SOME RESULTS ON

Cryptanalysis of RSA and Factorization

A thesis presented to the Indian Statistical Institute
in fulfillment of the thesis requirement for the degree of
Doctor of Philosophy in Mathematics

by

Santanu Sarkar

Applied Statistics Unit
INDIAN STATISTICAL INSTITUTE
Kolkata, West Bengal, India, 2011

SOME RESULTS ON

Cryptanalysis of RSA and Factorization

A thesis presented to the Indian Statistical Institute
in fulfillment of the thesis requirement for the degree of
Doctor of Philosophy in Mathematics

by

Santanu Sarkar

under the supervision of

Professor Subhamoy Maitra

Applied Statistics Unit
INDIAN STATISTICAL INSTITUTE
Kolkata, West Bengal, India, 2011

Abstract

In this thesis, we propose some new results in Cryptanalysis of RSA and related Factorization problems. Till date, the best known algorithm to solve the Integer Factorization problem is the Number Field Sieve, which has a runtime greater than $\exp(\log^{1/3} N)$ for factoring an integer N. However, if one obtains certain information about the RSA parameters, there are algorithms which can factor the RSA modulus $N = pq$ quite efficiently. The intention of this thesis is to identify such weaknesses of the RSA cryptosystem and its variants. Further we study results related to factorization.

In Africacrypt 2008, Nitaj presented a class of weak keys in RSA considering certain properties of the encryption exponent e. We show that this result can be generalized from different aspects. We consider the cases when e satisfies an equation of the form $eX - \psi Y = 1$ under some specific constraints on two integers X, Y and a function ψ. Using the idea of Boneh and Durfee (Eurocrypt 1999, IEEE-IT 2000), we show that the LLL algorithm can be efficiently applied to get ψ in cases where Y satisfies certain bounds. This idea extends the class of weak keys presented by Nitaj when ψ is of the form $(p - u)(q - v)$ for RSA primes p, q and integers u, v. Further, we consider the form $\psi = N - pu - v$ for integers u, v to present a new class of weak keys in RSA. This idea does not require any kind of factorization as used in Nitaj's work.

Next, we analyze the security of RSA where multiple encryption are available for the same modulus N. We show that if n many corresponding decryption exponents $(d_1, \ldots, d_n)$ are generated, then RSA is insecure when $d_i < N^{\frac{3n-1}{4n+4}}$, for all i, $1 \leq i \leq n$ and $n \geq 2$. Our result improves the bound of Howgrave-Graham and Seifert (CQRE 1999).

We also discuss the factorization of N by reconstructing the primes from randomly known bits. We revisit the work of Heninger and Shacham (Crypto 2009) and provide a combinatorial model for the reconstruction where some random bits of the primes are known. This shows how one can factorize N given the knowledge of random bits in the least significant halves of the primes. We also explain a lattice based strategy in this direction. More importantly, we study how N can be factored given the knowledge of some blocks of bits in the most significant halves of the primes. We present improved theoretical result and experimental evidences in this direction.

In PKC 2009, May and Ritzenhofen presented interesting problems related to factoring large integers with some implicit hints. One of the problems considers $N_1 = p_1q_1$ and $N_2 = p_2q_2$, where p_1, p_2, q_1, q_2 are large primes, and the primes p_1, p_2 are of same bitsize such that certain amount of Least Significant Bits (LSBs) of p_1, p_2 are same. May and Ritzenhofen proposed a strategy to factorize both N_1, N_2 efficiently with the implicit information that p_1, p_2 share certain amount of LSBs. We explore the same problem with a different lattice-based strategy. In a general framework, our method works when implicit information is available related to Least Significant as well as Most Significant Bits (MSBs). We show that one can factor N_1, N_2 (simultaneously) efficiently when p_1, p_2 share certain amount of MSBs and/or LSBs. We also solve the implicit factorization problem given three RSA moduli $N_1 = p_1q_1, N_2 = p_2q_2, N_3 = p_3q_3$, when p_1, p_2, p_3 share certain portion of LSBs as well as certain portion of MSBs. Furthermore, we study the case when p_1, p_2 share some bits in the middle. Our strategy presents new and encouraging results in this direction. Moreover, some of the observations by May and Ritzenhofen get improved when we apply our ideas for the LSB case.

In CaLC 2001, Howgrave-Graham proposed a method to find the Greatest Common Divisor (GCD) of two large integers when one of the integers is exactly known and the other one is known approximately. We present two applications of the technique. The first one is to show deterministic polynomial time equivalence between factoring $N = pq$ and knowledge of $q^{-1} \bmod p$. As the second application, we consider the problem of finding smooth integers in a short interval. Next, we analyze how to calculate the GCD of k (≥ 2) many large integers, given their approximations. Two versions of the existing approximate common divisor problem are special cases of our analysis when $k = 2$. Further, we relate the approximate common divisor problem to the implicit factorization problem. Our strategy can be applied to the implicit factorization problem in a general framework considering the equality of (i) Most Significant Bits (MSBs), (ii) Least Significant Bits (LSBs) and (iii) MSBs and LSBs together. We present new and improved theoretical as well as experimental results in comparison with the state of the art works in this area.

Acknowledgements

There are many people who helped in different ways to make this thesis possible. I would like to thank them all for their suggestions and advice throughout the tenure of my Doctoral studies.

I am extremely lucky to have worked with my supervisor Prof. Subhamoy Maitra (our beloved Subhamoy-da). I do not want to belittle his contributions towards my Doctoral research by thanking him. But, I would definitely like to take this opportunity to express my gratitude for his warm guidance, patience and tolerance during my Ph.D. course. I am indebted to him for all kinds of support he has provided me throughout the last three years.

It is my honor to have worked as a part of the Cryptology Research Group at ISI Kolkata, founded and guided by Prof. Bimal Roy. I would like to express my heartfelt gratitude towards Prof. Roy, who always supported me for my research and created opportunities for me to visit different places of the world to interact with eminent researchers in the field. These interactions have immensely helped me in broadening my horizon of knowledge.

I would also like to thank Prof. Rana Barua for his great teaching on Cryptography that motivated me to initiate my research work in this field. I am grateful to Prof. Palash Sarkar for his numerous invaluable advice during my research. Further, it is my pleasure to thank Prof. Arup Bose and Dr. Amartya Dutta for their teaching during my course work, and Prof. Goutam Mukherjee for allowing me to use my office-space at the Stat-Math Unit, ISI Kolkata.

I am grateful to Mr. Sourav Sen Gupta, who helped me whenever I faced any problem. We have detailed discussions on several technical issues and in particular, I acknowledge his contribution in making substantial improvement towards the editorial quality of my thesis.

I would especially like to thank Mr. Sumit Kumar Pandey, who have directly had a positive impact on my interest in mathematics. I would like to express my warm gratitude to Kishan-da, Venku-da, Mahavir-da, Somitra-da, Dalai-da, Prem-da, Sumanta-da, Tanmoy-da, Sanjoy-da, Sushmita-di, Pinaki-da, Srimanta-da, Biswarup, Rajat, Koushik, Ramij, Soumen, Debasis, Debabrata, Rishi, Goutam, Sanjay, Butu and Somindu for being there beside me at times when I needed them the most. My special thanks to each member of ASU, SMU, RS Hostel, Deans Office and Cash Section of Indian Statistical Institute, Kolkata, for all the help and support they provided throughout my stay at the Institute.

I would like to thank my Didi (elder sister) for her love and support. Last but not the least, I am grateful to my Parents for their understanding, never ending blessings, faith and support in every step of my life. I love you Maa and Baba!

To Thaakuma (Grandmother), the first teacher in my life.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

The word ‘Cryptology’ originates from the Greek root words ‘kryptós’ (hidden) and ‘logia’ (study). So, quite literally, cryptology is the study and practice of hiding information, that is, the art and science of information security. It is quite natural that the notion of security requires the presence of an adversary, and any system must be tested against all potential attacks. Based on this idea, cryptology has been branched into two parts - Cryptography and Cryptanalysis. While cryptography ensures secure communication between two parties over insecure communication channels (telephone, courier, fax, e-mail etc.), the field of cryptanalysis deals with the study of breaking a cryptographic scheme by obtaining the classified information without authorization.

Cryptology was used vastly in ancient civilizations throughout the world. Around 3000 BC, the Egyptians could communicate securely by writing messages in the coded Hieroglyph. ‘Artha-Sastra’ by the renowned Indian politician Kautilya (350–283 BC) discussed the use of cryptology in political circles. Julius Caesar (100–44 BC) used the Caesar cipher to send confidential messages.

The subject resurrected and evolved in a completely new form at the dawn of the modern information era. During World War II, cryptology gained much more importance in political affairs. It was the famous episode of Enigma that made the world aware of the potential power of the subject. Today, cryptology has earned its repute as a prime branch of modern science that deals with information security issues in the arena of digital globalization. It is used for numerous purposes of everyday life, including Internet banking, identification cards, secure databases, e-mails, social networking, and many more.

The motive of Cryptography, as it stands in the modern society, has broadened to cover not only encryption and decryption, but all of the following categories.

Confidentiality deals with the process of encryption and decryption of data. This is the basic goal of cryptography that ensures secure data communication over insecure channels, possibly at the presence of a malicious adversary.

Data Integrity is the mechanism to check and make sure that the data has not been tampered. This process is important to verify the consistency of the data and make sure no alterations have been made to the original communication.

Authentication ensures proper identification of the sender (entity authentication), and the identification of the origin of communicated data (data origin authentication). This feature is required to ensure that data is accepted only from the authorized senders.

Non-Repudiation is another important aspect of cryptography to solve disputes between the sender and the receiver. It may happen that the sender, after communicating some data, maliciously claims that he/she has not sent it at all. Non-repudiation provides the receiver with a proof to refute this claim. The proof can also be presented to a third-party arbitrator in such a case.

In this thesis, we mainly concentrate on the Confidentiality aspect of cryptosystems. For further details about modern day Cryptology, one may refer to "Cryptography - Theory and Practice" by D. R. Stinson [126], or "Handbook of Applied Cryptography" by A. J. Menezes, P. C. van Oorschot and S. A. Vanstone [87]. An interested reader may also refer to "The Code Book" by S. Singh [122] to know the exciting history of evolution of the subject.

1.1 Encryption and Decryption

Let us turn our attention to the functional aspect of a cryptographic scheme for confidentiality. In this section, we shall introduce the notions of encryption, decryption and the respective keys associated with these operations.

Suppose Alice wants to send some plaintext M to Bob, and Charles is a malicious adversary who is eavesdropping on the insecure channel of communication. The scenario is as shown in Figure 1.1. Using some encryption function E, Alice encrypts the plaintext M to get the ciphertext $C = E(M)$ and sends C to Bob. When Bob receives C, using his decryption function D, he decrypts the ciphertext C to get back the plaintext $M = D(C)$.

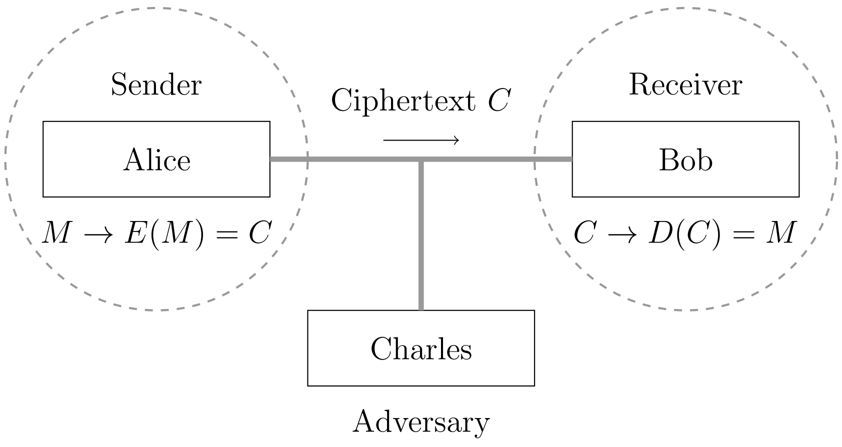

Figure 1.1: Communication over an Insecure Channel.

Intuitively, one must have the decryption function D to be the inverse of the encryption function E. Kerckhoffs' law [87] in cryptography mandates that the security of a cryptosystem should not depend on the secrecy of either algorithm E or D. The security should only depend on some secret parameter(s) used in the process of encryption and/or decryption. This secret parameter is called the *Key* of the system in cryptographic terms.

Thus, in a secure cryptographic protocol, Alice will encrypt plaintext M using the modified encryption function $C = E(e_k, M)$, which is dependent on the encryption key e_k. Likewise, Bob will use the modified decryption function $M = D(d_k, C)$, dependent on the decryption key d_k. For correct encryption and decryption using these key-dependent functions, some mathematical relation between e_k and d_k must exist in such a system. Depending on this relation between the encryption key and the decryption key, the study of modern cryptography can be classified under two major heads, namely Symmetric Key Cryptography and Asymmetric Key Cryptography.

1.2 Symmetric Key Cryptography

In symmetric key cryptography, the encryption key is trivially related to the decryption key. So either the encryption key e_k is same as the decryption key d_k, or there is a simple transformation between them. These type of cryptosystems are also called 'secret key' cryptosystems, as the keys are kept hidden from unauthorized users.

There are two major types of symmetric key cryptosystems, namely Block Ciphers and Stream Ciphers. We can broadly characterize them as follows.

Block Ciphers. The goal of block ciphers is to scramble the plaintext block-by-block using the basic tools of confusion and diffusion applied to the plaintext over multiple rounds. A generic setup for a block cipher would use complex substitution and permutation rounds on the plaintext block, using a secret key, to create the effect of confusion and diffusion.

Stream Ciphers. On the other hand, the inherent goal of a stream cipher is to generate a pseudo-random stream of data using a randomly chosen key of fixed (preferably short) length. After the generation of such a pseudo-random stream, the plaintext is simply XOR-ed with the stream to obtain the ciphertext.

One may say that the security of a block cipher depends upon the amount of confusion and diffusion created over rounds, whereas that of a stream cipher depends upon the indistinguishability of its output stream from an actual random stream of bytes. The reader may refer to [126] for further details.

One of the most well known Block Ciphers at present is the *Advanced Encryption Standard* (AES) [1]. Before this, the *Data Encryption Standard* (DES) [32] was the most popular one. RC4 [108], SNOW [34], TURING [111] etc. are some of the popular stream ciphers. One may refer to the eStream project [39] for recent developments in the area of stream cipher design. Although symmetric key cryptosystems are very fast in practice, there are few drawbacks, as follows.

Key distribution problem: A secure and authenticated secret channel should be needed to distribute the secret keys beforehand.

Key management problem: In a network of n users, every pair of users must share a secret key, for a total of $\binom{n}{2} = \frac{n(n-1)}{2}$ keys. If n is large, then the number of keys becomes unmanageable.

Signature problem: Consider the situation where Alice sends a message (encrypted/signed using some symmetric key cipher) to Bob, and later refuses this communication. Now, in case of such a dispute, Bob will have to prove to an unbiased arbitrator (may be the judge or jury at the court) that the message was in fact sent by Alice. Now, the arbitrator can not verify who actually encrypted/signed that particular plaintext, as both Alice and Bob are capable of encrypting/signing it using the same (shared) key. This problem of authentication is known as repudiation. This problem can not be solved using symmetric key systems, as the concept of non-repudiation using signatures can not be accomplished.

To overcome these problems, Diffie and Hellman [33] introduced the notion of 'asymmetric key' cryptosystems in 1976. Invention of asymmetric key cryptosystems is arguably the most celebrated breakthrough in modern day cryptography. However, one should note that public key cryptosystems are much slower than symmetric key cryptosystems in general.

1.3 Asymmetric Key Cryptography

In asymmetric key cryptography, the encryption and the decryption keys are different, but they are related by some mathematical relation. These type of cryptosystems are also called 'public key' cryptosystems, as the encryption key is made public while the decryption key is kept secret. Hence, it is required for the security of the system that, finding the decryption key from the encryption key is computationally infeasible.

The public key cryptosystems are most often based on some computationally hard mathematical problem. A list of well known hard problems is as follows.

Integer Factorization Problem (IFP). The problem is to find a proper factor of a given positive integer $N > 1$. The first usable public key cryptosystem RSA [110] is based on the hardness of integer factorization problem. Rabin cryptosystem [104] is also based on integer factorization problem.

Quadratic Residue Problem (QRP). The problem is to decide whether an element $x \in \mathbb{Z}_N$ has a modular square root where N is a composite number. It can be proved that if factorization of N is known then QRP in $\mathbb{Z}_N$ is no longer hard. Goldwasser-Micali [45] cryptosystem is based on QRP.

