,i,-i] \\ [1,1,1,1,-1,-1,-1,-1] & \sqrt{2}\ [0,0,-1,1,-i,i,0,0] \\ [1,1,-1,-1,1,1,-1,-1] & \sqrt{2}\ [0,0,1,-1,-i,i,0,0] \\ [1,1,-1,-1,-1,-1,1,1] & \sqrt{2}\ [1,-1,0,0,0,0,-i,i] \end{array}$$

Each vector is a function $b(x) = [b(1), b(-1), b(i), b(-i), b(j), b(-j), b(k), b(-k)]$.

An irreducible representation ρ of G of degree d_ρ gives rise to d_ρ^2 orthonormal functions. Given a function $f : G \to \mathbb{C}$ it is convenient to write the projection of f on the orthonormal functions generated by ρ in a $d_\rho \times d_\rho$ matrix denoted by $\hat{f}_\rho$. Formally, the *coefficient matrix* is defined by:

$$\hat{f}_\rho = \frac{\sqrt{d_\rho}}{|G|} \sum_{x \in G} f(x) \cdot (\rho(x))^*$$

where $(\rho(x))$ is the unitary matrix representing $\rho(x)$. Using this notation, the following theorem is a simple consequence of Theorem 5.3.3.

Theorem 5.3.4 (Fourier Inversion) *Any function $f : G \to \mathbb{C}$ satisfies*

$$f(x) = \sum_{\rho} \sqrt{d_\rho} \quad \mathrm{Tr}(\hat{f}_\rho \cdot (\rho(x)))$$

where the sum is taken over all the irreducible representations of G.

For a function $f : G \to \mathbb{C}$ we denote by $L_G(f)$ the L_1-norm of f with respect to $\mathcal{B}_G$, i.e.

$$L_G(f) = \sum_{\rho} \sum_{k,l} |(\hat{f}_\rho)_{k,l}|$$

We now turn our attention to determining the irreducible representations of a direct product of two groups.

Theorem 5.3.5 *Let $T = G \times H$. Any element $t \in T$ can be uniquely written as $g_t h_t$ where $g_t \in G$ and $H_t \in H$. Let $\rho_1, \ldots, \rho_p$ be the irreducible representations of G and $\sigma_1, \ldots, \sigma_q$ be the irreducible representations of H. Then T has exactly pq irreducible representations given by*[5]

$$\tau_{k,l}(t) = \rho_k(g_t) \otimes \sigma_l(h_t) \qquad for \qquad 1 \leq k \leq p \ , \ 1 \leq l \leq q$$

The above theorem enables us to easily determine the orthonormal basis associated with groups which are a direct product of small groups. For instance, an embedding of the group Q^n(n copies of the quaternion group) in the boolean cube $\{0,1\}^{3n}$ gives rise to an orthonormal basis of the space of functions $\{0,1\}^{3n} \to \mathbb{C}$. Furthermore, the functions which make up the basis are easy to evaluate.

Let ρ be an irreducible representation of a group G. By Theorem 5.3.3 the function $\rho_{k,l}(x)$ has norm $1/d_\rho$. To normalize it, the function has to be multiplied by $\sqrt{d_\rho}$. When the dimension is large this normalization could make the function attain large values. The following simple lemma shows that when learning a boolean function, the basis functions associated with representations of large degrees can be ignored.

[5] *For $A : V \to V$ and $B : W \to W$ the tensor product $A \otimes B : V \otimes W \to V \otimes W$ is the operator sending a basis element $(v \otimes w)$ to $(Av \otimes Bw)$. Technically, if A is an $n \times n$ matrix and $B = (b_{i,j})$ is an $m \times m$ matrix, then $A \otimes B$ is an $nm \times nm$ matrix made up of $m \times m$ blocks where the (i,j)'th block is $b_{i,j}A$.*

Lemma 5.3.6 *Let G be a p-group and f a boolean function $f : G \to \{1, -1\}$. Let ρ be a representation of G. The norm of all entries of $\hat{f}_\rho$ is less than $\frac{1}{\sqrt{d_\rho}}$.*

Proof Using the fact that the matrices representing ρ are monomial and Theorem 5.3.3 we get:

$$\begin{aligned}\left|(\hat{f}_\rho)_{l,k}\right| &= \frac{\sqrt{d_\rho}}{|G|}\left|\sum_{x\in G} f(x)\cdot\overline{\rho_{k,l}(x)}\right| \leq \frac{\sqrt{d_\rho}}{|G|}\sum_{x\in G}|\rho_{k,l}(x)| = \\ &= \frac{\sqrt{d_\rho}}{|G|}\sum_{x\in G}|\rho_{k,l}(x)|^2 = \frac{\sqrt{d_\rho}}{d_\rho} = \frac{1}{\sqrt{d_\rho}}\end{aligned}$$

□

From here on whenever we speak of a representation it is assumed to be irreducible.

5.4 An algorithm for finding large Fourier coefficients

Let G be a group structure defined on $\{0,1\}^n$. Let $\mathcal{B}_G$ be the orthonormal basis of $\mathcal{W}_n$ induced by G. We present an algorithm for finding all coefficients of $f : G \to \{-1, 1\}$ on $\mathcal{B}_G$ of norm bigger than some given $\beta > 0$. The algorithm is randomized and runs in time polynomial in $n, \frac{1}{\beta}, \log\frac{1}{\delta}$ where δ is the probability of failure.

The algorithm can be made to work over any group G as long as one can evaluate the functions in $\mathcal{B}_G$ in polynomial time. For now we restrict our attention to special groups which we refer to as *product groups*. These groups are direct products of small groups. The theory needed to make the algorithm work for an arbitrary 2-group embedded in $\{0,1\}^n$ is explained in Section 5.6.

Definition 5.4.1 *A family of groups $\{G_k\}_{k\geq 1}$ is said to be a* product group family *if $G_k = H^k$ for all k and some fixed p-group H. A member of a product group family will be called a* product group.

Thus, a product group is a direct product of a number of copies of some group H. A product group G admits a normal series of subgroups

$$\{1\} = N_0 \lhd N_1 \lhd \ldots \lhd N_t = G$$

where $N_j = H^j$. Observe that for each $0 < j < t$ we have $G = G/N_j \times N_j$. We call this normal series a *product tower* of G. Note that $|N_{j+1}/N_j| = |H| = O(1)$.

The product tower of G plays an important role in the algorithm. It is worth noting that the algorithm works for any p-group G by using a principle composition series[6] of the group instead of a product tower. This requires an application of Clifford theory [Cli37] and is explained in Section 5.6.

An important property of product groups is that $\rho_{k,l}(g)$ can be evaluated in polynomial time (in t) for any representation ρ. This follows from Theorem 5.3.5 since if $G = H^t$ then

$$\rho(g) = \sigma_1(h_1) \otimes \sigma_2(h_2) \otimes \ldots \otimes \sigma_t(h_t)$$

where $\sigma_1, \ldots, \sigma_t$ are H-representations and $g = h_1 \cdots h_t$ is the unique way of writing $g \in G$ as a product of elements in H. The representations σ_k of H can be evaluated in constant time since H has a fixed size. Thus, evaluating $\rho_{k,l}(g)$ amounts to t multiplications.

A product group family $G = H^m$ whose size is always a power of two can be embedded naturally in the boolean cube $\{0,1\}^n$: assign distinct sets of variables to encode the different copies of H that comprise the group G. More generally we say that a product group G is *naturally embedded* in the boolean cube $\{0,1\}^n$ if there exists a polynomial time algorithm (in n) which given two n-bit strings g_1, g_2 outputs the string representing $g_1 + g_2$. Here '+' refers to the group action on $\{0,1\}^n$ defined by G. From here on we only consider natural embeddings in the boolean cube. We can now state the main theorem of this chapter. We refer to this theorem as the generalized Goldreich-Levin theorem.

[6] A principle composition series is a series of subgroups $N_0 < N_1 < \ldots < N_k = G$ where each N_i is normal in G and N_{i+1}/N_i is simple. Every p-group, i.e. group whose order is a power of p, has a principle composition series.

Theorem 5.4.1 (Generalized Goldreich-Levin)
Let $\{G_k\}_{k\geq 1}$ be a product group family naturally embedded in the boolean cube. Let $f_k : G_k \to \{1,-1\}$ be a function. There exists a randomized algorithm whose running time is polynomial in $\log |G_k|, \frac{1}{\beta}, \log \frac{1}{\delta}$ which given an oracle for f outputs, with probability at least $1-\delta$, all representations ρ and indices k,l such that $\left|(\hat{f}_\rho)_{k,l}\right| > \beta$.

Given an oracle for a function $f : G \to \{-1,1\}$ the algorithm works in two steps:

1. Find all representations ρ of G such that

$$\| \hat{f}_\rho \|^2 = \mathrm{Tr}(\hat{f}_\rho \hat{f}_\rho^*) = \sum_{k,l} \left|(\hat{f}_\rho)_{k,l}\right|^2 > \beta^2$$

2. For each representation ρ generated in step 2, find all the elements of the matrix $\hat{f}_\rho$ whose norm is bigger than β.

We begin the detailed description of the algorithms by first explaining how to carry out step 2. Given a representation ρ we wish to find all the entries (k,l) of the matrix $\hat{f}_\rho$ such that $|(\hat{f}_\rho)_{k,l}| > \beta$. By Lemma 5.3.6 the representations of G of degree greater than $(1/\beta)^2$ need not be considered. Hence, step 2 of the algorithm may be carried out by testing the norm of $(\hat{f}_\rho)_{k,l}$ for all $1 \leq k,l \leq d_\rho$. To approximate the value of

$$(\hat{f}_\rho)_{k,l} = \mathrm{E}\left[\sqrt{d_\rho}\ f(x)\ \overline{\rho_{l,k}(x)}\right]$$

for a given k,l we use random sampling of $x \in G$. This is explained in more detail in the next section.

When the group G is Abelian all the irreducible representations have degree 1. In fact, this is a characterization of Abelian groups. It follows that when G is Abelian, step 2 of the algorithm becomes trivial.

5.4.1 Finding Representations ρ with $\| \hat{f}_\rho \| > \beta$

We now describe how to implement step 1 of the algorithm. Here is quick a overview: we perform a kind of binary search on the representations of G. We partition the

representations into a small number of classes. We then determine which classes contain large coefficients, and repeat the search in those classes. The process terminates when all classes contain only one representation.

We use the product tower of G to partition the representations of G into classes. By Theorem 5.3.5 we know that all representations of G are of the form $\sigma \otimes \tau$ where σ, τ are representations of N_j and G/N_j respectively. For a representation σ of N_j, define the following function on G

$$f_\sigma^{(j)}(x) = \sum_{\rho=\sigma\otimes\tau} \sqrt{d_\rho}\ \mathrm{Tr}(\hat{f}_\rho \rho(x))$$

where the sum is over all representations τ of G/N_j. By Theorem 5.3.4 it should be clear that $f \equiv \sum_\sigma f_\sigma^{(j)}$ since every representation of G is counted exactly once in the sum. By Parseval's identity,

$$\| f_\sigma^{(j)} \|^2 = \sum_{\rho=\sigma\otimes\tau} \| \hat{f}_\rho \|^2$$

Thus, if some representation $\rho = \sigma \otimes \tau$ of G satisfies $\| \hat{f}_\rho \| > \beta$ then certainly $\| f_\sigma^{(j)} \| > \beta$. This leads to the following algorithm:

Algorithm COEF(j, σ)

Input: A representation σ of N_j.