Discrete Logarithm Problem (DLP). Let G be a large cyclic group with generator g. Then given any element $y = g^a$ in G, the problem of finding the exponent a is called DLP over G. El Gamal system [35, 36] is a public key cryptosystem based on DLP. In an elliptic curve group over a finite field, a similar problem of ECDLP can be formulated using the additive structure of the group. Based on the hardness of ECDLP and ECDHP (elliptic curve Diffie-Hellman problem), the notion of Elliptic Curve Cryptography was proposed independently by Koblitz [71] and Miller [91].

Knapsack Problem. Given a set of n positive integers $\{k_i\}$ and a positive integer N, the problem is to find whether it is possible to represent $N = \sum_{i=1}^{n} a_i k_i$ where each a_i is either 0 or 1. This is also called the subset sum problem. Many cryptosystems like Merkle-Hellman cryptosystem [88] were proposed based on this subset sum problem, and most of them have been broken. One may refer to [117, 118] for an account on such attacks.

Shortest Vector Problem (SVP). The problem is to find the shortest nonzero vector in a high dimensional lattice. This is hard in general and a few cryptosystems like NTRU [58], Ajtai-Dwork system [4, 5] are based on this problem. It is worth noting that neither IPF nor DLP is hard under the quantum computation model, but SVP continues to remain hard in the quantum era.

1.4 Goal of this Thesis

The main goal of this thesis is Cryptanalysis of RSA modulus $N = pq$ and related Factorization problems. It is still unknown whether there is an efficient (polynomial time) algorithm to solve the 'Integer Factorization Problem (IFP)' in the classical model. The best known algorithm to solve this problem is the Number Field Sieve (NFS) [76], which has runtime greater than $\exp(\log^{1/3} N)$. However, if one obtains certain information about the RSA parameters, there are algorithms which can factor N quite efficiently. Our intention is to identify such weaknesses of the RSA cryptosystem and also to look into certain versions of factorization problem (in this thesis, the implicit factorization problem) that can be solved efficiently.

1.5 Organization of the Thesis

The thesis presents several cryptanalytic results against the RSA cryptosystem when parts of the secret key are known or known in disguise. Most of our results use the LLL algorithm to obtain the secret key, and the main idea for each application is how to phrase the problem as a lattice problem and how to compute the dimension and determinant of the lattice. Lattice based root-finding ideas for polynomials is the theme for most of the results presented here. Moreover, all the results and discussions are targeted towards finding weaknesses in the RSA cryptosystem and its implementation. There are several attempts at RSA cryptanalysis in the same vein, and this thesis also illustrates how these results improve on the previous state of the art.

It is recommended that one reads the chapters in the order they are presented. However, the reader may choose to browse quickly to a chapter of his/her choice and refer back to Chapter 2 for the mathematical background. A short summary for each chapter is presented as follows.

Chapter 1: In the current chapter, we have discussed some introductory materials regarding cryptography, and its major classifications. We also present the goal and structure of this thesis.

Chapter 2: In the next chapter, we start with some basic mathematical definitions and an overview of the RSA cryptosystem. In section 2.5, we introduce lattice based root finding techniques for modular polynomials. In section 2.6, we discuss the approach to find roots of a polynomial over integers. We will use these root finding techniques frequently in the thesis.

Chapter 3: In this chapter, we discuss our work to identify encryption exponents for which RSA becomes weak. The materials of this chapter are based on our publication [81].

Chapter 4: Here we study the vulnerabilities of RSA in terms of its decryption exponent. We go through the model of RSA in the presence of many decryption exponents, and discuss our work regarding cryptanalysis of RSA within this model. The materials of this chapter are based on our publications [114, 115].

Chapter 5: In addition to the weaknesses of RSA due to weak keys, it may also be vulnerable due to leaked information about the RSA primes. In this chapter, we discuss the factorization of the RSA modulus N by reconstructing the primes from randomly known bits. In Sections 5.1 and 5.2, we analyze the fact that $N = pq$ can be factored in reasonable time complexity when a few bits of the primes are known from the least significant halves. In Section 5.3, we analyze the same when a few bits are known from the most significant halves of the primes. The materials of this chapter are based on our publication [82].

Chapter 6: At times, one may not obtain explicit bitwise information about the primes, but have some implicit knowledge regarding those. An attempt at RSA factorization based on this implicit knowledge is of interest as well. This chapter deals with the analysis of a situation when one can factor RSA moduli $N_1 = p_1q_1, N_2 = p_2q_2, \ldots, N_k = p_kq_k$ in polynomial time if $p_1, p_2, \ldots, p_k$ share a few bits. This problem is called the implicit factorization problem, introduced by May and Ritzenhofen in [86]. In Section 6.1, we present the implicit factorization strategy for two or three large integers when they share MSBs and/or LSBs. Next, in Section 6.2, we analyze the same problem when the primes p_1, p_2 share a (contiguous) portion of bits at the middle. The materials of this chapter are based on our publications [113, 116].

Chapter 7: In this chapter, we present two generalizations, Extended Partially Approximate Common Divisor Problem (EPACDP) and Extended General Approximate Common Divisor Problem (EGACDP), of the 'approximate common divisor problem' introduced by Howgrave-Graham [61]. We also propose two applications of 'approximate common divisor problem'. In Sections 7.4 and 7.6, we propose two methods to solve EPACDP, and in Section 7.7, we discuss the solution of EGACDP. Most importantly, continuing from Chapter 6, we discuss the applications of EPACDP for implicit factorization when $p_1, p_2, \ldots, p_k$ share some MSBs and or LSBs. The materials of this chapter are based on our publications [112,116].

Chapter 8: This chapter concludes the thesis. Here we present a comprehensive summary of our work that has been discussed throughout the thesis. We analyze and compare our work with the contemporary advances in the field of cryptography and also discuss open problems which might be interesting for further investigation along this line of research.

1.6 Prerequisites

Public Key Cryptography is a highly mathematical endeavor. RSA itself is based on the integer factorization problem, which has baffled the mathematicians for ages. To understand the intricate details of RSA and lattice-based results proposed in this thesis, one requires a strong foundation in Mathematics. We frequently use involved results of number theory, linear algebra and probability in this thesis, and expect the reader to possess a good grasp on these topics.

Although we present a comprehensive overview of all necessary mathematical preliminaries in Chapter 2, a graduate level training in mathematics is recommended to read the material comfortably.

Chapter 2

Mathematical Preliminaries

This chapter is dedicated to provide the reader with a comprehensive overview of the mathematical framework one may need to read this thesis. Due to the requirement for complexity analysis of the algorithms we discuss in this thesis, let us start with a brief overview of the basic asymptotic notations in Section 2.1. In Section 2.2, we discuss the RSA cryptosystem in detail and also describe the different variants of RSA used in practice. Next, we move on to our basic focus, cryptanalysis of RSA, in Section 2.3 and present a comprehensive summary of attacks on the RSA cryptosystem proposed in the last few decades. Most of the works in this thesis, as well as many partial key exposure attacks on RSA depend on lattice based polynomial solving techniques. Thus we study the basic properties of lattices in Section 2.4, and discuss the existing techniques to solve modular and integer polynomials in Sections 2.5 and 2.6 respectively.

2.1 Asymptotic Notation

In mathematics, computer science, and other fields related to computation, the time and space requirement for an algorithm to run is generally represented as a function of the size of the input(s). In this context, the asymptotic notations are used to describe the limiting behavior of these functions, in an asymptotic sense, as the input size tends to a specific value or towards infinity. Bachmann-Landau notations [27] are a family of asymptotic notations used to compare computational complexity (time or space) of algorithms. This family comprises of notations 'Big O' (O), 'Small o' (o), 'Big Omega' (Ω), 'Big Theta' (Θ) and 'Small Omega' (ω), and we shall use the first two in this thesis.

Definition 2.1 (Small o Notation)**.** Given two functions $f(n)$ and $g(n)$, we say that $f(n) \in o(g(n))$ if, for any constant $c > 0$, there exists a constant $N > 0$ such that $0 \leq f(n) < cg(n)$ for all $n \geq N$.

Definition 2.2 (Big O Notation)**.** Given two functions $f(n)$ and $g(n)$, we say that $f(n) \in O(g(n))$ if there exist constants $c > 0$ and $N > 0$ such that $0 \leq f(n) \leq cg(n)$ for all $n \geq N$.

In other words, we say that $f(n) \in o(g(n))$ if $\limsup \frac{f(n)}{g(n)} = 0$, and $f(n) \in O(g(n))$ if $\limsup \frac{f(n)}{g(n)}$ is some finite constant. For example, $n^2 + 1 \in o(n^3)$, and $n^2 + 1 \in O(n^2)$, where n is a positive integer. But $n^2 + 1 \notin o(n^2)$, as $\lim_{n\to\infty} \frac{n^2+1}{n^2} = 1$. From the discussion so far, one may observe that if $f(n) \in O(g(n))$, then f is asymptotically *bounded above* by g, up to a constant factor. However, $f(n) \in o(g(n))$ indicates that f is asymptotically *dominated* by g. Thus, the Small o is a stricter condition compared to Big O. In formal terms, we have $o(g(n)) \subseteq O(g(n))$. One may refer to [27] for further technical details regarding asymptotic notations.

Throughout this thesis, we shall denote the bitsize of an integer N by l_N and it is defined as the number of bits in N. That is, $l_N = \lceil \log_2 N \rceil$ when N is not a power of 2 and $l_N = \log_2 N + 1$, when N is a power of 2. We say that an algorithm $\mathcal{A}$ is a *polynomial time* algorithm if its running time is polynomial in the bitsize of its input. For an example, consider the algorithm for schoolbook multiplication. It is a polynomial time algorithm, because for any two integer inputs a, b, we can find $a \times b$ in time $O(\log a \cdot \log b)$, which is polynomial (quadratic) in the combined input bitsize of $\log a + \log b$.

2.2 RSA Cryptosystem

RSA was designed by R. Rivest, A. Shamir and L. Adleman in 1977, when they were at the Massachusetts Institute of Technology (MIT). It was published in 1978 and over the last three decades, RSA has become the most popular public key cryptosystem. One of the main reasons behind the soaring popularity of RSA is the simplicity of its theory and computations. It is widely used in modern electronic commerce protocols and is believed to be secure provided the parameters are implemented properly. In 2002, all three inventors of RSA received the Turing Award for their ingenious contribution for making public-key cryptography useful in practice. Let us systematically study the RSA cryptosystem before proceeding any further.

2.2.1 Classical Model of RSA

In a public key cryptosystem, there exists three major components, namely the key generation process, and the algorithms for encryption and decryption. Key generation and decryption are performed by the recipient, whereas the encryption occurs on the side of the sender. In case of RSA, the three phases can be described as follows. We shall henceforth assume that Alice is the sender and Bob is the receiver in our cryptographic scheme.

Key Generation

To create a public/private key pair for the RSA cryptosystem, Bob first chooses randomly two 'large' primes p, q (recommended to use primes of same bitsize with minimum size of 512 each). Then he calculates the product $N = pq$ and Euler's totient function $\phi(N) = (p-1)(q-1)$. Bob keeps the values $p, q, \phi(N)$ secret, as any one who knows any one of these values will be able to decrypt messages sent to Bob.

Bob's next step is to find two positive integers e, d such that $ed \equiv 1 \pmod{\phi(N)}$. Bob publishes the pair (e, N) as his public key, which can be used to encrypt messages meant for Bob. The pair (d, N) is Bob's secret key and these are used to decrypt the received ciphertexts.

Encryption

To send a plaintext to Bob, Alice first transforms her message in to an element m of $\mathbb{Z}_N$, and calculates $c \equiv m^e \pmod{N}$. The ciphertext c is sent to Bob.

Decryption

After receiving the ciphertext c, Bob decrypts c by computing $c^d \bmod N$ and gets back m.

Correctness of RSA algorithm

Here we present the proof of the correctness of the RSA algorithm, that is $c^d \bmod N = m$. We know that $ed \equiv 1 \pmod{\phi(N)}$. So, we can write $ed = 1 + k\phi(N)$ for some integer k. Let us first assume that $m \in \mathbb{Z}_N^*$ (integers less than and co-prime to N). Then,

$$\begin{aligned} c^d &\equiv m^{ed} \pmod N \\ &\equiv m^{1+k\phi(N)} \pmod N \\ &\equiv m \cdot (m^{\phi(N)})^k \pmod N \\ &\equiv m \pmod N \qquad (\text{as } m^{\phi(N)} \equiv 1 \pmod N \text{ by Euler's Theorem [126]}). \end{aligned}$$

Now assume that $m \in \mathbb{Z}_N \setminus \mathbb{Z}_N^*$, that is $\gcd(m, N) > 1$. If $m \equiv 0 \pmod p$ and $m \equiv 0 \pmod q$, then $m \equiv 0 \pmod N$. Then $c^d \equiv m^{ed} \equiv m \equiv 0 \pmod N$. On the other hand, let us assume without loss of generality that $m \equiv 0 \pmod p$ and $m \not\equiv 0 \pmod q$. Then $c^d \equiv m^{ed} \equiv m \equiv 0 \pmod p$. So, p divides $m^{ed} - m$. We also have

$$\begin{aligned} m^{ed} &\equiv m^{1+k\phi(N)} \pmod q \\ &\equiv m \cdot m^{k(p-1)(q-1)} \pmod q \\ &\equiv m \cdot \left(m^{q-1}\right)^{k(p-1)} \pmod q \\ &\equiv m \pmod q \qquad \text{as } m^{q-1} \equiv 1 \pmod q. \end{aligned}$$

So, q divides $m^{ed} - m$. Hence, $pq = N$ divides $m^{ed} - m$ i.e., $m^{ed} \equiv m \pmod N$.

Example of RSA Cryptosystem

Let us present a toy example to illustrate the basic operations.

Example 2.3. Bob chooses two primes $p = 653, q = 877$, and calculates $N = pq = 572681$, $\phi(N) = (p-1)(q-1) = 571152$. Suppose that Bob picks an integer $e = 13$ as the encryption exponent. Now he has to find the decryption exponent d which is e^{-1} in $\mathbb{Z}_{\phi(N)}$. One can check that $13 \times 395413 \equiv 1 \pmod{571152}$. Hence, the RSA parameters for Bob are

- public key: $(13, 572681)$, and

- private key: $(395413, 572681)$.

To encrypt a plaintext $m = 12345$, Alice uses Bob's public key $(13, 572681)$, and calculates $c = 12345^{13} \bmod 572681 = 536754$ and sends c to Bob. To decrypt $c = 536754$, Bob calculates $536754^{395413} \bmod 572681 = 12345 = m$.

However, one may note that if an adversary can factor N, then he/she can find d from the public key (e, N). In Example 2.3, it is not very hard to factor $N = 572681$ into its prime factors. In practice though, the bitsize of N is taken large enough so that it can not be factored efficiently using the current computation power. It is recommended to use N of bitsize 1024 or more to prevent factorization. Security issues of RSA will be discussed in more details later in this chapter.

Example 2.4. In a practical scenario, the RSA parameters will look as follows.
$p =$ 846599862936164736402988177812099956013778770876315707836731563770
5880893839981848305923857095440391598629588811166856664047346930517527
891174871536167839,
$q =$ 1217643468620406884679731818277104033968965197246189229334942736503
0339100965821711975719883742949180031386696753968921229679623132353468
174200136260738213,
$N =$ 10308567936391526757875542896033316178883861174865735387244345263
7137208314161521669308869345882336991188745907630491004512656603926295
3518502967942206721243236328408403417100233192004322468033366480788753
9303481101449158308722791555032457532325542013658355061619621556208246
3591629130621212947471071208931707,
$e = 2^{16} + 1 =$ 65537, and
$d =$ 101956309423526004076893177133219940094766772585504692321252302615
1120238295258506352584280960487541607315458593878388760777253827593350
0788233193317652234750616708162985718345962209115090210535366860135950
1135207708372912478251719497009548072271475262211661830196811724409660
406447291034092315494830924578345.
The public key will be (e, N) and the private key (d, N), as usual.

2.2.2 Implementation of RSA

The practical implementation of the RSA algorithm follows the structure as shown in Algorithm 1. To discuss the implementation cost of RSA in terms of time and space complexity, one may need to refer back to the basic definitions of asymptotic notations at the beginning of this chapter.

```
1 Generate randomly two large primes $p, q$;
2 Compute $N = pq, \phi(N) = (p-1)(q-1)$;
3 Generate $e$ randomly such that $\gcd(e, \phi(N)) = 1$;
4 Compute $d$ such that $ed \equiv 1 \pmod{\phi(N)}$;
5 Encryption: $m \mapsto m^e \bmod N$;
6 Decryption: $c \mapsto c^d \bmod N$;
```

Algorithm 1: Implementing RSA [126].

Primality Testing

In Step 1 of Algorithm 1, one needs to generate large random primes p, q. This is done by generating large random numbers and thereafter testing those for primality. This approach works well due to the Prime Number Theorem (PNT) [126].

Theorem 2.5 (PNT). *There are approximately $\frac{N}{\ln N}$ primes smaller than N.*

In simpler words, the Prime Number Theorem states that on an average, one will find a prime if he/she chooses $\ln p$ many random integers of bitsize l_p.

The next step is primality test. There are many efficient primality testing algorithms in practice. One may use the Solovay-Strassen Algorithm [123] or the Miller-Rabin Algorithm [90, 105] for this purpose. The time complexity of both Solovay-Strassen and Miller-Rabin algorithms is $O(l_p^3)$ to test an l_p bit integer. For practical RSA instances, l_p will be 512 or 1024-bit integers, and hence both the algorithms work well to test whether a number is prime or not. However, both are probabilistic algorithms, and in practice Miller-Rabin algorithm fares better than the Solovay-Strassen algorithm. This is because the probability of failure of Miller-Rabin test (at most 4^{-k} for k different candidates) is much less compared to the one for Solovay-Strassen (at most 2^{-k} for k different candidates). If the Generalized Riemann Hypothesis is true then Miller-Rabin Algorithm can be run deterministically with time complexity $O(l_p^5)$. There are many more probabilistic polynomial time primality test algorithms based on elliptic curves, namely Goldwasser-Kilian [44], Atkin-Morain [6] etc.

Finding a deterministic polynomial time primality test was a open question for a long period. In 2002, Agrawal, Kayal and Saxena [3] proposed a deterministic primality test called the AKS algorithm. The time complexity of AKS was of $O(l_p^{12+\epsilon})$, and in 2005, Pomerance and Lenstra [101] improved the time complexity up to $O(l_p^{6+\epsilon})$. However, Miller-Rabin algorithm is much faster than AKS in practice and hence it is the one used in the implementation of RSA. A formal description of the Miller-Rabin test is presented in Algorithm 2.

```
   Input: $n > 3$, an odd integer to be tested for primality
   Output: ‘composite’ if $n$ is composite, otherwise ‘probably prime’
 1 Write $n - 1 = 2^s \cdot d$ with $d$ odd;
 2 Pick random $a \in [1, n-1]$;
 3 Set $x \leftarrow a^d \bmod n$;
 4 if $x = 1$ then
 5     Return (‘probably prime’);
   end
 6 for $i = 0$ to $s - 1$ do
 7     if $x \equiv -1 \pmod{n}$ then
 8         Return (‘probably prime’);
       end
 9     else
10         $x \leftarrow x^2 \bmod n$;
       end
   end
11 Return (‘composite’);
```

Algorithm 2: Miller-Rabin Primality Test [126].

Product of Two Integers

Multiplication is fast. Even if one uses a trivial schoolbook multiplication algorithm to find N and $\phi(N)$ in Step 2 of Algorithm 1, it can be done in polynomial number ($l_p \times l_q = O(l_p^2)$, as $l_p \approx l_q$) of single digit multiplications. However, with large sized integers like p and q, it may be useful to try the Karatsuba multiplication algorithm [87], which reduces the complexity to $3 \cdot l_p^{\log_2 3} \approx 3 \cdot l_p^{1.585}$ number of single digit multiplications.

Extended Euclidean Algorithm (EEA)

In Step 3 of Algorithm 1, we choose random integers e and test if $\gcd(e, \phi(N)) = 1$. To find the GCD, one can use the Euclidean algorithm as follows.

```
Input: Two positive integers $a, b$
Output: $\gcd(a, b)$, the GCD of the two input integers
1 Initialize $r_0 = a$, $r_1 = b$ and $m = 1$;
2 while $r_m \neq 0$ do
3     $q_m \leftarrow \lfloor \frac{r_{m-1}}{r_m} \rfloor$;
4     $r_{m+1} \leftarrow r_{m-1} - q_m r_m$;
5     $m = m + 1$ ;
  end
  m=m-1 ;
6 Return $r_m$;
```

Algorithm 3: The Euclidean Algorithm [126].

Let us present a simple proof that the Euclidean algorithm is efficient, that is, it runs in time polynomial in the size of the inputs. Note that we can list down the steps of the Euclidean algorithm as follows (with $a = r_0$, $b = r_1$).

$$\begin{aligned}
r_0 &= q_1 r_1 + r_2 \qquad \text{with } 0 < r_2 < r_1, \\
r_1 &= q_2 r_2 + r_3 \qquad \text{with } 0 < r_3 < r_2, \\
&\vdots \\
r_{m-2} &= q_{m-1} r_{m-1} + r_m \qquad \text{with } 0 < r_m < r_{m-1}, \\
r_{m-1} &= q_m r_m + 0 \qquad \text{as } r_{m+1} = 0.
\end{aligned}$$

Thus, we have $a = r_0 > b = r_1 > r_2 > r_3 > \cdots > r_{m-1} > r_m = \gcd(a, b)$ from the algorithm. Again, we know that $q_i \geq 1$ for each $i = 1, \ldots, m$, and thus $r_0 \geq r_1 + r_2 > 2r_2$, $r_2 \geq r_3 + r_4 > 2r_4$, and so on. This produces the relation

$$b = \begin{cases} r_1 > 2r_3 > 4r_5 > \cdots > 2^{m-1} r_m & \text{if } m \text{ is odd,} \\ r_1 > r_2 > 2r_4 > 4r_6 > \cdots > 2^{m-2} r_m & \text{if } m \text{ is even,} \end{cases}$$

which, in turn, gives $m < \log_2 b + 2$. Notice that m is the number of steps the loop in the Euclidean algorithm runs, and hence, the time complexity of the algorithm to find $\gcd(a, b)$ comes as $O(\log b)$ divisions. Each of these divisions takes at most $O(\log^2 a)$ operations where $a \geq b$. Hence, the time complexity of finding $\gcd(a, b)$ amounts to $O(\log^2 a \cdot \log b)$, that is $O(\log^3 a)$. If we perform a more rigorous analysis, the computational complexity to find the GCD of two integers, each of size l_N, can be proved to be $O(l_N^2)$ [126], and one needs to iterate this step a few times to obtain a suitable e.

After finding an e which is relatively prime to $\phi(N)$, one needs to find its inverse modulo $\phi(N)$ in Step 4 of the algorithm. That is, one needs to find an integer d such that $ed \equiv 1 \pmod{\phi(N)}$. For that one can use the Extended Euclidean algorithm, as presented in Algorithm 4. The time complexity of this algorithm is the same as that of the Euclidean algorithm, i.e, $O(l_N^2)$ for a, b with bitlength l_N.

```
Input: Two positive integers $a, b$
Output: $r, s, t$ with $r = sa + tb$ where $r = \gcd(a, b)$
1  Initialize $a_0 = a, b_0 = b, t_0 = 0, t = 1, s_0 = 1, s = 0$;
2  $q = \lfloor \frac{a_0}{b_0} \rfloor$;
3  $r = a_0 - qb_0$;
4  while $r > 0$ do
5      $t_1 = t_0 - qt$;
6      $t_0 = t$;
7      $t = t_1$ ;
8      $t_1 = s_0 - qs$;
9      $s_0 = s$;
10     $a_0 = b_0$;
11     $b_0 = r$;
12     $q = \lfloor \frac{a_0}{b_0} \rfloor$;
13     $r = a_0 - qb_0$;
   end
14 $r = b_0$;
15 return $r, s, t$.
```

Algorithm 4: The Extended Euclidean Algorithm [126].

Square and Multiply Algorithm

For Steps 5 and 6 in Algorithm 1, one has to perform modular exponentiations. For this purpose, one may use the Square and Multiply Algorithm [126]. The time complexity of a single modular exponentiation $x^y \bmod N$ using the square and multiply algorithm is $O(l_y l_N^2)$, which is $O(l_N^3)$ as the exponents e, d are both less than N. For a quick reference, we present the famous square and multiply algorithm for modular exponentiation in Algorithm 5.

```
  **Input**: $x, y, N$
  **Output**: $x^y \bmod N$
1 $z = y, u = 1, v = x$;
2 **while** $z > 0$ **do**
3     **if** $z \equiv 1 \pmod 2$ **then**
4         $u = uv \bmod N$;
      **end**
5     $v = v^2 \bmod N$; $z = \lfloor \frac{z}{2} \rfloor$ ;
  **end**
6 return $u$.
```

Algorithm 5: Square and Multiply Algorithm.

2.2.3 Variants of RSA

CRT-RSA

To speed up the decryption phase of RSA, Quisquater and Couvreur [103] proposed the use of Chinese Remainder Theorem (CRT) in the decryption phase. This variant of RSA is known as CRT-RSA, and it is the most widely accepted version of RSA in practice. The backbone of the scheme is CRT, stated as follows.

Theorem 2.6 (CRT). *Suppose $p_1, p_2, \dots, p_k$ $(k \geq 2)$ are pairwise relatively prime positive integers. Then for any set of integers $a_1, a_2, \dots, a_k$, there exists a unique $x < p_1 p_2 \cdots p_k$ such that*

$$x \equiv a_1 \pmod{p_1},\ x \equiv a_2 \pmod{p_2},\ \dots,\ x \equiv a_k \pmod{p_k}.$$

In case of CRT-RSA, we consider the special case of $k = 2$. In this scenario, x can be deduced as follows.

$$\begin{aligned} x \equiv a_2 \pmod{p_2} &\Rightarrow x = a_2 + lp_2 \quad \text{for some integer } l \text{, and thus} \\ x \equiv a_1 \pmod{p_1} &\Rightarrow a_2 + lp_2 \equiv a_1 \pmod{p_1} \\ &\Rightarrow l \equiv (a_1 - a_2) \times \left(p_2^{-1} \bmod p_1\right) \bmod p_1. \end{aligned}$$

This value of l, put back into the first congruence, gives the formula for x as

$$x = a_2 + lp_2 = \left(a_2 + ((a_1 - a_2) \times (p_2^{-1} \bmod p_1) \bmod p_1) \times p_2\right) \bmod p_1 p_2.$$

Example 2.7. Take $p_1 = 2$, $p_2 = 3$ and $a_1 = 1$, $a_2 = 2$. The goal is to find an integer $x < p_1p_2 = 6$ such that $x \equiv a_1 \pmod{p_1}$ and $x \equiv a_2 \pmod{p_2}$. One can check $x = 5$ uniquely satisfies such conditions.

Let us now describe the CRT-RSA model. Recall that the decryption key in RSA is (d, N). In CRT-RSA, the decryption key is (d_p, d_q, p, q) where

$$d_p \equiv d \bmod p - 1 \quad \text{and} \quad d_q \equiv d \bmod q - 1.$$

In the decryption phase of CRT-RSA, one have to first calculate

$$c_p \equiv c^{d_p} \pmod{p} \quad \text{and} \quad c_q \equiv c^{d_q} \pmod{q}.$$

Note that $m \equiv c^d \equiv c^{d_p} \equiv c_p \pmod{p}$ by Fermat's little Theorem [126], as $d_p \equiv d \bmod p - 1$. Similarly, $m \equiv c_q \pmod{q}$. Using c_p, c_q, p and q, one can find unique solution for $m \bmod N$ using Chinese Remainder Theorem (CRT), as follows.

$$m = \big(c_q + \big((c_p - c_q) \times \big(q^{-1} \bmod p\big) \bmod p\big) \times q\big) \bmod N.$$

Example 2.8. Consider Example 2.3 in RSA, with $p = 653, q = 877, N = 572681, e = 13$ and $d = 395413$. In this case $d_p = 301$ and $d_q = 337$. So when Bob gets the ciphertext $c = 536754$ as in Example 2.3, he first calculates

$$c_p = 536754^{301} \bmod 653 = 591 \quad \text{and} \quad c_q = 536754^{337} \bmod 877 = 67.$$

After finding c_p and c_q, Bob gets the plaintext $m = 12345$ using CRT on $c_p = 591, c_q = 67, p = 653$ and $q = 877$.

Note that the computation of m involves 1 modular subtraction (modulo p), 1 modular addition (modulo N) and 2 modular multiplications (modulo p and N). Each of these operations is very fast in practice. The most time consuming operation in the formula is the modular inversion $q^{-1} \bmod p$. Hence, to make the calculation of m faster, $q^{-1} \bmod p$ is stored as a part of the CRT-RSA decryption keys. As $l_p \approx l_q \approx \frac{l_N}{2}$ and the computations in CRT-RSA are performed modulo p, q instead of modulo N, the decryption phase in CRT-RSA is four times faster than RSA (three times if RSA uses Karatsuba multiplication [87] techniques).

CRT-RSA with $d_p - d_q = 2$

To reduce the storage space for the CRT-RSA parameters, Qiao and Lam [102] proposed using CRT-RSA with $d_p - d_q = 2$. In this case, one needs to store only one of the decryption exponents and the other one can be generated trivially at runtime. This variant of RSA is very useful in case of hand-held devices like smart-cards, which have comparatively low storage capacities. However, Jochemsz and May [65] showed that some lattice-based attacks on CRT-RSA can be made stronger if $d_p - d_q = 2$, and this is a major drawback of this variant of RSA.

Multi Prime RSA

Some time it is useful to choose an RSA modulus $N = p_1 p_2 \cdots p_r$ with distinct primes p_i, or to choose $N = p^r q$ with $p \neq q$. The first one is known as multi prime RSA, and the second variant was proposed by Takagi [127]. The reader may refer to [53] for various interesting results related to multi prime RSA.

Common Prime RSA

A special instance of RSA where $\gcd(p-1, q-1)$ is large, is known as common prime RSA. Attacks on RSA which exploit small decryption exponent d work less efficiently [52] in this variant of RSA. The reader may refer to [54] to know the current results related to common prime RSA.

2.3 Cryptanalysis of RSA

From the previous discussion, we may recall that $c = m^e \bmod N$ is the main relation between the plaintext and the ciphertext in RSA. From the point of view of an attacker, the RSA Problem can be framed as follows.

Problem 2.9 (RSA Problem). *Consider an RSA setup with public key (e, N), private key (d, N) and $c = m^e \bmod N$. Given just $\langle c, e, N \rangle$, find message m.*

As $c^{1/e}$ is m in $\mathbb{Z}_N$, if one can find the e-th root of the ciphertext c in $\mathbb{Z}_N$, then he/she can obtain the plaintext m. So RSA problem can also be stated as a problem to find the e-th root of a given integer modulo N. However it is believed to be hard when one chooses the RSA parameters properly. In this section, we shall take a tour of the existing attempts to solve the RSA problem in practice. A nice treatise on this topic can also be found in the recent book by Yan [131].

2.3.1 Factoring RSA Modulus

Note that if one can factor the RSA modulus N into its prime factors p, q, then one can easily calculate $\phi(N)$. So the decryption exponent d can be efficiently calculated from $e, \phi(N)$ using Extended Euclidean Algorithm. In this case RSA problem will no longer be hard. It is still unknown whether the converse is true. In 1998, Boneh and Venkatesan [18] provided some evidence to conclude that RSA problem may be easier than the factorization problem. After this work, a lot of research have been done in this direction [2, 19, 67].

Currently, the best known factorization algorithm is the Number Field Sieve (NFS) [76], with a time complexity of

$$\exp\left((1.902 + o(1)) \ln(N)^{\frac{1}{3}} \left(\ln\left(\ln(N)\right)\right)^{\frac{2}{3}}\right)$$

in an asymptotic sense ($N \to \infty$). In December 2009, 768-bit RSA modulus [69] has been factored using the Number Field Sieve. This factorization took over 2 years to execute and is currently the record in the field.

When one of the prime factors of N is significantly smaller than the other, one can use the Elliptic Curve Method (ECM) [79] for factorization. In March 2010, the current record for ECM factorization was set by factoring an integer with the smallest prime factor of size 241 bits [132]. For more results regarding integer factorization, the reader may refer to [75].

Shor [119, 120] proved that the factorization problem can be solved in polynomial time in the quantum computation model. However it is not yet clear whether quantum computers with sufficiently large register can be constructed.

Under the scope of this thesis, we concentrate on the factorization of RSA modulus $N = pq$. A lot of research has been done on general purpose factorization and there exist numerous interesting results. For example, one may refer to the result of Boneh et al [17] about the factorization of $n = p^r q$ for large r. Similarly, the paper of Erdös and Pomerance [37] can be cited as a relevant work on the largest prime factors of n and $n + 1$. One may also see the papers by Luca and Stănică [80], and Chen and Zhu [20] for results related to the prime power factorization of $n!$ for any positive integer n. There have been many attempts to factor integers of other special forms as well; namely the factorization of Fermat numbers [78], the factorization of Mersenne numbers [132] etc.