Output: All representations $\rho = \sigma \otimes \tau$ where $\| \hat{f}_\rho \| > \beta$.

1. If $j = t$, where t is the length of the product tower of G, then σ is a representation of $N_t = G$. Output σ and stop.

2. For each representation ν of H do:
   a. Define $\sigma' = \sigma \otimes \nu$ to be a representation of N_{j+1}.
   b. If $\| f_{\sigma'}^{(j+1)} \| > \beta$, then call COEF($j+1, \sigma'$).

Calling COEF(0,T) where T is the trivial representation of $N_0 = \{1\}$ will clearly only produce representations ρ of G for which $\| \hat{f}_\rho \| > \beta$. By the discussion above, all such representations will be generated.

By Parseval's identity, for any N_j we have $\sum_\sigma \parallel f_\sigma^{(j)} \parallel^2 = \parallel f \parallel^2 = 1$. Hence, for any N_j at most $1/\beta^2$ of the $f_\sigma^{(j)}$ can satisfy $\parallel f_\sigma^{(j)} \parallel > \beta$. Thus, at any level of the recursion at most $1/\beta^2$ functions need be checked.

Thus the algorithm will run in polynomial time in $1/\beta$ and will output the correct values. The problem that remains is performing the test in step 2b. Given j and σ the following theorem enables us to quickly approximate $\parallel f_\sigma^{(j)} \parallel^2$.

Theorem 5.4.2

$$\parallel f_\sigma^{(j)} \parallel^2 = \mathop{\mathrm{E}}_{\substack{x\in G \\ y\in N_j \\ z\in N_j}} \left[d_\sigma^2 \cdot f(xy) \cdot f(xz) \cdot \overline{\mathrm{Tr}(\sigma(y))} \cdot \mathrm{Tr}(\sigma(z)) \right]$$

Before proving the theorem we first discuss how it can be used to evaluate $\parallel f_\sigma^{(j)} \parallel^2$. For $(x, y, z) \in G \times N_j \times N_j$ define

$$\alpha(x, y, z) = d_\sigma^2 \; f(xy) \; f(xz) \, \overline{\mathrm{Tr}(\sigma(y))} \; \mathrm{Tr}(\sigma(z))$$

By Lemma 5.3.6 the largest degree d_σ that has to be considered is $(1/\beta)^2$. This implies that $\alpha(x, y, z)$ can be evaluated in time $O(t/\beta^2)$ by evaluating each of the elements on the diagonal of $\sigma(y)$ and $\sigma(z)$.

Since the norm of $\mathrm{Tr}(\sigma(y))$ is bounded by d_σ we see that the norm of the values attained by $\alpha(x, y, z)$ are bounded by $(1/\beta)^4$. To approximate $\parallel f_\sigma^{(i)} \parallel^2$ we sample $\alpha(x, y, z)$ at many uniformly and independently chosen points

$$(x_1, y_1, z_1), \ldots, (x_m, y_m, z_m) \in G \times N_j \times N_j$$

and then approximate $\parallel f_\sigma^{(j)} \parallel^2$ by $\frac{1}{m} \sum_k \alpha(x_k, y_k, z_k)$. To bound the error we use the standard Chernoff bounds (see [AS92, p. 240]). The number of sample points, m, required to make the error less than $\beta/2$ with probability δ comes out to be $O((1/\beta)^{10} \log \frac{1}{\delta})$. A similar method can be used to approximate the value of the entries in $\hat{f}_\rho$ for a representation ρ of G.

Theorem 5.4.2 is crucial for the algorithm to work. The fact that such a theorem exists when the basis is associated with a group is the reason why the group structure is essential. No such theorem holds in general for arbitrary orthonormal bases. The proof of Theorem 5.4.2 requires the following two lemmas.

Lemma 5.4.3 *Let $G = N \times (G/N)$ where $N \lhd G$. Let σ be a representation of N and $\rho = \sigma' \otimes \tau'$ a representation of G where σ' is a representation of N and τ' representation of G/N. Then for any $d_\rho \times d_\rho$ matrix A:*

$$B \stackrel{def}{=} \frac{1}{|N|} \sum_{x \in N} \mathrm{Tr}(A\rho(x)) \cdot \mathrm{Tr}(\sigma(x)^*) = \begin{cases} 0 & \text{if } \sigma' \neq \sigma \\ \frac{1}{d_\sigma} \mathrm{Tr}(A) & \text{if } \sigma' = \sigma \end{cases}$$

Proof Let $E_{k,l}$ be the $d_\rho \times d_\rho$ matrix whose (k,l)'th element is 1 and the rest are 0. We first prove the lemma when $A = E_{k,l}$ for some (k,l). Observe that

$$\mathrm{Tr}(E_{k,l}\rho(x)) = \rho_{l,k}(x)$$

For all $x \in N$ we have $\rho(x) = \sigma' \otimes I_{d_{\tau'}}$. Hence, the function $(\rho_{l,k}(x))_{x \in N}$ is either identically 0 or a function $\sigma'_{u,v}(x)$ for some u, v.

Assume $\sigma \neq \sigma'$. Clearly, if $\rho_{l,k}(x)$ is identically 0 then $B = 0$. Otherwise, by the orthogonality relations for the representations of the group N we get

$$\begin{aligned} B &= \frac{1}{|N|} \sum_{x \in N} \mathrm{Tr}(E_{k,l}\rho(x)) \ \mathrm{Tr}(\sigma(x)^*) = \frac{1}{|N|} \sum_{x \in N} \rho_{l,k}(x) \ \mathrm{Tr}(\sigma(x)^*) = \\ &= \frac{1}{|N|} \sum_{x \in N} \sigma'_{u,v}(x) \ \mathrm{Tr}(\sigma(x)^*) = 0 \end{aligned}$$

Similarly, when $\sigma = \sigma'$ and $k \neq l$ we find that $B = 0$. Since $\mathrm{Tr}(E_{k,l}) = 0$ when $k \neq l$, the lemma is correct in this case as well.

Finally, when $\sigma = \sigma'$ and $k = l$ we need to show that $B = \frac{1}{d_\sigma}$. For $x \in N$ we have $\rho_{k,k}(x) = \sigma_{u,u}(x)$ for some u. Thus, by the orthogonality relations (Theorem 5.3.3) we get

$$B = \frac{1}{|N|} \sum_{x \in N} \sigma_{u,u}(x) \ \mathrm{Tr}(\sigma(x)^*) = \frac{1}{d_\sigma}$$

This proves the lemma for the elementary matrices $E_{k,l}$. Observe that if the theorem is true for matrices X, Y then it is true for any linear combination of X and Y. Since any $d_\rho \times d_\rho$ matrix is a linear combination of elementary matrices, the lemma is true for all matrices. □

Using this lemma we can now show how to evaluate $f_\sigma^{(j)}(x)$ for a given $x \in G$.

Lemma 5.4.4

$$\frac{1}{d_\sigma} f_\sigma^{(j)}(x) = \frac{1}{|N_j|} \sum_{y \in N_j} f(xy)\ \ \mathrm{Tr}(\sigma(y)^*)$$

Proof Expanding the right hand side by using the inversion formula (Theorem 5.3.4) gives

$$\frac{1}{|N_j|} \sum_{y \in N_j} f(xy)\ \ \mathrm{Tr}(\sigma(y)^*) = \frac{1}{|N_j|} \sum_{y \in N_j} \sum_{\rho} \sqrt{d_\rho}\ \ \mathrm{Tr}(\hat{f}_\rho \rho(xy))\ \ \mathrm{Tr}(\sigma(y)^*) =$$

$$\sum_{\rho} \sqrt{d_\rho} \frac{1}{|N_j|} \sum_{y \in N_j}\ \ \mathrm{Tr}(\hat{f}_\rho \rho(x)\rho(y))\ \ \mathrm{Tr}(\sigma(y)^*) = \sum_{\rho = \sigma \otimes \tau} \frac{\sqrt{d_\rho}}{d_\sigma}\ \ \mathrm{Tr}(\hat{f}_\rho \rho(x)) = \frac{1}{d_\sigma} f_\sigma^{(j)}(x)$$

The third equality follows from Lemma 5.4.3 by setting $A = \hat{f}_\rho \rho(x)$. □

Proof of Theorem 5.4.2:
By Lemma 5.4.4 we know that

$$f_\sigma^{(j)}(x) = d_\sigma \mathop{\mathrm{E}}_{y \in N_j} [f(xy)\ \ \mathrm{Tr}(\sigma(y)^*)]$$

Hence,

$$\begin{aligned} \| f_\sigma^{(j)} \|^2 &= \mathop{\mathrm{E}}_{x \in G} \left[|f_\sigma^{(j)}(x)|^2 \right] = \mathop{\mathrm{E}}_{x \in G} \left[\left| d_\sigma \mathop{\mathrm{E}}_{y \in N_j} [f(xy)\ \ \mathrm{Tr}(\sigma^*(y))] \right|^2 \right] = \\ &= \mathop{\mathrm{E}}_{x \in G} \left[d_\sigma^2 \mathop{\mathrm{E}}_{y \in N_j} [f(xy)\, \overline{\mathrm{Tr}(\sigma(y))}] \mathop{\mathrm{E}}_{z \in N_j} [f(xz)\ \ \mathrm{Tr}(\sigma(z))] \right] = \\ &= \mathop{\mathrm{E}}_{\substack{x \in G \\ y \in N_j \\ z \in N_j}} \left[d_\sigma^2 f(xy) f(xz)\, \overline{\mathrm{Tr}(\sigma(y))}\ \ \mathrm{Tr}(\sigma(z)) \right] \end{aligned}$$

This completes the proof of Theorem 5.4.2. □

The algorithm requires the ability to randomly sample elements of N_j. When G is embedded naturally in the boolean cube (as discussed in the paragraph preceding Theorem 5.4.1) this is indeed possible.

5.5 Applications

In this section we discuss some applications of the algorithm of Section 5.4 to learning theory. The first general application is an immediate corollary of Lemma 5.2.1 and Theorem 5.4.1.

Theorem 5.5.1 *Let $\mathcal{F}$ be a a family of functions $f : \{0,1\}^n \to \mathbb{C}$. Suppose there exists a group structure G defined on $\{0,1\}^n$ such that all f in $\mathcal{F}$ satisfy $L_G(f) < n^c$ for some constant c. Then $\mathcal{F}$ can be learned in polynomial time.*

Next we generalize the results in [KM91] for learning decision trees, to decision trees with arbitrary group operations. A group decision tree is similar to a standard decision tree. Internal nodes of the tree may decide which way to branch by performing a test on the input involving certain group operations. The precise definition is as follows:

Definition 5.5.1 *Let G be some group. A G-Decision Tree, denoted G–DT, is a tree where the leaves are labeled $\{1,-1\}$ and the internal nodes are labeled with normal subgroups of G. A node v labeled with a subgroup $N_v \lhd G$ has $|G/N_v|$ outgoing edges, each labeled with a different coset of N_v. A G–DT defines a function $f : G \to \{1,-1\}$ by following a path from the root down to the leaves according to which coset the input is in.*

To illustrate this we show that every standard decision tree may be viewed as a $\mathbb{Z}_2^n$–DT. Let T be a decision tree, i.e. each internal node v of T is labeled with some variable x_v. An input $x \in \{0,1\}^n$ will branch left at node v if $x_v = 0$ and right otherwise. To view T as a $\mathbb{Z}_2^n$–DT label each internal node v with the subgroup N_v defined by

$$N_v = \{x \in \mathbb{Z}_2^n \mid x_v = 0\}$$

where x_v is the label of node v in the tree T. The group N_v is a normal subgroup of $\mathbb{Z}_2^n$ of index 2. The coset of N_v is the set of elements in $\mathbb{Z}_2^n$ having a 1 at coordinate

x_v. Thus, in this resulting $\mathbb{Z}_2^n$–DT, input x will branch left if x is in N_v and right if x is in the coset of N_v. This is clearly equivalent to the original tree T.