2.3.2 Partial Exposure of Primes

Coppersmith [22,24] proved that factorization of the RSA modulus can be achieved in polynomial time given half of the Most Significant Bits (MSBs), that is the contiguous top half, of one of the factors. Later Boneh et al [16] proved a similar result when half of the Least Significant Bits (LSBs), that is the contiguous lower half, of one of the factors is known.

2.3.3 Small Public Exponent Attack

Bob may choose the public exponent e very small (3, say) to allow Alice the privilege of faster encryption. Suppose an attacker knows a contiguous chunk of Most Significant Bits (MSBs) of m, and constructs an approximation m_1, such that m_1 and m share the known chunk of bits at the top. If $|m - m_1| < N^{\frac{1}{e}}$, then Coppersmith [23] proved that the attacker can find m in polynomial time. For an example, if $e = 3$, the attacker needs to know the top $\frac{2}{3}$-rd portion of the plaintext m to recover the whole of m in polynomial time.

2.3.4 Related Message Attack

In 1995, Franklin and Reiter [42] proved that when two plaintexts are sent using the same RSA modulus N with small public exponent e, then RSA may be weak in cases where the two plaintexts are polynomially related. This result was later improved by Coppersmith, Franklin, Patarin and Reiter [26].

Suppose that Alice sends the messages m_1 and $m_2 = \alpha m_1 + \beta$ to Bob where the integers α and β are known. Also suppose that Bob uses $e = 3$. So, we have

$$c_1 = m_1^3 \bmod N \quad \text{and} \quad c_2 = m_2^3 \bmod N.$$

The attacker can find m_1 and m_2 by computing

$$\frac{\beta(c_2 + 2\alpha^3 c_1 - \beta^3)}{\alpha(c_2 - \alpha^3 c_1 + 2\beta^3)} \equiv m_1 \pmod{N} \quad \text{and} \quad m_2 = \alpha m_1 + \beta \bmod N.$$

2.3.5 Broadcast Attack

Håstad [48, 49] proved that for small encryption exponent e, if the same plaintext m is sent to different receivers, then RSA may be weak. In 2008, May and Ritzenhofen [85] improved this attack of Håstad.

2.3.6 Timing Attack

In 1995, Kocher [72] proposed a new attack on RSA to obtain the private exponent d. He showed that an attacker can get a few bits of d by timing characteristic of an RSA implementing device. After the publication of this idea, the vulnerabilities of RSA were tested against a lot of side channel attacks in this direction [8,16,38].

2.3.7 Small Decryption Exponent Attack

In 1990, Wiener [130] proved that if the decryption exponent $d < \frac{1}{3}N^{\frac{1}{4}}$, one can factor N in polynomial time when the primes p, q are of the same bitsize. He used certain results from Continued Fractions to prove this. Let us first take a look at the theoretical background.

Continued Fraction (CF)

Given a positive rational number $\frac{a}{b}$, it can be represented as a finite CF expression as follows.

$$\frac{a}{b} = q_1 + \cfrac{1}{q_2 + \cfrac{1}{q_3 + \cdots + \cfrac{1}{q_m}}}$$

In short, one can write the CF representation as $[q_1, q_2, q_3, \ldots, q_m]$. For example, take the rational number $\frac{34}{99}$. One can write this as $\frac{34}{99} = [0, 2, 1, 10, 3]$, that is,

$$0 + \frac{1}{\frac{99}{34}} = 0 + \frac{1}{2 + \frac{31}{34}} = 0 + \frac{1}{2 + \frac{1}{\frac{34}{31}}} = 0 + \frac{1}{2 + \frac{1}{1+\frac{3}{31}}} = 0 + \frac{1}{2 + \frac{1}{1+\frac{1}{\frac{31}{3}}}} = 0 + \frac{1}{2 + \frac{1}{1+\frac{1}{10+\frac{1}{3}}}}.$$

Now consider a subsequence $[0, 2, 1]$ of $[0, 2, 1, 10, 3]$. Note that $0 + \frac{1}{2+\frac{1}{1}} = \frac{1}{3} = \frac{33}{99}$, which is very close to $\frac{34}{99}$. This indicates that a subsequence of a CF representation may produce a good approximation to the rational number. Any initial subsequence of $[q_1, q_2, q_3, \ldots, q_m]$, i.e, $[q_1, q_2, q_3, \ldots, q_r]$, where $1 \leq r \leq m$ is called a convergent of the original sequence $[q_1, q_2, q_3, \ldots, q_m]$. For example, $[0, 2, 1]$ is a convergent of $[0, 2, 1, 10, 3]$, which implies that $\frac{1}{3} = \frac{33}{99}$ is a convergent of $\frac{34}{99}$. Also note that if the subsequence has a 1 at the end then it may also be represented by adding that 1 to the previous integer and thus shortening the length of the subsequence. For example, both $[0, 2, 1]$ and $[0, 3]$ provide the same rational number. There are many interesting results about the convergence of continued fraction representations. For the purpose at hand, we need the following result [47].

Theorem 2.10. *Suppose* $\gcd(a, b) = \gcd(c, d) = 1$ *and* $|\frac{a}{b} - \frac{c}{d}| < \frac{1}{2d^2}$. *Then* $\frac{c}{d}$ *is represented by one of the convergents of the continued fraction expansion of* $\frac{a}{b}$.

It is also important to note that for t bit integers a, b, the CF expression $[q_1, q_2, q_3, \ldots, q_m]$ of $\frac{a}{b}$ can be computed in $O(\text{poly(t)})$ time and can be stored in $O(\text{poly(t)})$ space. Let us see how this may be utilized to attack RSA.

Wiener's Attack

We have $ed \equiv 1 \pmod N$. So we can write $ed = 1 + k\phi(N)$ for some integer k. So,

$$\frac{e}{\phi(N)} - \frac{k}{d} = \frac{1}{d\phi(N)}.$$

Note that $N - \phi(N) = pq - (p-1)(q-1) = p + q - 1 < 3\sqrt{N}$ when p, q are of the same bitsize. In this case,

$$\begin{aligned}\left|\frac{e}{N} - \frac{k}{d}\right| &= \left|\frac{ed - kN}{Nd}\right| \\ &= \left|\frac{1 + k(\phi(N) - N)}{Nd}\right| \\ &< \frac{3\sqrt{N}k}{Nd} = \frac{3k}{d\sqrt{N}}\end{aligned}$$

Recall that $d < N^{\frac{1}{4}}/3$. As $k < d$, we have $3k < 3d < N^{\frac{1}{4}}$. Hence, $\left|\frac{e}{N} - \frac{k}{d}\right| < \frac{1}{dN^{\frac{1}{4}}}$. Thus,

$$\left|\frac{e}{N} - \frac{k}{d}\right| < \frac{1}{3d^2} < \frac{1}{2d^2}.$$

In such a case, we can find d, k by computing convergents of the CF expansion of $\frac{e}{N}$, as indicated by the Theorem 2.10. Even when $d > N^{\frac{1}{4}}/3$, Wiener's attack may work sometimes, but the success probability approaches zero [125] as d grows.

Recall the RSA equation $ed = 1+k\phi(N)$, i.e., $ed = 1+k(N+1-p-q)$. If one can find the root $(x_0, y_0) = (k \bmod e, (1-p-q) \bmod e)$ of the modular polynomial $f_e(x, y) = 1 + x(N + y)$ in $\mathbb{Z}_e$, one can factor N as long as $|1 - p - q| < e$. In Eurocrypt 1999, Boneh and Durfee [14] proved that one can find the root of the polynomial $f_e(x, y)$ in polynomial time as long as $d < N^{0.292}$.

The approach of Boneh and Durfee was based on certain results of lattices. Many other results in RSA cryptanalysis rely on the study of lattices as well. We shall also exploit some lattice-based tools throughout this thesis. Hence, let us get familiar with some basic concepts and a few specialized tools from the theory of lattices before proceeding any further.

2.4 Lattice and LLL Algorithm

In this section we discuss a few facts about lattices and lattice basis reduction techniques. For basic results in linear algebra that we may use, the reader may refer to [57]. Let us present a set of definitions to initiate our discussion.

Definition 2.11 (Inner Product)**.** Let $\mathbf{v_1} = (a_1, a_2, \ldots, a_m), \mathbf{v_2} = (b_1, b_2, \ldots, b_m)$ be two vectors in $\mathbb{Z}^m$. Then the inner product of $\mathbf{v_1}, \mathbf{v_2}$ is denoted by $\langle \mathbf{v_1}, \mathbf{v_2} \rangle$ and defined as $\langle \mathbf{v_1}, \mathbf{v_2} \rangle = a_1b_1 + a_2b_2 + \cdots + a_mb_m = \sum_{i=1}^{m} a_ib_i$.

Definition 2.12 (Euclidean norm)**.** Euclidean norm of a vector $\mathbf{v_1}$ is denoted by $||\mathbf{v_1}||$ and defined as $||\mathbf{v_1}|| = \sqrt{\langle \mathbf{v_1}, \mathbf{v_1} \rangle}$.

Definition 2.13 (Lattice)**.** Let $\mathbf{v_1}, \dots, \mathbf{v_n} \in \mathbb{Z}^m$ $(m \geq n)$ be n linearly independent vectors. A lattice L spanned by $\{\mathbf{v_1}, \dots, \mathbf{v_n}\}$ is the set of all integer linear combinations of $\mathbf{v_1}, \dots, \mathbf{v_n}$. That is,

$$L = \left\{ \mathbf{v} \in \mathbb{Z}^m \mid \mathbf{v} = \sum_{i=1}^{n} a_i\mathbf{v_i} \text{ with } a_i \in \mathbb{Z} \right\}.$$

We often say that L is the lattice spanned by the rows of the matrix M whose rows are $\mathbf{v_1}, \dots, \mathbf{v_n}$. We say m is the rank of the lattice L. The set of vectors $B = \{\mathbf{v_1}, \dots, \mathbf{v_n}\}$ is called a basis for L. The dimension of L is the number of linearly independent vectors in B, that is, $\dim(L) = n$. If $m = n$, then lattice L is called full rank lattice.

We say that two vectors $\mathbf{v_1}, \mathbf{v_2}$ are mutually orthogonal if $\langle \mathbf{v_1}, \mathbf{v_2} \rangle = 0$. Given a basis $B = \{\mathbf{v_1}, \dots, \mathbf{v_n}\}$ for a lattice L, a question of interest is whether it is possible to produce another basis B^* of L such that all the new basis vectors are orthogonal to each other. This question had given rise to orthogonalization techniques in lattices, and the main algorithm used for lattice basis orthogonalization is the Gram-Schmidt orthogonalization process, as stated in Definition 2.14.

Definition 2.14 (Gram-Schmidt Orthogonalization)**.** The Gram-Schmidt orthogonalization of the set of vectors $\{\mathbf{v_1}, \dots, \mathbf{v_n}\}$ in $\mathbb{Z}^m$ is denoted by $\{\mathbf{v_1}^*, \dots, \mathbf{v_n}^*\}$ where

$$\mathbf{v_i}^* = \mathbf{v_i} - \sum_{j=1}^{i-1} \mu_{i,j}\mathbf{v_j}^* \quad \text{with} \quad \mu_{i,j} = \frac{\langle \mathbf{v_i}, \mathbf{v_j}^* \rangle}{||\mathbf{v_j}^*||^2}.$$

After the Gram-Schmidt orthogonalization is performed on a lattice L, we can define another invariant of the lattice, called the Determinant.

Definition 2.15 (Determinant)**.** The determinant of L is defined as $\det(L) = \prod_{i=1}^{n} ||\mathbf{v_i}^*||$, where $||\mathbf{v}||$ denotes the Euclidean norm of $\mathbf{v}$, and $\mathbf{v_i}^*$ arise from Gram-Schmidt orthogonalization algorithm (Definition 2.14) applied to L.

When L is full rank, then $\det(L) = |\det(M)|$, where M is a matrix corresponding to L. For our purpose, we consider only full rank lattices in this thesis.

Example 2.16. Consider two vectors $\mathbf{v_1} = (1, 2), \mathbf{v_2} = (3, 4)$. Then $\langle \mathbf{v_1}, \mathbf{v_2} \rangle = 1 \cdot 3 + 2 \cdot 4 = 11$, and $||\mathbf{v_1}|| = \sqrt{5}$. The lattice L generated by $\mathbf{v_1}, \mathbf{v_2}$ is $L = \{\mathbf{v} \in \mathbb{Z}^2 \mid \mathbf{v} = a_1\mathbf{v_1} + a_2\mathbf{v_2} \text{ with } a_1, a_2 \in \mathbb{Z}\}$. Matrix M corresponding to L is $M = \begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix}$ and $B = \{\mathbf{v_1}, \mathbf{v_2}\}$ is a basis of L. Since $\mathbf{v_1}, \mathbf{v_2}$ are linearly independent, the dimension of L is 2 and L is a full rank lattice. Therefore, $\det(L) = |\det(M)| = 2$.

A problem of interest in the theory of lattices is the Shortest Vector Problem (SVP) [89]. This problem has been studied for ages and no exact solution to this has been found till date. The problem is as follows.

Problem 2.17 (SVP)**.** *Given a lattice L generated by a basis B, find the shortest vector $\mathbf{v} \in L$ with respect to a predetermined norm.*

Though no one could ever produce an algorithm that will solve SVP in polynomial time, there had been a lot of research in this area. A well known result by Minkowski [92] deals with this problem as well.

Theorem 2.18 (Minkowski)**.** *Every n-dimensional lattice L contains a nonzero vector $\mathbf{v}$ with $||\mathbf{v}|| \le \sqrt{n}\,(\det(L))^{\frac{1}{n}}$.*

Finding the shortest nonzero vector in a lattice is very hard in general. However, one can use the famous LLL reduction algorithm [77] of A. K. Lenstra, H. W. Lenstra Jr., and L. Lovász to approximate the shortest vector. The technique is as presented in Algorithm 6. LLL algorithm performs some elementary row operations on the matrix M corresponding to L, and produces an alternate basis with certain nice properties, as follows [77].

Definition 2.19 (LLL Reduced Basis)**.** We say that a set of basis vectors $B = \{\mathbf{r_1}, \mathbf{r_2}, \ldots, \mathbf{r_n}\}$ is LLL reduced if

$$(i) \quad |\mu_{ij}| \le \frac{1}{2} \quad \text{for all} \quad 1 \le i \le n \text{ and } j < i,$$

$$(ii) \quad \frac{3}{4}||\mathbf{r_i}^*||^2 \le ||\mu_{i+1,i}\mathbf{r_i}^* + \mathbf{r_{i+1}}^*||^2 \quad \text{for all} \quad 1 \le i \le n.$$

A natural question to ask in this direction is the measure of reduction using LLL. It is also of interest to know the running time of the algorithm for practical purpose. In this respect, we have the following result.

Lemma 2.20. *Let L be an integer lattice of dimension n generated by the basis vectors $\{\mathbf{v_1}, \mathbf{v_2}, \ldots, \mathbf{v_n}\}$. Then the LLL algorithm applied on L outputs a reduced basis of L spanned by $\{\mathbf{r_1}, \ldots, \mathbf{r_n}\}$ with*

$$||\mathbf{r_1}|| \leq ||\mathbf{r_2}|| \leq \cdots \leq ||\mathbf{r_i}|| \leq 2^{\frac{n(n-1)}{4(n+1-i)}} \det(L)^{\frac{1}{n+1-i}}, \quad \text{for } i = 1, \ldots, n$$

in time polynomial in the lattice dimension n and the bitsize of the entries of the matrix M corresponding to L.

```
   Input: A lattice $L$ with basis $\{\mathbf{v_1}, \mathbf{v_2}, \ldots, \mathbf{v_n}\} \in \mathbb{Z}^n$.
   Output: LLL reduced basis $\{\mathbf{r_1}, \mathbf{r_2}, \ldots, \mathbf{r_n}\}$ for $L$.
 1 Compute $\mathbf{v_1}^*, \mathbf{v_2}^*, \ldots, \mathbf{v_n}^*$;
 2 for $i = 2$ to $n$ do
 3     for $j = i - 1$ to 1 do
 4         $\mathbf{v_i} = \mathbf{v_i} - [\mu_{i,j}]\,\mathbf{v_j}$;    // $[\mu_{i,j}]$ means the integer closest to $\mu_{i,j}$
       end
   end
 5 if $\exists\ i$ such that $\frac{3}{4}||\mathbf{v_i}^*||^2 \geq ||\mu_{i+1,i}\mathbf{v_i}^* + \mathbf{v_{i+1}}^*||$ then
 6     $\mathbf{c} = \mathbf{v_i}$;
 7     $\mathbf{v_i} = \mathbf{v_{i+1}}$;
 8     $\mathbf{v_{i+1}} = \mathbf{c}$;
   end
 9 go to 1;
10 return $\{\mathbf{r_1}, \mathbf{r_2}, \ldots, \mathbf{r_n}\} = \{\mathbf{v_1}, \mathbf{v_2}, \ldots, \mathbf{v_n}\}$.
```

Algorithm 6: LLL algorithm for lattice reduction.

Example 2.21. Consider the Example 2.16, where lattice L is generated by $B = \{\mathbf{v_1}, \mathbf{v_2}\}$ with $\mathbf{v_1} = (1, 2), \mathbf{v_2} = (3, 4)$. Now we discuss how Algorithm 6 works on this basis B.

1. $\mathbf{v_1}^* = \mathbf{v_1} = (1, 2)$.
2. $[\mu_{2,1}] = \left[\frac{\langle \mathbf{v_2}, \mathbf{v_1}^* \rangle}{||\mathbf{v_1}^*||^2}\right] = [\frac{11}{5}] = 2$.
3. $\mathbf{v_2} = (3, 4) - 2(1, 2) = (1, 0)$.

4. $\frac{3}{4}||\mathbf{v_1}^*||^2 = \frac{3}{4}\cdot 5 = \frac{15}{4} > ||\mu_{2,1}\mathbf{v_1}^* + \mathbf{v_2}^*|| = ||\frac{1}{5}(1,2) + (\frac{4}{5}, -\frac{2}{5})|| = ||(1,0)|| = 1.$

5. $\mathbf{v_1} = (1,0), \mathbf{v_2} = (1,2).$

6. $\mathbf{v_1}^* = \mathbf{v_1} = (1,0).$

7. $[\mu_{2,1}] = \left[\frac{\langle \mathbf{v_2}, \mathbf{v_1}^* \rangle}{||\mathbf{v_1}^*||^2}\right] = [\frac{1}{1}] = 1.$

8. $\mathbf{v_2} = (1,2) - 1(1,0) = (0,2).$

9. $\frac{3}{4}||\mathbf{v_1}^*||^2 = \frac{3}{4} < ||\mu_{2,1}\mathbf{v_1}^* + \mathbf{v_2}^*|| = ||0(1,0) + (0,2)|| = 2.$

Hence LLL reduced basis is $B' = \{\mathbf{r_1}, \mathbf{r_2}\}$ with $\mathbf{r_1} = (1,0), \mathbf{r_2} = (0,2)$.

There are numerous applications of lattices in cryptology, both for cryptanalysis and for constructive cryptographic design. In this thesis we mainly focus on Coppersmith's [24] idea and its modifications for solving polynomials using lattice basis reduction. The root finding techniques proposed by Coppersmith are comprehensively discussed in the Doctoral thesis of Jochemsz [64]. For many more applications of lattices, the reader may refer to the papers by Joux and Stern [68], and Nguyen and Stern [94]. For more results related to lattices and lattice basis reduction, refer to [21, 95, 106, 121].

2.5 Solving Modular Polynomials

In 1996, Coppersmith [23] introduced a method for finding small modular roots of univariate polynomial. Since then, the method is used in various applications in Public Key Cryptography.

Let $f_N(x) = \sum_i a_i x^i$ be a univariate modular polynomial over $\mathbb{Z}_N$. The terms x^i of f_N with nonzero coefficients are called monomials. The norm of the polynomial is defined as $||f_N|| = \sqrt{\sum_i a_i^2}$. In general, the roots of $f_N(x)$ can not be found [25, Section 6] efficiently. However, finding small roots may be possible in polynomial time. To find a small modular root x_0 of $f_N(x)$, one needs to find a polynomial $h(x)$ such that $h(x_0) = 0$ holds over integers. Then following the method of Sturm sequence [70], the root x_0 may be obtained efficiently. This is the main idea behind Coppersmith's method.

2.5.1 Coppersmith's Method

Suppose we have a modular polynomial $f_N(x)$ with a small root x_0 in $\mathbb{Z}_N$. To construct the polynomial $h(x)$, we fix a positive integer m and generate a set of univariate polynomials g_{jk}, called the shift polynomials, as follows.

$$g_{jk}(x) = x^j (f_N(x))^k N^{m-k} \quad \text{for } k = 0, \ldots, m \text{ and some choice of } j.$$

Clearly, $g_{jk}(x_0) = 0 \bmod N^m$. Thus, any integer linear combination $h(x)$ of $g_{jk}(x)$ has also got a root x_0 modulo N^m. Now if $|h(x_0)| < N^m$, then $h(x_0) = 0$ holds over integers as well, and we have obtained our desired function $h(x)$.

The following theorem due to Howgrave-Graham [59] reformulates Coppersmith's idea of finding modular roots to prescribe a condition under which we can conclude that $h(x_0) = 0$ holds over integers.

Theorem 2.22. *Let $h(x) \in \mathbb{Z}[x]$ be an integer polynomial with n monomials. Further, let m be a positive integer. Then, $h(x_0) = 0$ over integers if the following two conditions are satisfied.*

- $h(x_0) \equiv 0 \pmod{N^m}$ *and* $|x_0| < X$
- $||h(xX)|| < \frac{N^m}{\sqrt{n}}$.

Proof. Notice that we have

$$|h(x_0)| = \left| \sum_i h_i x_0^i \right| \le \sum_i \left| h_i x_0^i \right| \le \sum_i |h_i| X^i \le \sqrt{n} \cdot ||h(xX)|| < N^m.$$

Now since N^m divides $h(x_0)$ by the first condition, $h(x_0) = 0$. ■

To obtain the desired function $h(x)$, we construct a lattice L with basis vectors coming from the coefficient vectors of the polynomials $g_{jk}(xX)$. Now, our goal is to find an integer linear combination of these coefficient vectors, that is a vector in the lattice L, for which the norm is smaller than $\frac{N^m}{\sqrt{n}}$. Such a function will satisfy the conditions of Theorem 2.22. But this problem is analogous to finding a short vector in a given lattice, and one can use the LLL algorithm to the basis spanned by the coefficient vectors of $g_{jk}(xX)$ to do so. Note that the polynomial $r_1(x)$ corresponds to the smallest vector generated by the LLL algorithm over an

n dimensional lattice L satisfies

$$||r_1(xX)|| < 2^{\frac{n-1}{4}}(\det(L))^{\frac{1}{n}}.$$

Thus, if $2^{\frac{n-1}{4}}(\det(L))^{\frac{1}{n}} < \frac{N^m}{\sqrt{n}}$, then the LLL reduction gives us a polynomial $h(x) = r_1(x)$ such that $||h(xX)|| < \frac{N^m}{\sqrt{n}}$ and hence $h(x_0) = 0$. This $h(x)$ is our desired polynomial to find x_0 using common root finding algorithms.

We can extend the idea for multivariate case as well. But in that case, we need some assumptions, and hence the method becomes heuristic. Consider a multivariate polynomial $f_N(x_1, \ldots, x_t)$ with root $(x_1^{(0)}, \ldots, x_t^{(0)})$. Similar to the univariate case, we can construct a lattice L using shift polynomials $g_{i_1,\ldots,i_t,k}(x_1, \ldots, x_t) = x_1^{i_1} \cdots x_t^{i_t} (f_N(x_1, \ldots, x_t))^k N^{m-k}$, for some fixed m, $k = 0, \ldots, m$ and $i_1, \ldots, i_t$ non-negative integers. Clearly, $g_{i_1,\ldots,i_t,k}(x_1^{(0)}, \ldots, x_t^{(0)}) \equiv 0 \pmod{N^m}$. Now one can extend Theorem 2.22 for the multivariate case as follows.

Theorem 2.23. *Let $h(x_1, \ldots, x_t) \in \mathbb{Z}[x_1, \ldots, x_t]$ be the sum of at most ω monomials. Suppose that $h(x_1^{(0)}, \ldots, x_t^{(0)}) \equiv 0 \pmod{N^m}$ where $|x_1^{(0)}| \leq X_1, \ldots, |x_t^{(0)}| \leq X_t$ and $||h(x_1X_1, \ldots, x_tX_t)|| < \frac{N^m}{\sqrt{\omega}}$. Then $h(x_1^{(0)}, \ldots, x_t^{(0)}) = 0$ over integers.*

Therefore, combining Lemma 2.20 and Theorem 2.23, we can say that if

$$2^{\frac{\omega(\omega-1)}{4(\omega+1-t)}} \det(L)^{\frac{1}{\omega+1-t}} < \frac{N^m}{\sqrt{\omega}}, \tag{2.1}$$

then the LLL basis reduction algorithm will produce $r_1(x_1^{(0)}, \ldots, x_t^{(0)}) = \cdots = r_t(x_1^{(0)}, \ldots, x_t^{(0)}) = 0$. One can now collect the root $(x_1^{(0)}, \ldots, x_t^{(0)})$ from the polynomials $r_1(x_1, \ldots, x_t), \ldots, r_t(x_1, \ldots, x_t)$ if they are algebraically independent. We say polynomials $r_1, \ldots, r_t$ are algebraically independent if and only if $P(r_1, \ldots, r_t) = 0 \Leftrightarrow P \equiv 0$ for any polynomial P defined over $Q[x_1, \ldots, x_t]$. However, all we know that the basis vectors $r_1, \ldots, r_t$ generated by LLL are linearly independent, and in general one can not prove their algebraic independence.

If $r_1, \ldots, r_t$ are algebraically independent, one can find the common root using the method of resultants [30, Section 3]. For our purpose we need the following.

Proposition 2.24. *Suppose that $r_1(x_1, \ldots, x_t)$ and $r_2(x_1, \ldots, x_t)$ share the common root $(x_1^{(0)}, \ldots, x_t^{(0)})$. Let $R(r_1, r_2)$ be the resultant of r_1, r_2 with respect to x_t. Then, we have*

- *$R(r_1, r_2)$ is a polynomial in $t-1$ variables $x_1, \ldots, x_{t-1}$,*

- $R(r_1, r_2) \neq 0$ *if* r_1, r_2 *are algebraically independent, i.e.,* $\gcd(r_1, r_2) = 1$, *and*
- $R(r_1, r_2)(x_1^{(0)}, \ldots, x_{t-1}^{(0)}) = 0$.

Using the method of resultants recursively, we can find a polynomial in x_1 (with root $x_1^{(0)}$) in $t-1$ iterations. We can then find $x_1^{(0)}$ easily using some root finding algorithm. Furthermore, using back-substitution $t-1$ times, we can find $x_2^{(0)}, \ldots, x_t^{(0)}$ as well, one in each step of back-substitution.

Similarly one can use the technique of Gröbner Basis [30] to find the roots. The Gröbner Basis $G = \{g_1, g_2, \ldots, g_l\}$ is a set of polynomials such that

$$g_1(x_1^{(0)}, \ldots, x_t^{(0)}) = g_2(x_1^{(0)}, \ldots, x_t^{(0)}) = \cdots = g_l(x_1^{(0)}, \ldots, x_t^{(0)}) = 0.$$

Each polynomial g_i can be computed with respect to some ordering that eliminates the variables. So it is easy to extract the desired root. However, the elimination of variables fails if the variety $V(I)$ of the ideal I generated by $r_1, r_2, \ldots, r_t$ is not zero-dimensional. An interested reader may refer to [30] for more details regarding Gröbner Basis.

Occasionally, one can collect the roots by examining special structure in the polynomials as well. For example, if we get a polynomial $f(x, y) = ax - by + b$ where a, b are constants, then $(b, a+1)$ is quite clearly a root of $f(x, y)$.

From our discussion so far, we can conclude that in case of solving multivariate modular polynomials, one needs the following assumption before attempting a lattice based technique.

Assumption 1: The common root $(x_1^{(0)}, \ldots, x_t^{(0)})$ can be efficiently collected from the polynomials $r_1, \ldots, r_t$ using the method of resultants, Gröbner Basis technique or exploiting the structure of the polynomials.

Once we assume the above mentioned statement, (2.1) reduces the condition for solving a modular polynomial using LLL to the following:

$$2^{\frac{\omega(\omega-1)}{4(\omega+1-t)}} \det(L)^{\frac{1}{\omega+1-t}} < \frac{N^m}{\sqrt{\omega}}.$$

If we treat the number of variables t, and the dimension of lattice L as constants, then we can simply state the condition as $\det(L) < N^{m\omega}$. This technique of finding modular solutions to polynomials was generalized by Jochemsz and May [65], and we shall discuss their method in the next section.

2.5.2 General Method by Jochemsz and May

In Asiacrypt 2006, Jochemsz and May [65] proposed a method to find a small root $(x_1^{(0)}, \ldots, x_t^{(0)})$ of a polynomial $f_N(x_1, \ldots, x_t)$ modulo a composite integer N of unknown factorization. Let us first study the basic strategy proposed by Jochemsz and May.

Basic Strategy

Suppose one knows an upper bound for the root, namely $|x_j^{(0)}| < X_j$ for some given X_j, for $j = 1, \ldots, t$. First choose a monomial l of f_N such that no monomial in f_N besides l is divisible by l. Let a_l be the coefficient of l in f_N. Assume N, a_l are relatively prime. Otherwise we have a proper factor of N by computing $\gcd(N, a_l)$. Define $f'_N = a_l^{-1} f_N \bmod N$. Also define the sets M_k as in [65, Basic Strategy] for $0 \leq k \leq m$, where m is a positive integer satisfying certain properties that we shall discuss shortly.

$$\begin{aligned} M_k = \{x_1^{i_1} x_2^{i_2} \cdots x_t^{i_t} \mid\ & x_1^{i_1} x_2^{i_2} \cdots x_t^{i_t} \text{ is a monomial of } f_N^m \\ & \text{and } \frac{x_1^{i_1} x_2^{i_2} \cdots x_t^{i_t}}{l^k} \text{ is a monomial of } f_N^{m-k}\}. \end{aligned}$$

Also denote $M_{m+1} = \emptyset$. Define the shift polynomials as follows:

$$\begin{aligned} g_{i_1,\ldots,i_t,k}(x_1, x_2, \ldots, x_t) =& \frac{x_1^{i_1} x_2^{i_2} \cdots x_t^{i_t}}{l^k} f_N^k(x_1, x_2, \ldots, x_t)^k N^{m-k} \\ & \text{for } k = 0, \ldots, m \text{ and } x_1^{i_1} x_2^{i_2} \cdots x_t^{i_t} \in M_k \setminus M_{k+1}. \end{aligned}$$

Note that for any shift polynomial g, we have $g(x_1^{(0)}, \ldots, x_t^{(0)}) \equiv 0 \pmod{N^m}$. Now one needs to form a lattice L by taking the coefficient vectors of the polynomials $g(x_1 X_1, \ldots, x_t X_t)$ as a basis. In [65], it is proved that when $X_1^{s_1} \cdots X_t^{s_t} < N^{s_0 - \epsilon}$ for some arbitrary small $\epsilon > 0$ with $s_r = \sum_{x_1^{i_1} \cdots x_t^{i_t} \in M_0} i_r$ for $r = 1, \ldots, t$ and $s_0 = \sum_{1 \leq k \leq m} |M_k|$, then one can find t many polynomials r_i such that $r_i(x_1^{(0)}, \ldots, x_t^{(0)}) = 0$ after the LLL lattice reduction over L. Corresponding to the arbitrary small $\epsilon > 0$, it is always possible to get some positive integer m which satisfies the above condition. This m is what we choose to define the sets M_k as before. After obtaining the polynomials r_i, one can collect the root $(x_1^{(0)}, \ldots, x_t^{(0)})$ efficiently under Assumption 1.

Extended Strategy

Now we may move on to the extended strategy of [65]. In some cases it is useful for some polynomial to have extra shifts over some variable(s). Suppose we use extra μ many shifts over x_1. Then the definition of the sets M_k will be change from the previous case, as follows.

$$M_k = \bigcup_{0 \le j \le \mu} \{x_1^{i_1+j} x_2^{i_2} \cdots x_t^{i_t} \mid x_1^{i_1} x_2^{i_2} \cdots x_t^{i_t} \text{ is a monomial of } f_N^m \text{ and } \frac{x_1^{i_1} x_2^{i_2} \cdots x_t^{i_t}}{l^k} \text{ is a monomial of } f_N^{m-k}\}.$$

Rest of the techniques are the same as those in the basic strategy. Let us illustrate the effectiveness of the extended strategy of [65] in cryptanalysis of RSA, using the following example.