The advantage of group decision trees over standard ones is that they can be defined over any domain. Furthermore, a function might only have very large $\mathbb{Z}_2^n$–DTs, but may have a small G–DT for a different 2-group G embedded in the boolean cube. The following theorem shows that the algorithm of Section 5.4 can be used to learn any G–DT in time polynomial in its size.

Theorem 5.5.2 *A function $f : G \to \{1, -1\}$ computed by a $G-DT$ with m leaves satisfies $L_G(f) \leq m$.*

Proof Let $f : G \to \{1, -1\}$ be computed by a G-decision tree T. Let leaf(T) be the set of leaves of T. For a matrix A let $|A| = \sum_{i,j} |(A)_{i,j}|$. For a leaf v define inp(v) to be the set of $x \in G$ that reach leaf v.

$$L_G(f) = \sum_{\rho} |\hat{f}_\rho| = \sum_{\rho} \left| \frac{\sqrt{d_\rho}}{|G|} \sum_{x \in G} f(x)(\rho(x))^* \right| \leq \frac{1}{|G|} \sum_{v \in \text{leaf}(T)} \sum_{\rho} \sqrt{d_\rho} \left| \sum_{x \in \text{inp}(v)} \rho(x) \right|$$

Let $v_1, \ldots, v_d$ be the nodes on the path to the leaf v. Consider the group $N = N_{v_1} \cap \ldots \cap N_{v_d} \lhd G$. It is not difficult to see that inp(v) must be a coset of N. A standard application of Schur's lemma shows that since N is normal in G

$$\sum_{x \in \text{inp}(v)} \rho(x) \neq 0 \quad \Longleftrightarrow \quad \rho|_N \equiv I$$

Since the matrices representing $\rho(x)$ are monomial, in the case where $\rho|_N \equiv I$ we have for any $a \in \text{inp}(v)$:

$$\left| \sum_{x \in \text{inp}(v)} \rho(x) \right| = |N||\rho(a)| = |N| d_\rho$$

When $\rho|_N \not\equiv I$ the above sum is 0. The representations of G satisfying $\rho|_N \equiv I$ can in fact be regarded as representations of G/N, since they are constant on cosets of N. Therefore,

$$\sum_{\rho} \sqrt{d_\rho} \left| \sum_{x \in \text{inp}(v)} \rho(x) \right| = \sum_{\rho|_N \equiv I} \sqrt{d_\rho}\ |N| d_\rho \leq |N| \sum_{\rho|_N \equiv I} d_\rho^2 = |N||G/N| = |G|$$

Hence,

$$L_G(f) \leq \frac{1}{|G|} \sum_{v \in \text{leaf}(T)} |G| = |\text{leaf}(T)|$$

□

This result generalizes the result in [KM91] to trees with arbitrary group operations. It would be interesting to determine whether the results in [Bel92] regarding decision trees generalize to these settings.

We mention a few more applications of these methods. The results for weak learning of polynomial size DNF reported in [BFJ*94] may be extended to arbitrary groups of the form $G = H^n$ for some fixed group H. The implication is that it is possible to generate many different weak approximations for DNF formulae by varying the group H. Of course, all these different approximations are correlated.

Another application comes up when implementing the KM-algorithm. To learn a given function f, one can run the algorithm using various groups and then pick the output that best approximates f. Clearly the most useful group G will be the one which has the lowest $L_G(f)$.

5.6 Generalization to arbitrary groups

So far, the results in this chapter were restricted to learning over product groups. We briefly sketch how the methods can be extended to any p-group for fixed p.

Recall that in the case of a product group, the product tower was used to partition the representations into equivalence classes. In the case of p-groups we use a principal composition series of the group.

Theorem 5.6.1 *Let G be a p-group. G has a principle composition series, i.e. a tower of subgroups*

$$\{1\} = N_0 < N_1 < \ldots < N_t = G$$

where for each $0 < j \leq t$ we have $N_j \lhd G$. Furthermore, $N_j/N_{j-1} \cong \mathbb{Z}_p$.

Proof Any p-group is nilpotent and hence has a central series. The central series can be refined to satisfy the conditions of the theorem. □

We use the principle composition series of G to define functions $f_\sigma^{(j)}$ as we did in Section 4.1. We then generalize Theorem 5.4.2 which will enable us to determine the norm of $f_\sigma^{(j)}$. First we present some elementary results from Clifford theory. See [Cli37, Dor71] for a more thorough presentation.

Let G be a finite group and N a normal subgroup of G. Two irreducible representations τ, σ of N are called *G-conjugate* if there exists $g \in G$ such that for all $x \in N$, $\sigma(x) = \tau(g^{-1}xg)$. Note that all G-conjugates have the same degree.

Let ρ be an irreducible representation of G. Let ρ' be the restriction of ρ to N. The representation ρ' of N will decompose into irreducible representations of N.

Theorem 5.6.2 *Let σ be an irreducible representation of N which is a constituent of ρ'. Let m be the multiplicity of σ in ρ', i.e. the number of irreducible constituents of ρ' equivalent to σ. Then all the irreducible constituents of ρ' are G-conjugate to σ and they all have the same multiplicity m.*

The value of m is called the *N-multiplicity* of ρ and is denoted by m_ρ. An irreducible representation ρ of G is said to *contain* an irreducible representation σ of N if σ is an irreducible constituent of ρ restricted to N. Two irreducible representations of G are called *N-associates* if when restricted to N they have an irreducible constituent in common. Note that associates can only differ in the multiplicity m.

Let N_j be a group from the principal composition series of G. For an irreducible representation σ of N_j define the function:

$$f_\sigma^{(j)} = \sum_\rho \sqrt{d_\rho}\ \mathrm{Tr}(\hat{f}_\rho \rho(x))$$

where the sum is taken over all representations ρ of G that contain σ. Observe that all such ρ are associates of one another.

Theorem 5.6.3 *Let ρ be an irreducible representation of G containing σ with N_j-multiplicity m. Then*

$$\| f_\sigma^{(j)} \|^2 = \mathop{\mathrm{E}}_{\substack{x\in G\\ y\in N_j\\ z\in N_j}} \left[\left(\frac{d_\sigma}{m}\right)^2 f(xy) f(xz) \overline{\mathrm{Tr}(\rho(y))}\ \mathrm{Tr}(\rho(z)) \right]$$

The proof is similar to the proof of Theorem 5.4.2. We only need to generalize Lemma 5.4.3 in the following way:

Lemma 5.6.4 *Let G be a finite group and N a normal subgroup of G. Let ρ, ρ' be irreducible representations of G. Let σ be an irreducible representation of N contained in ρ', and $m_{\rho'}$ be the N-multiplicity of ρ'. Then for any $d_\rho \times d_\rho$ matrix A:*

$$B \stackrel{def}{=} \frac{1}{|N|} \sum_{x\in N} \mathrm{Tr}(A\rho(x))\ \mathrm{Tr}(\rho'(x)^*) = \begin{cases} \frac{m_{\rho'}}{d_\sigma}\ \mathrm{Tr}(A) & \text{if } \rho, \rho' \text{ are } N\text{-associates,} \\ 0 & \text{otherwise.} \end{cases}$$

Proof Sketch As in the proof of Lemma 4.1 it is suffices to prove the lemma when A is an elementary matrix $E_{k,l}$. By Theorem 5.6.2 we know that ρ decomposes over N into conjugate representations $\sigma^1, \ldots, \sigma^r$ of N. Similarly, ρ' decomposes into $\tau^1, \ldots, \tau^{r'}$. Observe that $\mathrm{Tr}(E_{k,l}\ \rho(x)) = \rho_{l,k}(x)$. Hence, $\mathrm{Tr}(E_{k,l}\ \rho(x))_{x\in N}$ is either identically 0 or is $\sigma^w_{u,v}(x)$ for some u, v, w.

If ρ, ρ' are not N-associates then $\sigma^1, \ldots, \sigma^r$ and $\tau^1, \ldots, \tau^{r'}$ are distinct sets of representations of N. Hence, $\sigma^w_{u,v}(x)$ is orthogonal to $\mathrm{Tr}(\tau^s(x)^*)$ implying that $B = 0$.

Assume ρ, ρ' are N-associates then ρ, ρ' decompose over N into the same set of representations $\sigma^1, \ldots, \sigma^r$ of N. Then when $k = l$

$$B = \frac{1}{|N|} \sum_{x\in N} \rho_{k,k}(x)\ \mathrm{Tr}(\rho'(x)^*) = m_{\rho'} \sum_{j=1}^{r} \frac{1}{|N|} \sum_{x\in N} \sigma^w_{u,u}(x)\ \mathrm{Tr}(\sigma^j(x)^*) = \frac{m_{\rho'}}{d_{\sigma^w}} = \frac{m_{\rho'}}{d_\sigma}$$

□

In order to use Theorem 5.6.3 we must know for a given representation ρ of G the values of m_ρ and d_σ. They can be computed without actually decomposing ρ over N_j.

Now that we have defined the functions $f_\sigma^{(j)}$ and showed how to approximate their norm, the rest of the algorithm remains unchanged.

To summarize, we have shown that the algorithm can be made to work for any p-group. Recall that for product groups there exists a polynomial time algorithm for evaluating the representations of the group. This is no longer true for arbitrary p-groups. Thus, this generalization will yield "real" algorithms only when there is sufficient information about the group to efficiently evaluate its representations. Alternatively, this generalization can produce learning algorithms that use oracles for evaluating the representations.

5.7 Related work and some extensions

Goldreich, Rubinfeld and Sudan [GRS95] generalized the Goldreich-Levin theorem to fields of large characteristic and to polynomials of large degrees. They construct an algorithm that given a prime p and $\epsilon > 0$ and provided oracle access to a function $f : \mathbb{F}_p^n \to \mathbb{F}_p$ outputs a list of all linear polynomials in $\mathbb{F}_p[X_1, \ldots, X_n]$ which agree with f on $\frac{1}{p} + \epsilon$ of the inputs. The algorithm runs in polynomial time in n/ϵ. In other words, they learn a linear polynomial from a highly noisy oracle f. They prove a similar result for polynomials of higher degrees though the noise tolerance is not as good.

In the same spirit, the results of this chapter can be adapted to prove the following result: let G be a finite group and p be the smallest prime divisor of $|G|$. There is an algorithm which given oracle access to a function $f : G \to \mathbb{C}$ outputs a list of all representation ρ of G which agree with f on $\frac{1}{p} + \epsilon$ of the inputs. The algorithm runs in polynomial time in $\log |G|/\epsilon$. Hence, a representation of a group can be learned from a highly noisy oracle.

5.8 Conclusions

In this chapter we studied that large coefficient problem. We showed that for orthonormal basis associated with representations of finite groups one can solve the large coefficient problem in expected polynomial time. This generalizes the algorithm described by Goldreich and Levin and has new applications to learning theory.

The application to learning theory shows that one may use different groups structure to learn various target concepts. To illustrate this extra degree of freedom we note that there are 2,328 groups of order 128. Any one of these groups can be used to define an orthonormal basis of the space of function on $\{0,1\}^{7n}$. The learning algorithm can use any one of these bases. Thus, for some problems, learning is reduced to finding a group for which the target concept class has low norm. Notice that the learning algorithm presented here can be made to work over any finite domain by defining a group structure on the domain.

It is worth noting that the generation of arbitrary 2-groups of a given size is quite easy. It can be carried out by picking an arbitrary presentation of a 2-group. Once the group is determined it is possible to compute its representations using an algorithm due to Dixon [Dix70].

Chapter 6

Collusion-Secure Fingerprinting

The next problem we consider is a coding theory problem. The problem arises in the context of fingerprinting digital data. Fingerprinting consists of uniquely marking and registering each copy of the data. This marking allows a distributor to detect any unauthorized copy and trace it back to the user. This threat of detection will supposedly deter users from releasing unauthorized copies.

When fingerprinting digital data a problem arises when users collude. Two different fingerprinted objects can be compared and the differences between them detected. Hence, a set of users can collude to detect the location of the fingerprint. They can then alter the fingerprint to mask their identities. The problem of assigning codewords to users turns out to be an interesting coding theory problem. We develop the general theory involved and present several collusion secure fingerprinting schemes.

Fingerprinting is an extremely old cryptographic technique. For instance, several hundred years ago logarithm tables were protected by fingerprinting them. The idea was to introduce tiny errors in the insignificant digits (i.e. tenth digit right of the decimal point) of $\log x$ for a few random values of x. A different set of x's was chosen for each copy of the logarithm table. If an owner of a logarithm table ever sold illegal copies of his table, the tiny errors in the table enabled the "police" to trace the illegal table back to its owner. The owner would then be prosecuted and penalized in some way.

Nowadays no one is interested in protecting logarithm tables. However, the technique of fingerprinting is still in use. Examples include maps, diamonds and explosives. With the increasing popularity of digital data, there is a strong desire to fingerprint these data as well. Examples of digital data to which fingerprinting may apply include documents, images, movies, music and executables.

As was mentioned above, when fingerprinting digital data one must address the problem of collusion. For instance, suppose the logarithm table discussed above is stored as a file. Each user is given a slightly altered copy of the file. If two users get together they can easily run `diff` on their two files to discover all the locations where the files differ. This simple operation reveals the location of the hidden marks. The users can then remove these marks and resell their logarithm table without ever worrying about being caught. Notice that two users could only detect those marks in which their copies differ. They could not detect marks where both their copies agree. We intend to use this small amount of information which the two users could not remove to trace any copy they generate back to one of them.

Throughout the chapter we use the following terminology: A *mark* is a position in the object which can be in one of s different states. For instance, in the logarithm table example, introducing an error in the value of $\log 3$ means that $\log 3$ is marked. If there are s error values used we say that the mark has s possible states. A *fingerprint* is a collection of marks. Thus, a fingerprint can be thought of as a word of length L over an alphabet Σ of size s. Here L is the number of marks embedded in the object. A *distributor* is the sole supplier of fingerprinted objects. A *user* is the registered owner of a fingerprinted object.

The process of fingerprinting an object involves assigning a unique codeword over Σ^L to each user. The user receives a copy of object with the marks set according to his assigned codeword. By colluding, users can detect a specific mark if it differs between their copies; otherwise a mark can not be detected. The main property the marks should satisfy is that users can not change the state of an undetected mark without rendering the object useless. We assume that marks satisfying these properties exist for the objects being fingerprinted. We refer to this as the *Marking Assumption* for

which a precise definition is given in the next section. Note that if there is no collusion, by the Marking Assumption, fingerprinting is trivial: the fingerprint assigned to each user will be the user's serial number.

There has been much research investigating the Marking Assumption in a variety of domains. Wagner [Wag83] gives a taxonomy of fingerprints and suggests subtle marks for computer software. Marks have been embedded in digital images [TNM90, Car95], in documents [BLMO94], and in computer programs [Glo92]. In all of these domains, our scheme allows these marks to be combined to form secure fingerprints. Thus our results are general, allowing a variety of digital data to be fingerprinted.

Throughout the chapter we use the following notation. Given an l-bit word $w \in \Sigma^l$ and a set $I = \{i_1, \ldots, i_r\} \subseteq \{1, \ldots, l\}$ we denote by $w|_I$ the word $w_{i_1} w_{i_2} \ldots w_{i_r}$ where w_i is the i'th letter of w. We refer to $w|_I$ as the *restriction* of w to the positions in I.

6.1 Fingerprinting codes

We begin by defining some notation. From here on Σ will denote some alphabet of size s representing the s different states of the marks. The letters in Σ will be denoted by the integers 1 to s.

Definition 6.1.1 *A set $\Gamma = \{w^{(1)}, \ldots, w^{(n)}\} \subseteq \Sigma^l$ will be called an (l, n)-**code**. The codeword $w^{(i)}$ will be assigned to user u_i, for $1 \leq i \leq n$. We refer to the set of words in Γ as the **codebook**.*

Definition 6.1.2 *Let $\Gamma = \{w^{(1)}, \ldots, w^{(n)}\}$ be an (l, n)-code and C be a coalition of users. For $i \in \{1, \ldots, l\}$ we say that position i is **undetectable** for C if the words assigned to users in C match in their i'th position. Formally, suppose $C = \{u_1, \ldots, u_c\}$. Then position i is undetectable if $w_i^{(u_1)} = w_i^{(u_2)} = \ldots = w_i^{(u_c)}$.*

As was discussed in the introduction our objective it to design a collusion-secure method of assigning codewords to users. Let C be a coalition of users. We must first characterize the set of objects which the coalition C can generate. Suppose the i'th mark is detectable by the coalition C. The coalition can generate an object in

which the i'th mark is in any of its s states. Furthermore, the coalition can generate an object in which the mark is in an *unreadable* state. When the police recovers an illegal copy of the object it can not determine which state an unreadable mark is in. For instance, in the logarithm table example this would correspond to the coalition creating a table which does not contain the entry $\log 3$ where $\log 3$ is a detectable mark. We denote a mark in an unreadable state by '?'. The resulting set of codewords is called the *feasible* set of the coalition. Formally the feasible set is defined as follows:

Definition 6.1.3 *Let $\Gamma = \{w^{(1)}, \ldots, w^{(n)}\}$ be an (l,n)-code and C be a coalition of users. Let R be the set of undetectable positions for C. Define the* ***feasible set*** *of C as*

$$F(C;\Gamma) = \left\{ w \in (\Sigma \cup \{?\})^l \;\; s.t. \;\; w|_R = w^{(u)}|_R \right\}$$

for some user u in C. Thus the feasible set contains all words which match the coalition's undetectable bits. Usually we omit the Γ and denote $F(C;\Gamma)$ by $F(C)$.

We will occasionally denote $\Sigma \cup \{?\}$ by Σ'. The Marking Assumption discussed in the introduction states that any coalition of c users is only capable of creating an object whose fingerprint lies in the feasible set of the coalition.

Example If two users A and B are assigned the codewords

$$\begin{array}{ccccc} 3 & 2 & 3 & 1 & 2 \\ 1 & 2 & 2 & 1 & 2 \end{array}$$

then their feasible set is $F(AB) = \Sigma' \cdot 2 \cdot \Sigma' \cdot 1 \cdot 2$.

6.2 Protection Against Naive Redistribution

To familiarize the reader with our notation we begin by considering a toy problem which we refer to as *naive redistribution*. Naive redistribution occurs when a user redistributes his copy of the object without altering it. If an unauthorized copy of the object is found containing user u's codeword we would like to say that user u

is guilty. However, u could claim that he was framed by a coalition who created an object containing his codeword. Thus, we would like to construct codes that satisfy the following property: no coalition can collude to frame a user not in the coalition. We usually relax this condition by limiting the size of the coalition to c users. We call such codes c-frameproof codes.

If the code used to fingerprint the object is kept hidden from the users, then the construction of frameproof codes becomes trivial: to every one of the n users assign a unique codeword chosen at random. A coalition of users can not frame a user not in the coalition since they do not know his codeword. We would like to construct codes that are c-frameproof even if the codebook is known to the users. This requirement can be formally stated as follows:

Definition 6.2.1 *A code Γ is c-**frameproof** if every set $W \subset \Gamma$, of size at most c, satisfies $F(W) \cap \Gamma = W$.*

The definition states that in a c-frameproof code, the only codewords in the feasible set of a coalition of at most c users are the codewords of members of the coalition. Thus, no coalition of at most c users can frame a user who is not a member of the coalition. It is interesting to note that for random codes the length of the code must be exponential in c. Otherwise, a coalition of c users is likely to detect all the bits.

6.2.1 Construction of c-frameproof codes

We now show a construction for c-frameproof codes over the binary alphabet $\Sigma = \{0, 1\}$. First we introduce a simple (n, n)-code which is n-frameproof. Define the code $\Gamma_0(n)$ to be the (n, n)-code containing all n-bit binary words with exactly one 1. For example, the code $\Gamma_0(3)$ for three users is $\{100, 010, 001\}$. The following claim is immediate.

Claim 6.2.1 *$\Gamma_0(n)$ is a (n, n)-code which is n-frameproof.*

It is not difficult to see that any n-frameproof code must have length at least n. This follows since any coalition of $n - 1$ users must not be able to detect at least one

of the bit positions; otherwise, they could frame a user not in the coalition. Since there are n coalitions of size $n-1$, the code must have at least length n. Hence, the code length of Γ_0 is optimal. The length of Γ_0 is linear in the number of users and is therefore impractical. We will use the code Γ_0 to construct shorter codes. We first recall some basic definitions from the theory of error correcting codes. See [vL82] for more details.

Definition 6.2.2 *A set, $\mathcal{C}$, of N words of length L over an alphabet of p letters is said to be an* $(L, N, D)_p$-Error Correcting Code *or in short, an $(L, N, D)_p$-ECC, if the Hamming distance between every pair of words in $\mathcal{C}$ is at least D.*

The idea of this construction is to compose the code $\Gamma_0(n)$ with an error correcting code. Let $\Gamma = \{w^{(1)}, \ldots, w^{(n)}\}$ be an (l, p)-code and let $\mathcal{C}$ be a $(L, N, D)_p$-ECC. We denote the composition of Γ and $\mathcal{C}$ by Γ'. The code Γ' is an (lL, N)-code defined as follows: for a codeword $v = v_1 v_2 \ldots v_L \in \mathcal{C}$ let

$$W_v = w^{(v_1)} \| w^{(v_2)} \| \ldots \| w^{(v_L)}$$

where $\|$ means concatenation of strings. The code Γ' is the set of all words W_v, i.e. $\Gamma' = \{W_v \mid v \in \mathcal{C}\}$.

Lemma 6.2.2 *Let Γ be a c-frameproof (l, p)-code and $\mathcal{C}$ be a $(L, N, D)_p$-ECC. Let Γ' be the composition of Γ and $\mathcal{C}$. Then Γ' is a c-frameproof code, provided $D > L\left(1 - \frac{1}{c}\right)$.*

Proof Let C be a coalition of c users. We show that $F(C; \Gamma')$ contains no words from Γ' other than those of C. Let $v^1, \ldots, v^c \in \mathcal{C}$ be the codewords of $\mathcal{C}$ from which the codewords of the coalition C were derived.