Boneh-Durfee [14] Attack

The attack by Boneh and Durfee [14] for low decryption exponent follows the extended strategy of [65]. Recall the RSA equation $ed = 1+k(N+1-p-q)$. Hence $1+k(N+1-p-q) \equiv 0 \pmod{e}$. The aim is to find the root $(x_0, y_0) = (k, -p-q+1)$ of the modular polynomial $f_e(x, y) = 1 + x(N + y) \bmod e$. Towards the solution, define the following sets (note that we are dealing with a bivariate polynomial).

$$M_k = \bigcup_{0 \le j \le \mu} \{x^{i_1} y^{i_2+j} \mid x^{i_1} y^{i_2} \text{ is a monomial of } f_e^m \text{ and } \frac{x^{i_1} y^{i_2}}{l^k} \text{ is a monomial of } f_e^{m-k}\}, \text{ for } l = xy.$$

Let us discuss the technique for $m = 3, \mu = 1$, say. Then we have

$$\begin{aligned}
M_0 &= \{x^3y^4, x^3y^3, x^3y^2, x^2y^3, x^3y, x^2y^2, x^3, x^2y, xy^2, x^2, xy, x, y, 1\},\\
M_1 &= \{x^3y^4, x^3y^3, x^3y^2, x^2y^3, x^3y, x^2y^2, x^2y, xy^2, xy\},\\
M_2 &= \{x^3y^4, x^3y^3, x^3y^2, x^2y^3, x^2y^2\},\\
M_3 &= \{x^3y^4, x^3y^3\}, \text{ and}\\
M_4 &= \emptyset.
\end{aligned}$$

The shift polynomials for the extended strategy in this case are as follows.

$$\mathcal{P} = \{ye^3, e^3, x^2e^3, x^3e^3, xe^3, x^2fe^2, xfe^2, fe^2, yfe^2, xf^2e, f^2e, yf^2e, f^3, yf^3\}.$$

Now take positive integers X, Y such that $|k| \le X$ and $|-p-q+1| \le Y$, and build a lattice L with the basis elements coming from the coefficients of the polynomials $p(xX, yY)$ where $p \in \mathcal{P}$. The lattice L is represented as follows.

poly	y	1	x^2	x^3	x	x^3y	x^2y	xy	xy^2	x^3y^2	x^2y^2	x^2y^3	x^3y^3	x^3y^4
ye^3	Ye^3													
e^3		e^3												
x^2e^3			X^2e^3											
x^3e^3				X^3e^3										
xe^3					Xe^3									
x^2fe^2			$-$	$-$		X^3Ye^2								
xfe^2			$-$		$-$		X^2Ye^2							
fe^2		$-$			$-$			XYe^2						
yfe^2	$-$							$-$	XY^2e^2					
xf^2e			$-$	$-$	$-$	$-$	$-$			X^3Y^2e				
f^2e		$-$	$-$		$-$		$-$	$-$			X^2Y^2e			
yf^2e	$-$						$-$	$-$	$-$		$-$	X^2y^3e		
f^3		$-$	$-$	$-$	$-$	$-$	$-$	$-$		$-$	$-$		X^3Y^3	
yf^3	$-$					$-$	$-$	$-$	$-$	$-$	$-$	$-$	$-$	x^3y^4

Here '$-$' denotes nonzero elements not belonging to the diagonal. If we perform LLL lattice reduction over L as defined by the above mentioned matrix, we shall obtain two polynomials $f_1(x, y), f_2(x, y)$ such that $f_1(x_0, y_0) = f_2(x_0, y_0) = 0$. Thereafter, we can collect the x_0 and y_0 from $f_1(x, y), f_2(x, y)$ efficiently, subject to Assumption 1.

In [14], Boneh and Durfee proved that when $m \to \infty$, if a proper choice of μ is made depending on m, then one can solve $f_e(x, y)$ in $\mathbb{Z}_e$ when $|d| < N^{0.284}$. However, in the same paper [14], they improved the bound to $|d| < N^{0.292}$ using a sublattice of the lattice L described above. This is the best bound till date. Though they [14] conjectured that RSA may be insecure for $|d| < \sqrt{N}$, it is still an open problem.

2.6 Solving Integer Polynomials

Although there exist several techniques for solving univariate polynomials over integers, it is not so easy to solve bivariate integer polynomials. Coppersmith [22] introduced a method to find small integer roots for a bivariate polynomial $f(x, y)$. Without loss of generality, we can assume that $f(x, y)$ is irreducible. If $f(x, y)$ is reducible, we can factor $f(x, y)$ by the method of Wang and Rothschild [129] and try to find the roots of its factors individually. Coron [28] reformulated Coppersmith's method to propose the following idea.

2.6.1 Coron's Method

The main aim is to find a polynomial $h(x, y)$ which is algebraically independent of $f(x, y)$ and which shares the integer root (x_0, y_0) of $f(x, y)$. Let, $|x_0| \leq X$, $|y_0| \leq Y$, and assume that $f(x, y) = \sum_{i,j} a_{ij} x^i y^j$. Now, define $W = \max_{i,j} |a_{ij} X^i Y^j|$. Define $R = X^{l_1} Y^{l_2} W$ for some non-negative integers l_1, l_2. Further let us define

$$g_{ij}(x, y) = x^i y^j f(x, y) \frac{R}{W X^i Y^j} \quad \text{and} \quad h_{ij}(x, y) = x^i y^j R,$$

for some pair of integers i, j. Note that these $g_{ij}(x, y)$'s are analogous to the shift polynomials as in Section 2.5.2. To ensure that all the shift polynomials $g_{ij}(x, y)$ have integer coefficients, choose $l_1 \geq i$ and $l_2 \geq j$. Also note that $g_{ij}(x_0, y_0) \equiv 0 \pmod{R}$ and $h_{ij}(x_0, y_0) \equiv 0 \pmod{R}$. Now, construct a lattice L using the coefficient vectors of $g_{ij}(xX, yY)$ and $h_{ij}(xX, yY)$ as a basis. Let ω be the dimension of the lattice L, and assume that $r_1(xX, yY), \ldots, r_\omega(xX, yY)$ are the polynomials corresponding to the vectors of the LLL reduced basis of L.

Now, from Lemma 2.20, we know that $||r_1(xX, yY)|| \leq 2^{\frac{\omega-1}{4}} \det(L)^{\frac{1}{\omega}}$. Also, from Theorem 2.22, we know that if $||r_1(xX, yY)|| < \frac{R}{\sqrt{\omega}}$, then $r_1(x_0, y_0) = 0$. So, when $2^{\frac{\omega-1}{4}} \det(L)^{\frac{1}{\omega}} < \frac{R}{\sqrt{\omega}}$, then $r_1(x_0, y_0) = 0$. Since we choose $R = X^{l_1} Y^{l_2} W$, $r_1(x, y)$ is divisible by $X^{l_1} Y^{l_2}$. Now it can be shown that $r_1(x, y)$ is algebraically independent of $f(x, y)$. One can deduce this from the following result by Coron [28].

Theorem 2.25. *If $h(x, y)$ is a multiple of $f(x, y)$, then $h(x, y)$ is divisible by $X^{l_1} Y^{l_2}$ if and only if it has norm at least $2^{-(\rho+1)^2+1} X^{l_1} Y^{l_2} W$, where ρ is the maximum degree of the polynomials f, h in each variable separately.*

Hence, if $||r_1(xX, yY)|| \leq 2^{\frac{\omega-1}{4}} \det(L)^{\frac{1}{\omega}} \leq 2^{-(\rho+1)^2+1} X^{l_1} Y^{l_2} W = 2^{-(\rho+1)^2+1} R$, then $r_1(x,y)$ is algebraically independent from $f(x,y)$ and if $||r_1(xX,yY)|| \leq 2^{\frac{\omega-1}{4}} \det(L)^{\frac{1}{\omega}} \leq \frac{R}{\sqrt{\omega}}$, then $r_1(x_0,y_0) = 0$. The reader may note the difference between the above conditions and those in case of modular polynomials; here the terms do not depend on N anymore. So we can assume that when $\det(L) < R^\omega$, one can find a polynomial $h(x,y) = r_1(x,y)$ such that $h(x,y)$ is independent of $f(x,y)$, and $h(x,y)$ share the root (x_0,y_0) with $f(x,y)$. Thereafter, we can collect the root (x_0,y_0) efficiently using the method of resultants.

Similar to the modular case, we can extend the above idea to multivariate polynomials as well. But in that case we would need some assumptions, and hence the method becomes heuristic. Suppose we want to find the root $(x_1^{(0)},\ldots,x_t^{(0)})$ of

$$f(x_1,x_2,\ldots,x_t) = \sum a_{i_1 i_2\ldots i_t}\, x_1^{i_1}x_2^{i_2}\cdots x_t^{i_t}.$$

Let $|x_i^{(0)}| \leq X_i$ for $1 \leq i \leq t$. We define $W = ||f(x_1X_1, x_2X_1,\ldots,x_tX_t)||_\infty = \max_{i_1,i_2,\ldots,i_t} |a_{i_1i_2\ldots i_t}X_1^{i_1}X_2^{i_2}\cdots X_t^{i_t}|$. Let us also define $R = WX_1^{l_1}X_2^{l_2}\cdots X_t^{l_t}$ for some specific choices of l_i. Now, we define the shift polynomials:

$$\begin{aligned} g_{i_1\ldots i_t}(x_1,\ldots,x_t) &= x_1^{i_1}\cdots x_t^{i_t} f(x_1,\ldots,x_t)\cdot\frac{R}{WX_1^{i_1}\cdots X_t^{i_t}}, \text{ and}\\ h_{i_1\ldots i_t}(x_1,\ldots,x_t) &= x_1^{i_1}\cdots x_t^{i_t}\cdot R, \end{aligned}$$

for some set of integers $i_1,\ldots,i_t$. Choose l_i such that all shift polynomials $g_{i_1\ldots i_t}, h_{i_1\ldots i_t}$ have integer coefficients. Now, one may construct a lattice L using the coefficient vectors of $g_{i_1\ldots i_t}(x_1X_1,\ldots,x_tX_t)$ and $h_{i_1\ldots i_t}(x_1X_1,\ldots,x_tX_t)$ as a basis. Let ω be the dimension of the lattice L. From Lemma 2.20 and Theorem 2.23, we know that if

$$2^{\frac{\omega(\omega-1)}{4(\omega+2-t)}} \det(L)^{\frac{1}{\omega+2-t}} < \frac{R}{\sqrt{\omega}},$$

then we can find polynomials $r_1, r_2, \ldots, r_{t-1}$ such that $r_i(x_1^{(0)},\ldots,x_t^{(0)}) = 0$, for $1 \leq i \leq t-1$. Moreover, the following generalization of Coron's Theorem by Hinek and Stinson [56] guarantees the algebraic independence of all r_i from f.

Theorem 2.26. *If $h(x_1,\ldots,x_t)$ is a multiple of $f(x_1,\ldots,x_t)$, then h is divisible by $X_1^{l_1}X_2^{l_2}\cdots X_t^{l_t}$ if and only if it has norm at least $2^{-(\rho+1)^2+1}X_1^{l_1}X_2^{l_2}\cdots X_t^{l_t}W$, where ρ is the maximum degree of the polynomials f, h in each variable separately.*

Since it may be possible that r_i and r_j are algebraically dependent for some $1 \leq i \neq j \leq t-1$, we need to assume that the polynomials r_i, generated by LLL lattice reduction on L, are algebraically independent. If so, then we can collect the root $(x_1^{(0)}, \dots, x_t^{(0)})$ from $f, r_1, r_2, \dots, r_{t-1}$ using the method of resultants. The reader may note that this method is heuristic, depending on the validity of algebraic independence assumption. However, in Eurocrypt 2007, Bauer and Joux [7] proposed a deterministic method for finding integer roots of a trivariate polynomial. But now, let us study a generalization of the above method.

2.6.2 General Method by Jochemsz and May

In Eurocrypt 2005, Blömer and May [10] presented a general technique for solving a bivariate integer polynomial. In Asiacrypt 2006, Jochemsz and May [65] generalized the same and proposed a method to find a small root $(x_1^{(0)}, \dots, x_t^{(0)})$ of a polynomial $f(x_1, \dots, x_t)$. In this section we discuss this generalized idea.

Let d_i be the maximal degree of x_i in f for $1 \leq i \leq t$. Let us define

$$\begin{aligned} W &= ||f(x_1X_1, x_2X_2, \dots, x_tX_t)||_\infty, \text{ and} \\ R &= WX_1^{d_1(m-1)}X_2^{d_2(m-1)} \cdots X_t^{d_t(m-1)}. \end{aligned}$$

Let $a_0 = f(0, 0, \dots, 0)$ and assume that $a_0 \neq 0$. If $a_0 = 0$, then find some $(y_1, \dots, y_t)$ such that $f(y_1, \dots, y_t) \neq 0$, and define a new polynomial $f_1(x_1, x_2, \dots, x_t) = f(x_1 + y_1, x_2 + y_2, \dots, x_t + y_t)$. Clearly, the constant term of the new polynomial f_1 is nonzero as $f_1(0, 0, \dots, 0) = f(y_1, y_2, \dots, y_t) \neq 0$, and one can find the roots of f_1. Next, assume that $\gcd(a_0, R) = 1$. If not, then using the idea of [28, Appendix A], we increase X_i, W such that $\gcd(a_0, R) = 1$. Now we define $f'(x_1, x_2, \dots, x_t) = a_0^{-1} f \pmod R$, and start with the basic strategy.

Basic Strategy

Define the following sets for some positive integer m.

$$\begin{aligned} S &= \bigcup \{x_1^{i_1}x_2^{i_2} \cdots x_t^{i_t} \mid x_1^{i_1}x_2^{i_2} \cdots x_t^{i_t} \text{ is a monomial of } f^m\}, \text{ and} \\ M &= \bigcup \{x_1^{i_1}x_2^{i_2} \cdots x_t^{i_t} \mid x_1^{i_1}x_2^{i_2} \cdots x_t^{i_t} \text{ is a monomial of } f^{m+1}\}. \end{aligned}$$

Let l_j be the largest exponent of x_j that appears in the monomials of S.

Also define the following polynomials:

$$g_{i_1,i_2,\ldots,i_t} = x_1^{i_1}x_2^{i_2}\cdots x_t^{i_t} \cdot f'(x_1,x_2,\ldots,x_t) \cdot \prod_{j=1}^{t} X_j^{l_j-i_j} \quad \text{for } x_1^{i_1}x_2^{i_2}\cdots x_t^{i_t} \in S, \text{ and}$$
$$h_{i_1,i_2,\ldots,i_t} = x_1^{i_1}x_2^{i_2}\cdots x_t^{i_t} \cdot R \quad \text{for } x_1^{i_1}x_2^{i_2}\cdots x_t^{i_t} \in M \setminus S.$$

Note that for any shift polynomial g or h,

$$g(x_1^{(0)},\ldots,x_t^{(0)}) \equiv h(x_1^{(0)},\ldots,x_t^{(0)}) \equiv 0 \pmod{R}.$$

Now one have to construct a lattice L by taking the coefficient vectors of the polynomials $g(x_1X_1,\ldots,x_tX_t)$ and $h(x_1X_1,\ldots,x_tX_t)$ as a basis. In [65], it is proved that if

$$X_1^{s_1}\cdots X_t^{s_t} < W^{s-\epsilon}$$

with $s_r = \sum_{x_1^{i_1}\cdots x_t^{i_t}\in M\setminus S} i_r$ for $r = 1,\ldots,t$ and $s = |S|$, then one can find $t-1$ many polynomials r_i as the basis vectors of the LLL reduced basis of L, such that $r_i(x_1^{(0)},\ldots,x_t^{(0)}) = 0$ for all $1 \le i \le t-1$. Thereafter, subject to Assumption 1, we can efficiently collect the common root $(x_1^{(0)},\ldots,x_t^{(0)})$ from $f, r_1, \ldots, r_{t-1}$. Note that, similar to the modular case, the choice of m here depends on the arbitrary constant $\epsilon > 0$. Let us now discuss the extended strategy of [65] for finding integer roots of a polynomial.

Extended Strategy

Alike the modular case, it is useful for some polynomials to use extra shifts for some variable(s). Suppose that we use extra μ many shifts over x_1. Then the modified sets S, M will be defined as follows.

$$S = \bigcup_{0\le j\le\mu} \{x_1^{i_1+j}x_2^{i_2}\cdots x_t^{i_t} \mid x_1^{i_1}x_2^{i_2}\cdots x_t^{i_t} \text{ is a monomial of } f^m\}, \text{ and}$$
$$M = \bigcup \{\text{monomials of } x_1^{i_1}x_2^{i_2}\cdots x_t^{i_t}\cdot f \mid x_1^{i_1}x_2^{i_2}\cdots x_t^{i_t} \in S\}.$$

Now every idea of the basic strategy will remain the same except for the fact that we have to define $R = W \cdot \prod_{j=1}^{t} X_j^{l_j}$, where l_j is the maximum degree of x_j in the monomials of S. Let us now give a practical example of this strategy.