Assume towards a contradiction that $F(C; \Gamma')$ contains a word $W \in \{0,1\}^{lL}$ which belongs to a user $u \notin C$. Let $z \in \mathcal{C}$ be the codeword from which W was derived. For all $k = 1, \ldots, c$, the words z and v^k match in less than L/c positions. This follows since the minimal Hamming distance of $\mathcal{C}$ is bigger than $L(1 - 1/c)$. Hence, there must exist a position $1 \leq j \leq L$ for which $z_j \neq v^k_j$ for all $k = 1, \ldots, c$.

Let $C_j = \{w^{(v_j^1)}, \ldots, w^{(v_j^c)}\}$. Since Γ is a c-frameproof code we know that $w^{(z_j)}$ is not in $F(C_j; \Gamma)$. Since $w^{(z_j)}$ is a subword of W, this implies that $W \notin F(C; \Gamma')$. This contradiction proves the lemma. □

We note that the condition that $\mathcal{C}$ has a large minimal distance can be relaxed. To make the proof work it suffices to require that no set of c words of $\mathcal{C}$ "cover" a word of $\mathcal{C}$ outside the set. This property has been studied in [EFF85]. Using this relaxed requirement does not improve the constructions.

The error correcting codes we are using have large minimal distance and hence, low rate. By picking the codewords randomly it is possible to obtain a good low rate code. We state this in the following lemma, which is immediate from the Chernoff bound [AS92, Appendix A].

Lemma 6.2.3 *For any positive integers p, N let $L = 8p \log N$. Then there exists a $(L, N, D)_{2p}$-ECC where $D > L(1 - \frac{1}{p})$.*

The main theorem of this section now follows.

Theorem 6.2.4 *For any integers $n, c > 0$ let $l = 16c^2 \log n$. Then there exist an (l, n)-code which is c-frameproof.*

Proof By Lemma 6.2.3 we know that there exists an error correcting code with parameters $(L, n, L(1-1/c))_{2c}$ for $L = 8c \log n$. Combining this with the code $\Gamma_0(2c)$ and Lemma 6.2.2 we get a c-frameproof code for n users whose length is $2cL = 16c^2 \log n$. □

To make this construction explicit we must use an explicit low rate error correcting code. Explicit constructions of such codes are described in [ABN*92]. The explicit construction are not as good as the bounds provided by Lemma 6.2.3. Using a simple explicit low rate code it is possible to obtain codes of length $l = c^2 \log^2(n)$.

6.3 c-secure Codes

We now turn our attention to the full problem of collusion secure fingerprinting. Suppose a distributor marks an object with a code Γ. Now, suppose a coalition of users, C, colludes to generate an unregistered object marked by some word x and then distributes this new object. When this object is found, the distributor would like to detect a subset of the coalition who created it. In other words, there must exist a *tracing algorithm* A which on input x output a member of the coalition C. For our purposes the tracing algorithm A may be regarded as a function $A : \{0,1\}^n \rightarrow \{1, \ldots, n\}$ where n in the number of users. This leads to the following definition:

Definition 6.3.1 *A code Γ is **totally c-secure** if there exists a tracing algorithm A satisfying the following condition: if a coalition C of at most c users generates a word x then $A(x) \in C$.*

The tracing algorithm A on input x must output a member of the coalition C that generated the word x. Hence, an illegal copy can be traced back to at least one member of the guilty coalition. Clearly there is no hope in recovering the entire coalition since some of its members might be passive; they are part of the coalition, but they contribute nothing to the construction of the illegal copy.

We now derive a necessary condition of a code to be totally c-secure. Consider the following scenario: let Γ be some code. Let C_1 and C_2 be two coalitions of c users each such that $C_1 \cap C_2 = \emptyset$. Suppose an unregistered object is found which is marked by a codeword x which is feasible for both C_1 and C_2. Then both coalitions are suspect. Since their intersection is empty, it is not possible to determine with certainty who created the unregistered object. It follows that if Γ is totally c-secure then when the intersection of C_1 and C_2 is empty, the intersection of their feasible sets $F(C_1)$ and $F(C_2)$ must also be empty. In general we obtain the following lemma:

Lemma 6.3.1 *If Γ is a totally c-secure code then*

$$C_1 \cap \ldots \cap C_r = \emptyset \quad \Rightarrow \quad F(C_1) \cap \ldots \cap F(C_r) = \emptyset \ .$$

for all coalitions $C_1, \ldots, C_r$ of at most c users each.

It seems that totally secure codes provide a good solution to the problem of collusion. Unfortunately, when $c > 1$, totally c-secure codes do not exist.

Theorem 6.3.2 *For $c \geq 2$ and $n \geq 3$ there are no totally c-secure (l, n)-codes.*

Proof Clearly, it is enough to show that there are no totally 2-secure codes. Let Γ be an arbitrary (l, n)-code. Let $w^{(1)}, w^{(2)}, w^{(3)}$ be three distinct codewords assigned to users u_1, u_2, u_3 respectively. Define the majority word $M = \mathrm{MAJ}(w^{(1)}, w^{(2)}, w^{(3)})$ by

$$M_i = \begin{cases} w_i^{(1)} & \text{if} \quad w_i^{(1)} = w_i^{(2)} \ \text{ or } \ w_i^{(1)} = w_i^{(3)} \\ w_i^{(2)} & \text{if} \quad w_i^{(2)} = w_i^{(3)} \\ ? & otherwise \end{cases}$$

One can readily verify that the majority word M is feasible for all three coalitions $\{u_1, u_2\}, \{u_1, u_3\}, \{u_2, u_3\}$. However the intersection of the coalitions is empty. Therefore by Lemma 6.3.1 the code Γ is not totally 2-secure. □

The proof of the theorem shows that if a coalition employs the 'majority' strategy it is guaranteed to defeat all fingerprinting codes. Based on this result it seems that all is lost: fingerprinting is not possible in the presence of collusion. Fortunately there is a way out of this trap: *use randomness.*

Theorem 6.3.2 forces us to weaken our requirements for marking schemes. We intend to allow the distributor to make some random choices when embedding the codewords in the objects. The point is that the random choices will be kept hidden from the users. This enables us to construct codes which will capture a member of the guilty coalition with high probability.

An (l, n) fingerprinting scheme is a function $\Gamma(u, r)$ which maps a user number $1 \leq u \leq n$ and a string of random bits $r \in \{0, 1\}^*$ to a codeword in Σ^l. The random string r is the set of random bits used by the distributor and kept hidden from the users. We denote a fingerprinting scheme by Γ_r.

Suppose a coalition C of c users creates an illegal copy of an object. Fingerprinting schemes that enable the capture of a member of the coalition C with probability at

least $1-\epsilon$ are called *c-secure codes with ϵ-error*. Here the probability is taken over the random choices made by the distributor and the random choices made by the coalition C.

Definition 6.3.2 *A fingerprinting scheme Γ_r is c-secure with ϵ-error if there exists a tracing algorithm A satisfying the following condition: if a coalition C of at most c users generates a word x then*

$$\Pr[A(x) \in C] > 1 - \epsilon$$

where the probability is taken over the random bits r and the random choices made by the coalition.

The tracing algorithm A on input x outputs a member of the coalition C that generated the word x *with high probability*. With this definition at hand we turn to the construction of c-secure codes.

We point out that Chor, Fiat and Naor [CFN94] considered a similar problem in an entirely different settings. In our terms their result enables one to construct c-secure codes under the assumption that marks can not become unreadable. Under this assumption one can even construct totally c-secure codes. Indeed, the proof of Theorem 6.3.2 relied on the existence of unreadable marks.

Unfortunately, in the context of fingerprinting the assumption that marks can not become unreadable is unrealistic. As was discussed in Section 6.1 once a coalition detects a mark, that mark can be made unreadable in various ways (recall that by unreadable we mean that when an illegal copy is discovered it is impossible to determine which state the mark is in). For this reason, the results of Chor, Fiat and Naor are too weak to be used for fingerprinting.

6.4 Construction of collusion secure codes

The idea for the construction c-secure codes is similar to the one used in Section 6.2.1. We first construct an (l, n)-code which is n-secure. Thus, no matter how large the

coalition is, we will be able to trace an illegal copy back to a member of the coalition with high probability. The length of this code is $n^{O(1)}$ and hence, too large to be practical. We then show how this code can be used to construct c-secure codes for n users whose length is $\log^{O(1)}(n)$ when $c = O(\log n)$.

We begin by presenting an (l, n)-code which is n-secure with ϵ-error for any $\epsilon > 0$. Let c_m be a column of height n in which the first m bits are 1 and the rest are 0. The code $\Gamma_0(n, d)$ will consist of all columns $c_1, \ldots, c_{m-1}$, each duplicated d times. The amount of duplication will determine the error probability ϵ. For example, the code $\Gamma_0(4, 3)$ for four users A, B, C, D is

$$\begin{aligned} A: &\quad 111111111 \\ B: &\quad 000111111 \\ C: &\quad 000000111 \\ D: &\quad 000000000 \end{aligned}$$

Let $w^{(1)}, \ldots, w^{(n)}$ denote the codewords of $\Gamma_0(n, d)$. Before the distributor embeds the codewords of $\Gamma_0(n, d)$ in an object he makes the following random choice: the distributor randomly picks a permutation[1] $\pi \in S_l$. User u_i's copy of the object will be fingerprinted using the word $\pi w^{(i)}$. Note that the same permutation π is used for all users. The point is that π will be kept hidden from the users. Keeping the permutation hidden from the users is equivalent to hiding the information of which mark in the object encodes which bit in the code. It is a bit surprising that this simple random action taken by the distributor is sufficient to overcome the barrier of Theorem 6.3.2 and enables us to prove the following theorem:

Theorem 6.4.1 *For $n \geq 3$ and $\epsilon > 0$ let $d = 2n^2 \log(2n/\epsilon)$. The fingerprinting scheme $\Gamma_0(n, d)$ is n-secure with ϵ-error.*

The length of this code is $d(n-1) = O(n^3 \log \frac{n}{\epsilon})$. To prove the theorem we must

[1] S_l is the full symmetric group of all permutations on l letters. For a word $x \in \{0,1\}^l$ and a permutation $\pi \in S_l$ we denote by πx the l-bit word $x_{\pi(1)} x_{\pi(2)} \ldots x_{\pi(l)}$.

describe an algorithm, which given a word x generated[2] by some coalition C, outputs a member of C with probability $1 - \epsilon$. First we introduce some notation.

1. Let B_m be the set of all bit positions in which the users see columns of type c_m. That is, B_m is the set of all bit positions in which the first m users see a 1 and the rest see a 0. The number of elements in B_m is d.

2. For $2 \leq s \leq n-1$ define $R_s = B_{s-1} \cup B_s$.

3. For a binary string x, let weight(x) denote the number of 1's in x.

Before we describe the algorithm we give some intuition. Suppose user s is *not* a member of the coalition C which produced the word x. The hidden permutation π prevents the coalition from knowing which marks represent which bits in the code $\Gamma_0(n, d)$. The only information the coalition has is the value of the marks it can detect. Observe that without user s a coalition sees exactly the same values for all bit positions $i \in R_s$. For instance, in the code $\Gamma_0(4, 3)$ above, the coalition A, C, D sees the exact same bit pattern for all bit positions in R_2. Hence, for a bit position $i \in R_s$, the coalition C can not tell if i lies in B_s or in B_{s-1}. This means that whichever strategy they use to set the bits of $x|_{R_s}$, the 1's in $x|_{R_s}$ will be roughly evenly distributed between $x|_{B_s}$ and $x|_{B_{s-1}}$ with high probability. Hence, if the 1's in $x|_{R_s}$ are not evenly distributed then, with high probability, user s is a member of the coalition that generated x.

Algorithm 1 *Given $x \in \{0,1\}^l$, find a subset of the coalition that produced x.*
1. If weight($x|_{B_1}$) > 0 then output "User 1 is guilty".
2. If weight($x|_{B_{n-1}}$) $< d$ then output "User n is guilty".
3. for all $s = 2$ to $n-1$ do: Let $k =$ weight($x|_{R_s}$). If

$$\textit{weight}(x|_{B_{s-1}}) < \frac{k}{2} - \sqrt{\frac{k}{2} \log \frac{2n}{\epsilon}}$$

then output "User s is guilty".

[2]When we say that a coalition C generated a word x, we mean that the bits of x have already been unscrambled using π^{-1}. For example, the first bit of x is the value of the mark which encodes the first bit of the codewords.

One issue needs some clarification: the word x found in the illegal copy may contain some unreadable marks '?'. As a convention these bits are set to '0' before the word x is given to Algorithm 1. As a result the algorithm indeed receives a word in $\{0,1\}^l$. The correctness of Algorithm 1 is proved in the next two lemmas.

Lemma 6.4.2 *Consider the code $\Gamma_0(n,d)$ where $d = 2n^2 \log(2n/\epsilon)$. Let S be the set of users which algorithm 1 pronounces as guilty on input x. Then with probability at least $1-\epsilon$, the set S is a subset of the coalition C that produced x.*

Proof Suppose user 1 was pronounced guilty, i.e. $1 \in S$. Then $\text{weight}(x|_{B_1}) > 0$. This implies that user 1 must be a member of C (otherwise the bits in B_1 would be undetectable for C which would imply that $\text{weight}(x|_{B_1}) = 0$). Similarly if $n \in S$ then $n \in C$.

Suppose the algorithm pronounces user $1 < s < n$ as guilty. We show that the probability that s is not guilty is at most ϵ/n. This will show that the probability that there exists a user in S which is not guilty is at most ϵ.

Let s be an innocent user, i.e. $s \notin C$. As was discussed above, this means that the the coalition C can not distinguish between the bit positions in R_s. Since the permutation π was chosen uniformly at random from the set of all permutations, the 1's in $x|_{R_s}$ may be regarded as being randomly placed in $x|_{R_s}$. Let $k = \text{weight}(x|_{R_s})$. Define Y to be a random variable which counts the number of 1's in $x|_{B_{s-1}}$ given that $x|_{R_s}$ contains k 1's. For any integer r in the appropriate range:

$$\Pr[Y = r] = \frac{\binom{M}{r}\binom{M}{k-r}}{\binom{2M}{k}}$$

where $M = 2n^2 \log(2n/\epsilon)$ is the size of B_{s-1}. Clearly, the expectation of Y is $k/2$. To bound the probability that s was pronounced guilty we need to bound

$$\Pr\left[Y < k/2 - \sqrt{\frac{k}{2}\log\frac{2n}{\epsilon}}\right]$$

from above. This can be done by comparing Y to an appropriate binomial random variable.

Let X be a binomial random variable over k experiments with success probability $1/2$. A routine calculation shows that for any r we have that $\Pr[Y = r] \leq 2\Pr[X = r]$. This means that for any $a > 0$

$$\Pr\left[Y - \frac{k}{2} < a\right] \leq 2\Pr\left[X - \frac{k}{2} < a\right] \leq 2e^{-2a^2/k}$$

where the last inequality follows from the standard Chernoff bound [AS92, Appendix A]. Plugging in $a = \sqrt{\frac{k}{2}\log\frac{2n}{\epsilon}}$ leads to

$$\Pr\left[Y < \frac{k}{2} - \sqrt{\frac{k}{2}\log\frac{2n}{\epsilon}}\right] \leq 2e^{-\log(2n/\epsilon)} = \frac{\epsilon}{n} \ .$$

Thus, if user s is innocent then the probability of her being pronounced guilty by Algorithm 1 is at most ϵ/n. Therefore, the probability that some innocent user will be pronounced guilty is at most ϵ. This proves the lemma. □

Lemma 6.4.3 *Consider the code $\Gamma_0(n, d)$ where $d = 2n^2\log(2n/\epsilon)$. Let S be the set of users which algorithm 1 pronounces as guilty on input x. Then the set S in not empty.*

The proof of the lemma relies on the following claim.

Claim 6.4.4 *Suppose the set S is empty. Then for all s we have weight$(x|_{B_s}) \leq 2s^2\log(2n/\epsilon)$.*

Proof The proof is by induction on s. For $s = 1$, the claim is immediate since if user 1 is not guilty then weight$(x|_{B_1}) = 0$.

Now, we assume the claim holds for $s < n - 1$ and prove it for $s + 1$. Define

$$k = \text{weight}(x|_{B_s}) \quad ; \quad k' = \text{weight}(x|_{B_{s+1}}) \quad ; \quad t = \text{weight}(x|_{R_{s+1}}) \ .$$

Then the following three conditions are satisfied:

$$t = k + k' \quad ; \quad k \leq 2s^2\log(2n/\epsilon) \quad ; \quad k \geq \frac{t}{2} - \sqrt{\frac{t}{2}\log\frac{2n}{\epsilon}}$$

The first condition follows from the fact that $R_{s+1} = B_s \cup B_{s+1}$. The second is the inductive hypothesis and the third follows from the fact that user s was not pronounced guilty, i.e. $s \notin S$. We show that these three conditions imply $k' \leq 2(s+1)^2 \log(2n/\epsilon)$ which will prove the claim for $s+1$.

$$\begin{aligned} k' &= t - k \leq \frac{t}{2} + \sqrt{\frac{t}{2}\log\frac{2n}{\epsilon}} = \frac{k+k'}{2} + \sqrt{\frac{1}{2}(k+k')\log\frac{2n}{\epsilon}} \\ &\leq \frac{2s^2\log(2n/\epsilon) + k'}{2} + \sqrt{\frac{1}{2}(2s^2\log\frac{2n}{\epsilon} + k')\log\frac{2n}{\epsilon}} \end{aligned}$$

which leads to

$$k' \leq 2s^2\log\frac{2n}{\epsilon} + \sqrt{2(2s^2\log\frac{2n}{\epsilon} + k')\log\frac{2n}{\epsilon}} \ .$$

Suppose $k' = 2r^2\log(2n/\epsilon)$ for some constant r. Substituting for k' and dividing by $2\log(2n/\epsilon)$ we get

$$r^2 \leq s^2 + \sqrt{s^2 + r^2} \ .$$

It is not difficult to see that for this inequality to be satisfied when $s \geq 1$ we must have $r \leq s+1$. Hence, $k' \leq 2(s+1)^2\log(2n/\epsilon)$. □

Proof of Lemma 6.4.3 Suppose S is empty. Since user n was not pronounced guilty we know that $\text{weight}(x|_{B_{n-1}}) = d = 2n^2\log(2n/\epsilon)$. On the other hand, for $s = n-1$ Claim 6.4.4 shows that $\text{weight}(x|_{B_{n-1}}) \leq 2(n-1)^2\log(2n/\epsilon)$. This contradiction proves the lemma. □

This completes the proof of Theorem 6.4.1.

6.4.1 Logarithmic length c-secure codes

The n-secure code constructed in the previous section enables us to use the techniques of [CFN94] to construct c-secure (n, l)-codes of length $l = c^{O(1)}\log n$. We thank Naor for pointing out this relation. In this section we demonstrate how to apply the simplest technique from [CFN94] to construct a short c-secure code from the n-secure code of

the previous section. The basic idea is to use the n-secure code as the alphabet over which the techniques of [CFN94] can be applied.

Let $\mathcal{C}$ be an (L, N)-code over an alphabet of size n where the codewords are chosen independently and uniformly at random[3]. The idea is to compose our n-secure code $\Gamma_0(n, d)$ with the code $\mathcal{C}$ as we did in the proof of Lemma 6.2.2. We call the resulting code $\Gamma'(L, N, n, d)$. Thus, the code $\Gamma'(L, N, n, d)$ contains N codewords and has length $Ld(n-1)$. It is made up of L copies of $\Gamma_0(n, d)$. We will refer to these copies as the *components* of $\Gamma'(L, N, n, d)$. The point is that the codewords of the code $\mathcal{C}$ will be kept hidden from the users. This is in addition to keeping hidden the L permutations used when embedding the L copies of $\Gamma_0(n, d)$ in the object.

Theorem 6.4.5 *Given integers N, c and $\epsilon > 0$ set $n = 2c$; $L = 2c\log(2N/\epsilon)$ and $d = 2n^2\log(4nL/\epsilon)$. Then, $\Gamma'(L, N, n, d)$ is a code which is c-secure with ϵ-error. The code contains N words and has length $l = O(Ldn) = O(c^4\log(N/\epsilon)\log(1/\epsilon))$.*

To prove the theorem we show an algorithm that finds a member of the guilty coalition and then prove it's correctness.

Algorithm 2 *Given $x \in \{0,1\}^l$, find a member of the coalition that produced x.*

1. *Apply Algorithm 1 to each of the L components of x. For each component $i = 1, \ldots, L$ arbitrarily choose one of the outputs of Algorithm 1. Set y_i to be this chosen output. Note that y_i is a number between 1 and n. Next, form the word $y = y_1 \ldots y_L$.*

2. *Find the word $w \in \mathcal{C}$ which matches y in the most number of position (ties are broken arbitrarily).*

3. *Let u be the user whose codeword is derived from $w \in \mathcal{C}$. output "User u is guilty".*

[3]In [CFN94] the codewords of $\mathcal{C}$ are considered as random hash functions $h : \{1, \ldots, L\} \to \{1, \ldots, n\}$.

Lemma 6.4.6 *Let x be a word which was produced by a coalition C of at most c users. Then with parameters as in Theorem 6.4.5, Algorithm 2 will output a member of C with probability at least $1 - \epsilon$.*

Proof Sketch Let W be the set of codewords in $\mathcal{C}$ that correspond to the users in the coalition C. For every $1 \leq i \leq L$ Algorithm 1 guarantees that y_i will match w_i for some $w \in W$ with probability $1 - \epsilon/2L$. This follows from the choice of d and the fact that in component i the users in C see words from $\Gamma_0(n, d)$. It follows that the above condition will be satisfied in *every* component with probability at least $1 - \epsilon/2$. We refer to this as event A.

Recall that the size of W is at most c. Therefore, when event A occurs there must exist a word $w \in W$ which matches y in L/c positions. However, since the words in $\mathcal{C}$ are random and hidden from the users, any word in $\mathcal{C}$ which is not in W is expected to match y in only $L/n = L/2c$ positions. Using the Chernoff bound it can be shown that the probability that a random word will match y in L/c positions is less than $\epsilon/2N$. Hence, the probability the some word in $\mathcal{C} \setminus W$ will match y in L/c positions is at most $\epsilon/2$. This shows that when event A occurs, the algorithm will output a member of C with probability at least $1 - \epsilon/2$. Combining this with the fact that event A occurs with probability at least $1 - \epsilon/2$ proves the lemma. □

6.5 A lower bound

The following theorem provides a lower bound on the length of c-secure codes.