Attack by Ernst et al [38]

In 2005, Ernst et al [38] studied the case when some most significant bits (MSBs) of the decryption exponent d are known to the attacker. In such a situation, the attacker needs to find out the root of a polynomial of the form $f(x, y, z) = a_0 + a_1x + a_2y + a_3yz$. Let (x_0, y_0, z_0) be the root of $f(x, y, z)$ satisfying $|x_0| < X$, $|y_0| < Y$ and $|z_0| < Z$. Now suppose that we use extra shifts over the variable z. Let us discuss the technique with $m = 1, \mu = 1$. In this case, we have

$$\begin{aligned} S &= \{1, x, y, z, xz, yz, yz^2\}, \text{ and} \\ M &= \{1, x, y, z, xz, yz, yz^2, x^2, xy, xyz, y^2, y^2z, x^2z, xyz^2, y^2z^2, y^2z^3\}. \end{aligned}$$

In this case $W = \max\{|a_0|, |a_1X|, |a_2Y|, |a_3YZ|\}$. Define $R = XYZ^2W$, and calculate $f'(x, y, z) = a_0^{-1}f(x, y, z) \bmod R = 1 + ax + by + cyz$. In this case, one uses the shift polynomials

$$\begin{aligned} \mathcal{P}_1 =& \{f'XYZ^2, xf'YZ^2, yf'XZ^2, zf'XYZ, xzf'YZ, yzf'XZ, yz^2f'X\}, \text{ and} \\ \mathcal{P}_2 =& \{x^2R, xyR, xyzR, y^2R, y^2zR, x^2zR, xyz^2R, y^2z^2R, y^2z^3R\}, \end{aligned}$$

and builds a lattice L with the basis elements coming from the coefficients of $p(xX, yY, zZ)$ where $p \in \mathcal{P}_1 \cup \mathcal{P}_2$. The lattice L is represented as follows.

poly	1	x	y	z	xz	yz	yz^2	x^2	xy	xyz	y^2	y^2z	x^2z	xyz^2	y^2z^2	y^2z^3
$f'XYZ^2$	T	$-$	$-$			$-$										
$xf'YZ^2$		T						$-$	$-$	$-$						
$yf'XZ^2$			T						$-$		$-$	$-$				
$zf'XYZ$				T	$-$	$-$	$-$									
$xzf'YZ$					T					$-$			$-$	$-$		
$yzf'XZ$						T				$-$		$-$			$-$	
$yz^2f'X$							T							$-$	$-$	$-$
x^2R								X^2R								
xyR									XYR							
$xyzR$										$XYZR$						
y^2R											Y^2R					
y^2zR												Y^2ZR				
x^2zR													X^2ZR			
xyz^2R														XYZ^2R		
y^2z^2R															Y^2Z^2R	
y^2z^3R																Y^2Z^3R

Here $T = XYZ^2$ and '$-$' denotes the non zero elements in the matrix. If we perform the LLL lattice reduction on L, we get two polynomials $f_1(x, y, z)$ and $f_2(x, y, z)$ which share the root (x_0, y_0, z_0) of $f(x, y, z)$. From f, f_1, f_2, we can collect the root (x_0, y_0, z_0) by calculating the resultants, subject to Assumption 1.

2.7 Analysis of the Root Finding Techniques

In the previous sections we have studied a few root finding techniques for modular and integer polynomials using lattice based methods. We have so far analyzed the correctness and method of operation of these techniques. We shall now analyze the time complexity of the generic approach by Coppersmith [22], taking it as a representative of all similar techniques.

2.7.1 Time Complexity Analysis

The running time Coppersmith's root finding approach can be split into its two major components: lattice reduction, and root extraction using resultant method or Gröbner Basis calculation. The individual analysis of the two is as follows.

LLL Lattice Reduction

The runtime of Coppersmith's method is dominated by the lattice basis reduction which runs in time polynomial in the dimension ω of the lattice and bitsize of the entries of the lattice. The time complexity of lattice reduction algorithm by Nguyen and Stehlé [93] is $O\left(\omega^5\left(\omega + A\right)A\right)$, where A is the maximum bitsize of an entry in the lattice. In Coppersmith's construction, $\dim(L)$ depends only upon ϵ. Also, the bitsize of the entries in L are bounded by $\text{poly}(\log_2 \text{N})$. Hence we can find the LLL reduced basis vectors in time $\text{poly}(\log_2 \text{N})$.

Root Extraction

It is also worth noting that the degrees of $r_1, \dots, r_\omega$ are fixed as they depend only upon ϵ, and the coefficients of $r_1, \dots, r_\omega$ are bounded by a polynomial in $\log_2 N$. If $r_1(x_1, x_2, \dots, x_t)$ and $r_2(x_1, x_2, \dots, x_t)$ are two polynomials with degrees of x_1 equal to d_1, d_2 respectively, then to calculate the resultant $R(r_1, r_2)$ with respect to x_1, one needs to find the determinant of a matrix of size $(d_1 + d_2) \times (d_1 + d_2)$. One can calculate the determinant of a matrix of size D in $O(D^3)$ time. For detailed calculation of the resultant, the reader may consult [30, Chapter 3]. Similarly, the Gröbner Basis calculation is in general double-exponential in the degree of the polynomial, which in case of Coppersmith's method, is dependent only on ϵ.

Cumulative Runtime

From the earlier discussion, one may deduce that for a fixed positive number ϵ, and for a fixed number of variables, the time complexity of Coppersmith's lattice based root finding techniques is $\text{poly}(\log \mathrm{N})$. Although lattice reduction can be done in polynomial time, it is almost impossible to reduce a lattice of large dimension (larger than 400, say) that has large entries. The same problem occurs for the calculation of resultant or Gröbner Basis when the number of variables is too large.

2.8 Experimental Framework

Throughout this thesis, we have furnished numerous experimental results supporting our claims. These experiments, in most of the cases, are implementations of the lattice based root finding techniques and the algorithms that we have proposed. We have performed almost all (except for a few in Chapter 5) experiments using the following computing framework.

- Symbolic Computation Package: Sage [124] (versions 2.10 through 4.2)
- Operating System: Linux Ubuntu (8.10 through 9.10)
- System Configuration: Dual Core Intel® Pentium® D CPU 1.83 GHz with 2 GB RAM and 2 MB Cache

For the examples of prime reconstruction from random known bits of the primes from the LSB side, as presented in Chapter 5, we have implemented the reconstruction algorithm in the following framework.

- Coding Platform: C/C++ with gcc/g++ compiler using GMP library [43].
- Operating System: Linux Ubuntu 9.04
- System Configuration: Intel® Pentium® 4 CPU 1.7 GHz with 1 GB RAM and 2 MB Cache

2.9 Conclusion

Now that we have covered the preliminary topics of mathematics and cryptography that constitute the foundation of this thesis, we can get into the technical results. In a major part of the thesis, the lattice based root finding techniques serve as a backbone. One may refer back to the preliminaries whenever similar methods in the following chapters are encountered.

Chapter 3

A class of Weak Encryption Exponents in RSA

A lot of weakness of RSA have been identified in the past three decades, but still RSA can be securely used with proper precautions as a public key cryptosystem. In 1990, Wiener [130] proved that RSA is insecure if $d < \frac{1}{3}N^{\frac{1}{4}}$. Later 1999, Boneh and Durfee [14] improved this bound up to $N^{0.292}$. In [9], Blömer and May have shown that p, q can be found in polynomial time for every (N, e) satisfying $eX + \phi(N)Y = -y$, with $X \leq \frac{1}{3}N^{\frac{1}{4}}$ and $|y| = O(N^{-\frac{3}{4}}ex)$. Some extensions considering the difference $p - q$ have also been studied. The work of [9] uses the result of Coppersmith [24] as well as the idea of CF expression [130] in their proof. The number of such weak keys has been estimated as $N^{\frac{3}{4}-\epsilon}$.

In a similar direction of [9], further weak keys were presented by Nitaj [96,97]. The idea of [96] is as follows. Suppose that e satisfies the following property: there exist u, v, X, Y such that $eX - (p-u)(q-v)Y = 1$ with $1 \leq Y < X < 2^{-\frac{1}{4}}N^{\frac{1}{4}}$, $|u| < N^{\frac{1}{4}}$, $v = \left[-\frac{qu}{p-u}\right]$ ($[x]$ means the nearest integer of the real number x). If all the prime factors of $p - u$ or $q - v$ are less than 10^{50}, then N can be factored from the knowledge of N, e. The number of such weak exponents is estimated as $N^{\frac{1}{2}-\epsilon}$. So, in this case [96] number of weak exponents is smaller than [9].

In [96], Continued Fraction (CF) expression is used to find the unknowns X, Y among the convergents of $\frac{e}{N}$. We get immediate improvements over the results of [96] using the LLL [77] algorithm. Our results are as follows.

- The bound on Y can be extended till N^γ, with

$$\gamma < 4\alpha\tau\left(\frac{1}{4\tau}+\frac{1}{12\alpha}-\sqrt{\left(\frac{1}{4\tau}+\frac{1}{12\alpha}\right)^2+\frac{1}{2\alpha\tau}\left(\frac{1}{12}+\frac{\tau}{24\alpha}-\frac{\alpha}{8\tau}\right)}\right),$$

  given $e = N^\alpha$ and $|N-(p-u)(q-v)| = N^\tau$.

- The only constraint on X is to satisfy the equation $eX-(p-u)(q-v)Y = 1$, which gives

$$X = \frac{1+(p-u)(q-v)Y}{e}, \text{ i.e., } X = \lceil N^{1+\gamma-\alpha}\rceil.$$

- In [96], the constraint $1 \leq Y < X < 2^{-\frac{1}{4}}N^{\frac{1}{4}}$ forces that the upper bound of e is $O(N)$. However, in our case the value of e can exceed this bound. Our results work for e up to $N^{1.875}$ for $\tau = \frac{1}{2}$.

In fact, our result is more general. Instead of considering some specific form $eX-(p-u)(q-v)Y = 1$, we consider equations like $eX-ZY = 1$, where $Z = \psi(p,q,u,v)$ is a function of the RSA primes p,q and integers u,v. Given $e = N^\alpha$ and the constraint $|N-Z| = N^\tau$, we can efficiently find Z using the LLL algorithm when $|Y| = N^\gamma$, where

$$\gamma < 4\alpha\tau\left(\frac{1}{4\tau}+\frac{1}{12\alpha}-\sqrt{\left(\frac{1}{4\tau}+\frac{1}{12\alpha}\right)^2+\frac{1}{2\alpha\tau}\left(\frac{1}{12}+\frac{\tau}{24\alpha}-\frac{\alpha}{8\tau}\right)}\right).$$

We consider $Z = \psi(p,q,u,v) = N-pu-v$ to present a new class of weak keys in RSA. This idea does not require any kind of factorization as used in [96]. We estimate a lower bound of $N^{0.75-\epsilon}$ for the number of weak keys in this class. Hence the number of weak exponents is of the same magnitude as Blömer and May [9].

3.1 Our Basic Technique

In this section we build the framework for our analysis related to weak keys. First we present a result based on continued fraction (CF) expansions.

Lemma 3.1. *Let $N = pq$ be the RSA modulus. Consider that e satisfies the equation $eX - ZY = 1$ where $|N - Z| = N^\tau$. Then $\frac{Y}{X}$ is one of the convergents in the CF expansion of $\frac{e}{N}$ when $2XY < N^{1-\tau}$.*

Proof. It is quite easy to note that

$$\begin{aligned} & \frac{e}{N} - \frac{Y}{X} = \frac{eX - NY}{NX} = \frac{1 - (N - Z)Y}{NX} \approx -\frac{(N - Z)Y}{NX} \\ \Rightarrow \quad & \left|\frac{e}{N} - \frac{Y}{X}\right| \approx \left|\frac{(N - Z)Y}{NX}\right| = \frac{N^\tau Y}{NX} = \frac{N^{\tau-1}Y}{X}. \end{aligned}$$

So, $\frac{Y}{X}$ will be one of the convergents of $\frac{e}{N}$ if $\frac{N^{\tau-1}Y}{X} < \frac{1}{2X^2} \Leftrightarrow 2XY < N^{1-\tau}$. ■

We will use the above result later to demonstrate certain improvements over existing schemes. Next we present the following theorem which is the core of our results. For detailed ideas related to lattices, one may refer back to Chapter 2 or have a look at [14, 15].

Theorem 3.2. *Let $N = pq$ be the RSA modulus. Consider that e $(= N^\alpha)$ satisfies the equation $eX - ZY = 1$ where $|N - Z| = N^\tau$, and $|Y| = N^\gamma$. Then we can apply LLL algorithm to get Z efficiently when*

$$\gamma < 4\alpha\tau\left(\frac{1}{4\tau} + \frac{1}{12\alpha} - \sqrt{\left(\frac{1}{4\tau} + \frac{1}{12\alpha}\right)^2 + \frac{1}{2\alpha\tau}\left(\frac{1}{12} + \frac{\tau}{24\alpha} - \frac{\alpha}{8\tau}\right)}\right).$$

Proof. We have $eX - ZY = 1$, which can also be written as $eX = 1 + NY + (Z - N)Y$. Hence, $1 + NY + (Z - N)Y = 0 \bmod e$. Thus, we have to find the solution of $f(x, y) = 1 + Nx + xy$ in $\mathbb{Z}_e$, where $x = Y, y = Z - N$ (the unusual assignment of Y to x is to maintain similar notation as in [14] in the rest of the proof).

We have to find x, y such that $1 + x(N + y) \equiv 0 \pmod e$, where $|x| = N^\gamma = e^{\frac{\gamma}{\alpha}}$ and $|y| = N^\tau = e^{\frac{\tau}{\alpha}}$. Let $X_1 = e^{\frac{\gamma}{\alpha}}, Y_1 = e^{\frac{\tau}{\alpha}}$. One may refer to [14, Section 4] for

$$\begin{aligned} \det{}_x &= e^{m(m+1)(m+2)/3} \cdot X_1^{m(m+1)(m+2)/3} \cdot Y_1^{m(m+1)(m+2)/6}, \\ \det{}_y &= e^{tm(m+1)/2} \cdot X_1^{tm(m+1)/2} \cdot Y_1^{t(m+1)(m+t+1)/2}. \end{aligned}$$

Plugging in the values of X_1 and Y_1, we obtain the following.

$$\det{}_x = e^{m^3\cdot\left(\frac{1}{3}+\frac{\gamma}{3\alpha}+\frac{\tau}{6\alpha}\right)+o\left(m^3\right)},$$
$$\det{}_y = e^{tm^2\cdot\left(\frac{1}{2}+\frac{\gamma}{2\alpha}+\frac{\tau}{2\alpha}\right)+\frac{mt^2\tau}{2\alpha}+o\left(tm^2\right)}.$$

Now $\det(L) = \det_x \det_y$ and we need to satisfy $\det(L) < e^{mw}$, where $w = (m+1)(m+2)/2 + t(m+1)$, the dimension of L. To satisfy $\det(L) < e^{mw}$, we need $m^3\cdot\left(\frac{1}{3}+\frac{\gamma}{3\alpha}+\frac{\tau}{6\alpha}\right)+tm^2\cdot\left(\frac{1}{2}+\frac{\gamma}{2\alpha}+\frac{\tau}{2\alpha}\right)+\frac{mt^2\tau}{2\alpha} < \frac{m^3}{2}+tm^2$, ignoring the smaller terms. This leads to $m^2\cdot\left(\frac{1}{3}+\frac{\gamma}{3\alpha}+\frac{\tau}{6\alpha}-\frac{1}{2}\right)+tm\cdot\left(\frac{1}{2}+\frac{\gamma}{2\alpha}+\frac{\tau}{2\alpha}-1\right)+\frac{t^2\tau}{2\alpha} < 0$. After fixing an m, the left hand side is minimized at $t = m\left(\frac{\alpha}{2\tau}-\frac{1}{2}-\frac{\gamma}{2\tau}\right)$. Putting this value we have, $\left(\frac{1}{12}+\frac{\tau}{24\alpha}-\frac{\alpha}{8\tau}\right)+\gamma\cdot\left(\frac{1}{4\tau}+\frac{1}{12\alpha}\right)-\frac{\gamma^2}{8\alpha\tau} < 0$. So,

$$\gamma < 4\alpha\tau\left(\frac{1}{4\tau}+\frac{1}{12\alpha}-\sqrt{\left(\frac{1}{4\tau}+\frac{1}{12\alpha}\right)^2+\frac{1}{2\alpha\tau}\left(\frac{1}{12}+\frac{\tau}{24\alpha}-\frac{\alpha}{8\tau}\right)}\right).$$

Similar to the idea in [14, Section 4], if the first two elements (polynomials $P_1(x,y), P_2(x,y)$) of the reduced basis out of the LLL algorithm are algebraically independent (i.e., nonzero resultant $R(P_1,P_2)$ which is a polynomial of y, say), then we get y by solving $R(P_1,P_2) = 0$. The value of y gives $Z - N$. (This actually happens in practice as we have also checked by experimentation.) ■

Based on Theorem 3.2, one can design a probabilistic polynomial time algorithm $\mathcal{A}$, which takes $N, e = N^\alpha$ as inputs and provides the correct Z if

- $eX - ZY = 1$, with $|N - Z| = N^\tau$, and $Y = N^\gamma$, where
$$\gamma < 4\alpha\tau\left(\frac{1}{4\tau}+\frac{1}{12\alpha}-\sqrt{\left(\frac{1}{4\tau}+\frac{1}{12\alpha}\right)^2+\frac{1}{2\alpha\tau}\left(\frac{1}{12}+\frac{\tau}{24\alpha}-\frac{\alpha}{8\tau}\right)}\right),$$
- and the resultant polynomial $R(P_1,P_2)$ on y is nonzero with integer solution.