Theorem 6.5.1 *Let Γ be an (l, n) fingerprinting scheme over a binary alphabet. Suppose Γ is c-secure with ϵ error. Then the code length is at least $l \geq \frac{1}{2}(c-3)\log\frac{1}{\epsilon}$.*

Proof Recall that $\Gamma(u, r)$ is a function mapping a user u and a random string r to a codeword in $\{0,1\}^l$. For our purposes one may regard r as a fixed string. Let $1, \ldots, c+1$ be a set of $c+1$ users. Define $B_r(k)$ to be the set of bit positions where

users $1, \ldots, c+1$ see the value '1' exactly k times. More precisely:

$$B_r(k) = \left\{ i \;\middle|\; k = \#\{u \text{ s.t. } \Gamma(u,r)|_i = 1\} \right\}$$

Suppose $l < \frac{1}{2}(c-3)\log\frac{1}{\epsilon}$. We show that a coalition of c users can create a word which can not be traced. Observe that for all values of the random string r there must exist a $2 \leq k_0 \leq c-2$ such that

$$|B_r(k_0) \cup B_r(k_0+1)| \leq \log\frac{1}{\epsilon}$$

This follows since $\sum_{k=2}^{c-2} |B_r(k) \cup B_r(k+1)| \leq 2l < (c-3)\log\frac{1}{\epsilon}$. Define the word W_r by

$$(W_r)_i = \begin{cases} 0 & \text{if } i \in B_r(k) \text{ for some } k \leq k_0 \\ 1 & \text{otherwise} \end{cases} \tag{3}$$

In other words, W_r is '0' for all bit positions in blocks of weight $k \leq k_0$ and '1' in all other bit positions. We show that all coalitions of c users in $1, \ldots, c+1$ can create the word W_r with probability at least ϵ. Let C be such a coalition. Since $2 \leq k_0 \leq c-2$ the coalition C can determine the value of k_0 (the coalition can detect all bit positions in $B_r(k_0) \cup B_r(k_0+1)$ for k_0 in this range).

Let i be a detectable bit position for C. Since C includes all but one of the users in $1, \ldots, c+1$ it can determine that $i \in B_r(k) \cup B_r(k+1)$ for some k. However, it can not differentiate the case $i \in B_r(k)$ from $i \in B_r(k+1)$. Consequently, if the coalition sees the value '1' less than k_0 times in the i'th position (i.e. $\#\{u \in C \text{ s.t. } \Gamma(u,r)|_i = 1\} < k_0$) it must be case that $i \in B_r(k)$ for some $k \leq k_0$. Therefore, the coalition sets such bit positions to '0' (which is consistent with condition (3) defining W_r). Similarly if the value '1' appears at least k_0+1 times in the i'th position then $i \in B_r(k)$ for some $k \geq k_0+1$. Consequently, the coalition may safely set this bit position to '1'.

The case where the coalition sees '1' exactly k_0 times is a bit harder. In this case the coalition can not determine if $i \in B_r(k_0)$ or $i \in B_r(k_0+1)$. Hence it can not deterministically decide if the bit should be set to '0' or '1'. For such i the coalition flips an unbiased binary coin and sets the the i'th bit to the value of the coin. The

probability that all such bit positions are set correctly (i.e. the resulting word is W_r) is exactly

$$\left(\frac{1}{2}\right)^{|B_r(k_0)\cup B_r(k_0+1)|} \geq \left(\frac{1}{2}\right)^{\log\frac{1}{\epsilon}} = \epsilon$$

Hence with probability ϵ the coalition generates the word W_r. Since this holds for every coalition C of c users in $1,\ldots,c+1$ and any random string r it follows that the word W_r can not be traced back to a single user with probability ϵ. This completes the proof of the theorem. □

6.6 Distribution Scheme

Up until now, we have been ignoring distribution of the uniquely fingerprinted copies. This is fair, as at worst we can send each user an entire unique copy. However, this is impractical for products such as electronic books, software or CD-ROMs which are mass produced. We would like to come up with a scheme in which a user receives a bulk of data common to all users, and a small number of extra bits unique to him. We refer to the bulk of data as the *public data* and denote it by D. We refer to the extra bits given privately to user u as the *private data* and denote it by M_u. For the distribution scheme to be secure, given (D, M_u), user u should not be able to deduce any information about the fingerprints in copies given to other users.

Throughout this section, let Γ be an (l,n)-code which is used to fingerprint the objects. We denote the object to be distributed by P and let L be its length. Assume the object P can be partitioned into l pieces with exactly one mark in each piece. Let $p(r,s)$ be the rth piece which contains the rth mark in state $s \in \{0,1\}$. For any r, the pieces $p(r,0)$ and $p(r,1)$ are interchangeable, that is a copy with one replaced with the other will behave identically. Given a codeword $x = x_1x_2\ldots x_l \in \Gamma$, let $P(x) = \{p(1,x_1), p(2,x_2), \ldots, p(l,x_l)\}$ be the partition set implied by x.

Theorem 6.6.1 *It is possible to solve the distribution problem using $O(L)$ size public data and $O(L^{1/m}l/\log L)$ private data for some fixed constant m.*

Proof Sketch We encode each piece $p(r,s)$ with a secure private key cryptosystem. See [Gol90] for the precise definitions; such systems are known to exist under some standard hardness assumptions. Let $f_k : \{0,1\}^{L'} \to \{0,1\}^{L'}$ be such a system, where $L' = L/l$ is the length of a single piece $p(r,s)$. The key k will have length $K = (L')^{1/m}$ for some fixed constant m. The fact that f_k is a secure private key cryptosystem implies that for a random key k, given $f_k(x)$, no polynomial time predicate can extract one bit of information about x with non negligible probability. This property is crucial for the security of our distribution scheme.

For each piece, $p(r,s)$, pick a random key $k(r,s)$. The public data will be

$$D = \left\{ f_{k(r,s)}\Big(p(r,s)\Big) \;\mid\; 1 \le r \le l,\ s \in \{0,1\} \right\} .$$

The size of D is $2L$, twice the size of the original object. Let $x \in \Gamma$ be the word associated with user u. The private data given to user u is the collection of keys necessary to decrypt his pieces:

$$M_u = \{k(1,x_1), k(2,x_2), \ldots, k(l,x_l)\} .$$

Using this scheme, given (D, M_u), any user u can construct a usable copy of the distributed object. The size of the private data, M_u is $lK = O(l^{1-1/m} L^{1/m}) = O(lL^{1/m})$.

It is possible to further reduce the size of the private data by using double encryption: we group the keys $k(r,s)$ into blocks. Each block is encrypted and made public. Each user will receive keys that will enable him to decrypt his block of keys. This method requires public data of size $O(L)$ and private data of size $O(L^{1/m} \frac{l}{\log L})$ for some constant $m > 1$. □

It is worth noting that when implementing this distribution scheme, one can use a standard private key cryptosystem such as DES. Such systems use fixed length keys. This leads to private data of length $O(l/\log(L))$.

6.7 Conclusions

The most significant contribution of these results is to show how to overcome collusion when fingerprinting digital data. To summarize our results, we restrict the size of coalitions to be at most $\log n$ where n is the number of users. For the problem of naive redistribution considered in Section 6.2, we constructed codes of length $\log^3 n$. For the general redistribution problem considered in Section 6.3 we constructed codes of length $\log^6 n$. Furthermore, we demonstrated an efficient method for shipping fingerprinted data which requires only a small constant factor increase in the size of the data.

There are still many open problems which remain to be solved. The most relevant one is that of constructing shorter collusion secure codes. It seems that an n-secure code which is shorter than the one constructed in Section 6.4 should exist. Such a code would lead to an improvement in the general construction of c-secure codes.

Recall that throughout the chapter, we assumed that secure marks can be embedded in the fingerprinted data. A mark encodes a bit of information and is secure if it can only be detected by collusion. To emphasize the fact that we will not be dealing with the implementation of secure marks, we referred to the assumption that they exist as the "Marking Assumption". In many domains, one can construct secure marks with the aid of problems that are believed to be hard. For instance, when fingerprinting movies, a single mark can be encoded by using one camera view point versus another. The choice of one view point versus another in a specific scene, encodes one bit of information in the film. Given an image, the problem of transforming the image to an image taken from a different view point is believed to be hard. As this method of marking can be used to fingerprint movies, we say that the Marking Assumption holds in the domain of movies.

Showing that the Marking Assumption is satisfied for software is much harder. As was stated in the introduction, there is a great deal of empirical evidence to support the existence of secure marks in software. However, to the best of our knowledge, no formal results have been obtained. Progress in this direction would be of great practical importance.

Bibliography

[ABN*92] N. Alon, J. Bruck, J. Naor, M. Naor, and R. Roth. Construction of asymptotically good low-rate error-correcting codes through pseudo-random graphs. *IEEE Transactions on Information Theory*, 38:509–516, 1992.

[ACGS88] W. Alexi, B. Chor, O. Goldreich, and C. Schnorr. RSA and Rabin functions: Certain parts are as hard as the whole. *SIAM J. Computing*, 17(2):194–209, Nov. 1988.

[ADH94] L. Adleman, J. DeMarrais, and Ming-Deh Huang. A sub-exponential algorithm for discrete logarithm over the rational subgroup of the Jacobian of large genus hyperelliptic curves over finite fields. In *First Algorithmic Number Theory Symposium*, 1994.

[AF90] M. Abadi and J. Feigenbaum. Secure circuit evaluation: a protocol based on hiding information from an oracle. *J. of Cryptology*, 3(2):1–12, 1990.

[AS92] N. Alon and J. Spencer. *The probabilistic method.* Wiley, 1992.

[Bab86] L. Babai. On Lovasz' lattice reduction and the nearest lattice point problem. *Combinatorica*, 6:1–13, 1986.

[Bac90] E. Bach. Explicit bounds for primality testing and related problems. *Math. Comp.*, 55:355–380, 1990.

[BC90] Jurjen Bos and Matthijs Coster. Addition chain heuristics. In *Proc. CRYPTO 89*, pages 400–407. Springer-Verlag, 1990.

[Bel92] M. Bellare. A technique for upper bounding the spectral norm with applications to learning. In *Proc. 5th Annu. Conf. on Comp. Learning Theory*, pages 62–70, 1992.

[BFJ*94] A. Blum, M. Furst, J. Jackson, M. Kearns, Y. Mansour, and S. Rudich. Weakly Learning DNF and Characterizing Statistical Query Learning Using Fourier Analysis. In *Proc. 26th ACM Symp. on Theory of Computing*, pages 253–262, 1994.

[BL96] D. Boneh and R. Lipton. Black box fields and their application to cryptography. In *Proc. CRYPTO 96*, 1996. to appear.

[BLMO94] J. Brassil, S. Low, N. Maxemchuk, and L. O'Gorman. Electronic marking and identification techniques to discourage document copying. In *Proceedings of Infocom '94*, pages 1278–1287, June 1994.

[BM84] M. Blum and S. Micali. How to generate cryptographically strong sequences of pseudo-random bits. *SIAM J. Computing*, 13:850–864, 1984.

[BM89] M. Bellare and S. Micali. Non-interactive oblivious transfer and applications. In *Proc. CRYPTO 89*, pages 547–557, 1989.

[BO83] M. Ben-Or. Lower bounds for algebraic computation trees. In *Proc. 15th ACM Symp. on Theory of Computing*, pages 80–86, 1983.

[Bon95] D. Boneh. Learning using group representations. In *Proc. 8th Annu. Conf. on Comp. Learning Theory*, pages 418–427, 1995.

[BS84] L. Babai and E. Szemerédi. On the complexity of matrix group problems I. In *Proc. 25th IEEE Symp. on Foundations of Comp. Science*, pages 229–240, 1984.

[BS95] D. Boneh and J. Shaw. Collusion secure fingerprinting for digital data. In *Proc. CRYPTO 95*, pages 452–465, 1995.

[BSS89] L. Blum, M. Shub, and S. Smale. On a theory of computation and complexity over the real numbers. *Bulletin of the Amer. Math. Soc.*, 21:1–46, 1989.

[BV96] D. Boneh and R. Venkatesan. Hardness of computing the most significant bits of secret keys in Diffie-Hellman and related schemes. In *Proc. CRYPTO 96*, 1996. to appear.

[BW88] J. Buchmann and H. Williams. A key exchange system based on imaginary quadratic fields. *J. of Cryptology*, 1(2):107–118, 1988.

[Car95] G. Caronni. Assuring ownership rights for digital images. In H.H. Brueggemann and W. Gerhardt-Haeckl, editors, *Proceedings of 'reliable IT systems' (verlaessliche IT-Systeme) VIS '95*, Germany, 1995. Vieweg Publishing Company.

[CEP83] E. Canfield, P. Erdös, and C. Pomerance. On a problem of Oppenheim concerning "Factorisatio Numerorum". *J. Number Theory*, 17:1–28, 1983.

[CFN94] B. Chor, A. Fiat, and M. Naor. Tracing traitors. In *Proc. CRYPTO 94*, pages 257–270, 1994.

[Cli37] A.H. Clifford. Representations Induced in an Invariant Subgroup. *Ann. of Math.*, 38(2):533–550, 1937.

[Coh91] H. Cohen. *A course in computational algebraic number theory.* Springer-Verlag, 1991.

[Dam88] I. Damgård. On the randomness of Legendre and Jacobi sequences. In *Proc. CRYPTO 88*, pages 163–172, 1988.

[Dav33] H. Davenport. On the distribution of quadratic residues (mod p). *J. London Math. Soc.*, pages 46–52, 1933.

[Dav80] H. Davenport. *Multiplicative number theory.* Springer-Verlag, 1980.

[dB88] B. den Boer. Diffie-Hellman is as strong as discrete log for certain primes. In *Proc. CRYPTO 88*, pages 530–539, 1988.

[DeB66] N. DeBruijn. On the number of positive integers $\leq x$ and free of prime factors $> y$. *Indag. Math.*, 38:239–247, 1966.

[DH76] W. Diffie and M. Hellman. New directions in cryptography. *IEEE Transactions on Information Theory*, 22(6):644–654, 1976.

[Dix70] J. Dixon. Computing Irreducible Representations of Groups. *Math. Comp.*, 24:707–712, 1970.

[DLS81] P. Downey, B. Leony, and R. Sethi. Computing sequences with addition chains. *SIAM J. Computing*, 3:638–696, 1981.

[Dor71] L. Dornhoff. *Group Representation Theory, Part A*. Mercel Dekker, 1971.

[EFF85] P. Erdös, P. Frankl, and Z. Füredi. Families of finite sets in which no set is covered by the union of r others. *Israel J. of math.*, pages 79–89, 1985.

[EG85] T. El-Gamal. A public key cryptosystem and a signature scheme based on the discrete logarithm. *IEEE Transactions on Information Theory*, 31(4):469–472, 1985.

[FHK*88] A. Frieze, J. Hastad, R. Kannan, J. Lagarias, and A. Shamir. Reconstructing Truncated Integer Variables Satisfying Linear Congruences. *SIAM J. Computing*, 17(2):262–280, 1988.

[FM89] J. Feigenbaum and N. Merrirr. Open Questions and summary of discussions. In *Proceedings of DIMACS workshop on Distributed Computing and Cryptography*, volume 2, pages 1–32, 1989.

[FPS95] E. Flynn, B. Poonen, and F. Schaefer. Cycles of quadratic polynomials and rational points on a genus 2 curve. *AMS preprint*, 1995.

[Gar79] J. Garlinksi. *Intercept: the enigma war*. Dent, 1979.

[GL89] O. Goldreich and L. Levin. A hard core predicate for all one-way functions. In *Proc. 21st ACM Symp. on Theory of Computing*, pages 25–32, 1989.

[Glo92] D. Glover. *The protection of computer software.* Cambridge University, 2nd ed. edition, 1992.

[Gol90] S. Goldwasser. The search for provably secure cryptosystems. In *Cryptology and computational number theory*, AMS Lecture notes, pages 89–114, 1990.

[Gol95] O. Goldreich. *Foundations of cryptography.* Electronic Colloquium on Computational Complexity, 1995. Fragments of a book.

[GRS95] O. Goldreich, R. Rubinfeld, and M. Sudan. Learning polynomials with queries: the highly noisy case. In *Proc. 36th IEEE Symp. on Foundations of Comp. Science*, pages 294–303, 1995.

[IN89] R. Impagliazzo and M. Naor. Efficient cryptographic schemes provably as secure as subset sum. In *Proc. 30th IEEE Symp. on Foundations of Comp. Science*, pages 236–241, 1989.

[Jac94] J. Jackson. An efficient membership-query algorithm for learning DNF with respect to the uniform distribution. In *Proc. 35th IEEE Symp. on Foundations of Comp. Science*, pages 42–53, 1994.

[Kah67] D. Kahn. *The Codebreakers.* Macmillan, 1967.

[Kah91] D. Kahn. *Seizing the enigma : the race to break the German U-boat codes, 1939-1943.* Houghton Mifflin Co., 1991.

[KM91] E. Kushilevitz and Y. Mansour. Learning decision trees using the Fourier spectrum. In *Proc. 23rd ACM Symp. on Theory of Computing*, pages 455–464, 1991.

[Knu81] D. Knuth. *The art of computer programming*, volume 2. Addison Wesley, 1981.

[Kob87a] N. Koblitz. *A course in number theory and cryptography.* Springer-Verlag, 1987.

[Kob87b] N. Koblitz. Elliptic curve cryptosystems. *Math. Comp.*, 48:203–209, 1987.

[Kob88] N. Koblitz. A family of Jacobians suitable for discrete log cryptosystems. In *Proc. CRYPTO 88*, pages 94–99, 1988.

[Len87] H. W. Lenstra. Factoring integers with elliptic curves. *Annals of Math.*, 126:649–673, 1987.

[Lip94] R. Lipton. Straight line complexity and integer factorization. In *First Algorithmic Number Theory Symposium*, 1994.

[LL90] A. Lenstra and H.W. Lenstra. Algorithms in Number Theory. In *Handbook of Theoretical Computer Science (Volume A: Algorithms and Complexity)*, chapter 12, pages 673–715. Elsevier and MIT Press, 1990.

[LL94] A. Lenstra and H.W. Lenstra, editors. *The development of the number field sieve*, volume 1554 of *Lecture Notes in Mathematics.* Springer-Verlag, 1994.

[LLL82] A. Lenstra, H. Lenstra, and L. Lovasz. Factoring polynomial with rational coefficients. *Mathematiche Annalen*, 261:515–534, 1982.

[LMN89] N. Linial, Y. Mansour, and N. Nisan. Constant depth circuits, Fourier transform, and learnability. In *Proc. 30th IEEE Symp. on Foundations of Comp. Science*, pages 574–579, 1989.

[LO77] J. Lagarias and A. Odlyzko. Effective versions of the Chebotarev density theorem. In A. Frohlich, editor, *Algebraic Number Fields*, pages 409–464. Academic Press, 1977.

[Man92] Y. Mansour. An $O(n^{\log\log n})$ Learning Algorithm for DNF under the uniform distribution. In *Proc. 5th Annu. Conf. on Comp. Learning Theory*, pages 53–59, 1992.

[Mar77] D. Marcus. *Number fields*. Springer-Verlag, 1977.

[Mau94] U. Maurer. Towards the equivalence of breaking the Diffie-Hellman protocol and computing discrete logarithms. In *Proc. CRYPTO 94*, pages 271–281, 1994.

[McC88] K. McCurley. A key distribution system equivalent to factoring. *J. of Cryptology*, 1(2):95–105, 1988.

[Mil85] V. Miller. Use of elliptic curves in cryptography. In *Proc. CRYPTO 85*, pages 417–426, 1985.

[MS96] W. De Melo and B. Svaiter. The cost of computing integers. *Proc. Amer. Math. Soc.*, 1996. to appear.

[MV91] A. Menezes and S. Vanstone. Reducing elliptic curve logarithms to logarithms in a finite field. In *Proc. 23rd ACM Symp. on Theory of Computing*, pages 80–89, 1991.

[MY91] U. Maurer and Y. Yacobi. Non-interactive public-key cryptography. In *Proc. EUROCRYPT 91*, pages 498–507, 1991.

[Nat91] National Institute for Standards and Technology. *Specifications for the digital signature standard*, August 1991. Publication XX, draft.

[Nec94] V. Nechaev. Complexity of a determinate algorithm for the discrete logarithm. *Mathematical Notes*, 55(2):165–172, 1994.

[Neu86] J. Neukirch. *Class field theory*. Springer-Verlag, 1986.

[Nor50] D. Northcott. Periodic points on an algebraic variety. *Annals of Math.*, 51:167–177, 1950.

[Oka88] T. Okamoto. *Encryption and Authentication Schemes Based on Public Key Systems*. PhD thesis, Univ. of Tokyo, 1988.

[PH78] S. Pohlig and M. Hellman. An improved algorithm for computing discrete logarithms over $GF(p)$ and its cryptographic significance. *IEEE Transactions on Information Theory*, 24:106–110, 1978.

[Riv90] R. Rivest. Cryptography. In *Handbook of theoretical computer science*, volume A. Elsevier, 1990.

[RSA78] R. Rivest, A. Shamir, and L. Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Communications of the ACM*, 21:120–126, 1978.

[RT88] R. Read and W. Tutte. Chromatic polynomials. In *Selected topics in graphs theory*. Academic press, 1988.

[Sch85] R. Schoof. Elliptic Curves Over Finite Fields and the Computation of Square Roots mod p. *Math. Comp.*, 44(170):483–494, 1985.

[Sch87] C. Schnorr. A hierarchy of polynomial time lattice basis reduction algorithms. *Theoretical computer science*, 53:201–224, 1987.

[Ser77] J.P. Serre. *Linear representations of finite groups*. Springer-Verlag, 1977.

[Sho95] V. Shoup. Lower bounds for discrete logarithms and related problems. Manuscript, 1995.

[Sil86] J. Silverman. *The arithmetic of elliptic curves*. Springer-Verlag, 1986.

[Sil92] A. Silverberg. Points of finite order on Abelian varieties. *Contemporary math.*, 133:175–193, 1992.

[SS95] K. Sakurai and H Shizuya. Relationships among the computational powers of breaking discrete log cryptosystems. In *Proc. EUROCRYPT 95*, pages 341–355, May 1995.

[Str90] V. Strassen. Algberaic complexity theory. In *Handbook of theoretical computer science*, volume A. Elsevier, 1990.

[TNM90] K. Tanaka, Y. Nakamura, and K. Matsui. Embedding secret information into a dithered multi-level image. In *Proc. IEEE Military Communications Conference*, pages 216–220, September 1990.

[vL82] van Lint. *Introduction to coding theory.* Springer-Verlag, 1982.

[Wag83] N. Wagner. Fingerprinting. In *Proc. IEEE Symposium on Security and Privacy*, pages 18–22, April 1983.

[Wei84] A. Weil. *Number theory : an approach through history from Hammurabi to Legendre.* Birkhauser, 1984.

[Wol95] S. Wolf. Diffie-Hellman and Discrete Logarithms. Master's thesis, ETH Zurich, 1995.