If Z is known, one can try to get further information on the primes. In [96], $Z = \psi(p,q,u,v) = (p-u)(q-v)$ and this knowledge presents a class of weak keys in RSA. In our analysis (Section 3.3), we use $Z = \psi(p,q,u,v) = N - pu - v$.

We would now like to list the following points in this direction.

- For a fixed α, the value of γ decreases when τ increases, and
- given a fixed τ, the value of γ increases as α increases.

In Table 3.1, we present the numerical values of γ corresponding to α following Theorem 3.2 for three different values of τ, namely $\frac{1}{4}, \frac{1}{2}$ and $\frac{3}{4}$.

α	1	1.1	1.2	1.3	1.4	1.5	1.6	1.7	1.8	1.875
$\tau = \frac{1}{4}$	0.482	0.555	0.629	0.704	0.780	0.856	0.934	1.012	1.091	1.150
$\tau = \frac{1}{2}$	0.284	0.347	0.412	0.477	0.544	0.612	0.681	0.751	0.821	0.875
$\tau = \frac{3}{4}$	0.131	0.188	0.245	0.305	0.365	0.427	0.489	0.553	0.618	0.667

Table 3.1: The numerical upper bounds of γ (in each cell) following Theorem 3.2, given different values of α and τ.

In the work of [96], the value of τ has been taken as $\frac{1}{2}$. Thus we discuss some cases when $\tau = \frac{1}{2}$ to highlight the improvements we achieve over [96]. Note that for randomly chosen e with $e < \phi(N)$, the value of e will be $O(N)$ in most of the cases. In such a case, putting $\alpha = 1$ and $\tau = \frac{1}{2}$, we get that $\gamma < 0.284$.

When $\alpha < 1$ and $\tau = \frac{1}{2}$, the bound on γ will decrease and it will become 0 at $\alpha = \frac{1}{2}$. However, for randomly chosen e with $e < \phi(N)$, this will happen in negligibly small proportion of cases.

Most interestingly, the bound of γ will increase further beyond 0.284 when $\alpha > 1$. Wiener's attack [130] becomes ineffective when $e > N^{1.5}$ and the attack proposed by Boneh and Durfee [15] becomes ineffective when $e > N^{1.875}$. Similar to the result of [15], equations of the form $eX - ZY = 1$ cannot be used for $e > N^{1.875}$, since in such case no X will exist given the bound on Y. For $\tau = \frac{1}{2}$, we have presented the theoretical results for e reaching $N^{1.875}$ in Table 3.1. Experimental results will not reach this bound as we work with small lattice dimensions in practice, but even then the experimental results for e reach close to the value $N^{1.875}$ as we demonstrate results for $N^{1.774}$ in Section 3.2.3 for 1000-bit N.

Note that here we present a theoretical estimate on the bound of γ. These bounds may not be achievable in practice due to the large lattice dimensions. However, the experimental results, presented in Sections 3.2.3, 3.3.1, are close to the theoretical estimates.

3.2 Improvements over Existing Work

In this section we present various improvements over the work of [96]. For this, first we present an outline of the strategy in [96]. Consider that e satisfies $eX - (p-u)(q-v)Y = 1$ with $1 \leq Y < X < 2^{-\frac{1}{4}}N^{\frac{1}{4}}$, $|u| < N^{\frac{1}{4}}$, $v = \left[-\frac{qu}{p-u}\right]$. If all the prime factors of $p-u$ or $q-v$ are less than 10^{50}, then N can be factored from the knowledge of N, e. The number of such weak exponents are estimated as $N^{\frac{1}{2}-\epsilon}$. The flow of the algorithm in [96] is as follows.

1. Continued Fraction algorithm is used to find the unknowns X, Y among the convergents of $\frac{e}{N}$.
2. Then, the Elliptic Curve Factorization Method (ECM [79]) is used to partially factor $\frac{eX-1}{Y}$, i.e., into the factors $(p-u)(q-v)$.
3. Next, an integer relation detection algorithm (LLL [77]) is used to find the divisors of B_{ecm}-smooth part of $\frac{eX-1}{Y}$ in a small interval.
4. Finally, if $p-u$ or $q-v$ is found, the method due to [24] is applied.

After knowing $(p-u)(q-v)$, if one gets the factorization of $p-u$ or $q-v$, then it is possible to identify $p-u$ or $q-v$ efficiently and the overall complexity is dominated by the time required for factorization. According to [96], if ECM [79] is used for factorization, and if all prime factors of $p-u$ or $q-v$ are less than 10^{50}, then getting $p-u$ or $q-v$ is possible in moderate time. Once $p-u$ or $q-v$ is found, as u, v are of the order of $N^{\frac{1}{4}}$, using the technique of [24], it is possible to find p or q efficiently.

3.2.1 The Improvement in the Bounds of X, Y

In [96] the bounds of X and Y are given as $1 \leq Y < X < 2^{-\frac{1}{4}}N^{\frac{1}{4}}$. Since, u, v are of $O(N^{\frac{1}{4}})$, we get that $(p-u)(q-v)$ is $O(N)$. When e is $O(N^{1+\mu})$, $\mu > 0$ and X is $O(N^{\nu})$, $0 < \nu \leq \frac{1}{4}$, the value of eX is $O(N^{1+\mu+\nu})$. In such a case, Y will be $O(N^{\mu+\nu})$, which is not possible as $Y < X$. Thus the values of e are bounded by $O(N)$ in the work of [96]. Next we generalize the bounds on X, Y.

The method of [96] requires $1 \leq Y < X < 2^{-\frac{1}{4}}N^{\frac{1}{4}}$. For $\tau = \frac{1}{2}$, our result in Lemma 3.1 implies that it is enough to have $2XY < N^{\frac{1}{2}}$, which gives better bounds than [96] due to the following reasons.

- There is no need to have $Y < X$ when X, Y are of $O(N^{\frac{1}{4}})$.
- The bound of either X or Y can be greater than $N^{\frac{1}{4}}$ when the other one is less than $N^{\frac{1}{4}}$.

We have already noted that the values of e are bounded by $O(N)$ in the work of [96]. In our case, this bound is increased as well. The exponent e is of the order of $\frac{(p-u)(q-v)\cdot Y}{X}$, which is $O\left(\frac{N^{2-\tau}}{X^2}\right)$. Given $\tau = \frac{1}{2}$, when X is $O(N^{\frac{1}{4}})$, we get the bound on e as $O(N)$, which is same as what given in [96]. However, our bound increases as X decreases.

3.2.2 Further Improvement in the Bounds

With our results in Theorem 3.2, we get further bounds on X, Y. Note that $Y = N^{\gamma}$ and thus $X = \lceil N^{1+\gamma-\alpha} \rceil$. This gives $XY = N^{1+2\gamma-\alpha}$, where

$$\gamma < 4\alpha\tau \left(\frac{1}{4\tau} + \frac{1}{12\alpha} - \sqrt{\left(\frac{1}{4\tau} + \frac{1}{12\alpha}\right)^2 + \frac{1}{2\alpha\tau}\left(\frac{1}{12} + \frac{\tau}{24\alpha} - \frac{\alpha}{8\tau}\right)} \right),$$

as in Theorem 3.2. Our result implies the following improvements.

- The bound on Y can be extended till N^{γ}. One may have a look at Table 3.1 for the numerical values of the theoretical bounds of γ given some α. The experimental results are presented in Section 3.2.3.
- X has to satisfy the equation $eX - (p-u)(q-v)Y = 1$, which gives $X = \lceil N^{1+\gamma-\alpha} \rceil$. The value of X becomes smaller as α increases.
- The constraint $1 \leq Y < X < 2^{-\frac{1}{4}} N^{\frac{1}{4}}$ in [96] forces that the upper bound of e has to be $O(N)$. However, in our case the value of e can exceed this bound and the result works for e up to $N^{1.875}$ theoretically, as described in Section 3.1.

3.2.3 Experimental Results

Let us start with a complete example, as follows.

Example 3.3. Let us consider that N, e are available. We choose a 1000-bit N which is a product of two 500-bit primes. Let N be
6965301839116252842151601289304534942713324759565286529653767647876815
5930430666799437161475057925109785426932854120387907825516676039893976
7348434594081678022491354913429428712241705242402188329351298522684586
6182664930993217767070732004829330778668319338402258431072292258308654
308889461963963786753
and e be a 1000-bit number
6131104587526715137731121962949288372067853010907049196083307936858983
5495733258743040864963707793556567754437239459230699951652411404917214
6335728027441527646511562295901427592480445339976342362901246792172093
7327176882146018075507772227029755907501291937300116177647310527785231
764272675507839815927.

Running our method as explained in Theorem 3.2, we get $(p-u)(q-v)$ as
6965301839116252842151601289304534942713324759565286529653767647876815
5930430666799437161475057925109785426932854120387907825516676039893976
7348434593866189401923798371559862126676390152959012118814413687219025
3575242273886606436919626785397201816501702315901845767585961196177847
672535848282542000103.

The factorization of $(p-u)(q-v)$ is as follows:
$3^4 \times 17^{24} \times 61681^{24} \times$ 2297 $\times$ 141803 $\times$ 345133 $\times$ 14127457182016070043858828063633385965304567483679 $\times$ 17344035871618527481610408844179925763450067598688121780826850377496712904460130651142191039.

This requires 112151.62 seconds (less than 1.3 days) using the ECM method of factoring. In this case, $p-u$ is $3 \times 17^{24} \times 61681^{24} \times 345133$ and the rest of the terms will give $q-v$. Since, $p-u < N^{\frac{1}{4}}$, p can be found using the idea of [24] in polynomial time. Finally, one can find p, q as follows.
3232329308513348128043498783477144091769556249463616612811415899259655
9541241660031449960551292345720627446011227767334525515385020845432088
00320998727,
2154886205675565418695665855651429394513037499642411075207610599506437
3027494440020966640367528230480418297340818511556350343590013896954725
33547333239 respectively.

In this example, X, Y are respectively 275 and 274 bit numbers as follows.
3035420144102701673311659229411748291628760686018968001955956890217037

9456331382787 and
2671883975804894456278842490580443758950128272095600125097638973022749
4745205672316.

Note that these numbers are clearly greater than $N^{\frac{1}{4}}$ (in contrary to the bound presented in [96] where $1 \leq Y < X < 2^{-\frac{1}{4}}N^{\frac{1}{4}}$) as N is a 1000 bit integer here.

Now we show that the technique of [96] will not work here. We calculate the CF expansion of $\frac{e}{N}$ and study all the convergents $\frac{Y}{X}$ with denominator $X < 2^{-\frac{1}{4}}N^{\frac{1}{4}}$. Except $X = Y = 1$, no $\frac{eX-1}{Y}$ is an integer. When $X = Y = 1$, we have $\frac{eX-1}{Y} = e-1$. Thus in this case, $(p-u)(q-v) = e-1$. As given in [96, Lemma 4], one needs to satisfy the condition $|(p-u)(q-v) - N| < 2^{-\frac{1}{2}}N^{\frac{1}{2}}$. Thus, in this example, one needs to satisfy $|e - 1 - N| < 2^{-\frac{1}{2}}N^{\frac{1}{2}}$, which is not true.

Then we attempted different cases with $e = N^{\alpha}$ having varying α, given the same p, q as in Example 3.3. The experimental results are as shown in Table 3.2 where each run to find $(p-u)(q-v)$ requires less than 15 minutes.

α	1	1.1	1.2	1.3	1.4	1.5	1.6	1.7	1.8	1.875
γ	0.274	0.336	0.399	0.464	0.529	0.596	0.659	0.726	–	–

Table 3.2: The numerical values of γ given α found by experiment when N is of 1000 bits and p, q are as in Example 3.3. The lattice has the parameters $m = 7, t = 3, w = 60$.

Note that compared to Table 3.1, the results in Table 3.2 gives slightly lower values of γ. Further, we do not get the solutions for these p, q values when $\alpha = 1.8, 1.875$ as $1+\gamma-\alpha$ becomes very close to zero and hence X does not exist given the bound of Y. The maximum e for which we get a valid X is of size 1774 bits in this example. Thus the maximum value of α for which our method works in this example is 1.774. The value of γ in this example is 0.778 as Y is a 778-bit integer.

3.3 A New Class of Weak Keys

The problem with the idea of [96] is that one needs to factorize $(p-u)(q-v)$ in order to attack RSA, and this is only possible when the factors of either $(p-u)$ or $(q-v)$ are relatively small. In this section we present a new class of weak keys where there is no involvement of factorization at all. For this, let us first refer to the following existing result from [83, Theorem 10].

Theorem 3.4 (from [83])**.** *Let $N = pq$ be an RSA modulus with $q < p < 2q$. Let u be an (unknown) integer that is not a multiple of q. Suppose we know an approximation $\hat{P}$ of pu with $|pu - \hat{P}| \leq 2N^{\frac{1}{4}}$. Then N can be factorized in time polynomial in* $\log N$*.*

After finding X and Y from the continued fraction expression (as given in Lemma 3.1 of $\frac{e}{N}$, one can get $N - pu - v$ and hence $pu + v$. In our case, the $\hat{P}$ of Theorem 3.4 is $pu + v$. Following Theorem 3.4, if $|v| < N^{\frac{1}{4}}$ and u is not a multiple of q, then one can get p.

Next we present further extension on this class of weak keys using the idea of Theorem 3.2 and noting $Z = \psi(p, q, u, v) = pq - pu - v = N - pu - v$. In this case, $|N - Z| = |pu + v| = N^{\tau}$. When $Y = N^{\gamma}$ and $e = N^{\alpha}$ then $X = \lceil N^{1+\gamma-\alpha} \rceil$. This gives, $XY = N^{1+2\gamma-\alpha}$, where

$$\gamma < 4\alpha\tau \left(\frac{1}{4\tau} + \frac{1}{12\alpha} - \sqrt{\left(\frac{1}{4\tau} + \frac{1}{12\alpha} \right)^2 + \frac{1}{2\alpha\tau} \left(\frac{1}{12} + \frac{\tau}{24\alpha} - \frac{\alpha}{8\tau} \right)} \right),$$

as in Theorem 3.2.

It is clear that when $N^{1+2\gamma-\alpha}$ is greater than the bound $\frac{N}{pu+v} = N^{1-\tau}$ of Lemma 3.1, then we get a larger class of weak keys using the LLL method instead of the technique using continued fraction expression. This happens when $1 + 2\gamma - \alpha > 1 - \tau$, which is true taking the value of the upper bound on γ.

For example, taking $\alpha = 1$ and $\tau = \frac{1}{2}$ the upper bound of γ is 0.284 and in this case XY will be $N^{0.568}$. However, the bound from Lemma 3.1 in this case is $N^{0.5}$. For larger values of α, the bound on XY will increase further.

3.3.1 Experimental Results

We consider the same N as used in Example 3.3.

Example 3.5. Let us first apply the continued fraction method as explained in Lemma 3.1. The public exponent e is a 1000-bit number
69241917949048224443319199880656758349580894787556040212244865507412869
094970482880973033565767568702443953304958933817087359916417417403418

8975911152337295179208385986195123683049905106852500834430037397316270
0864249336588337032797599912676619611066951994203277539744143449608166
337312588384073377751.

Then the continued fraction of $\frac{e}{N}$ gives X, Y (respectively) as
16849966666969149871666884429387269171023215264087857800689756405 79,
1675051614915381290330108388747571693885770140577513454985303531396.

Then we find $pu + v$ as
1455711706935944763346481078909022402000177527246411623972976816582783
4476154028615829072068297700713978442098223807592204425878660324320671
085270850022954001141596160.

From $pu + v$ we get p using the idea of Theorem 3.4 as in this case $u = 2^{52}$ is not a multiple of q and $v = 2^{175}$ is less than $N^{\frac{1}{4}}$.

Next we give an example using the LLL technique that cannot be done using the technique of Lemma 3.1.

Example 3.6. The public exponent e is a 1000-bit integer as follows.
6875161700303375704546089777254797601588535353918071259131130001533282
9990657553375889250468602806539681604179662083731636763206733203004319
5254536198882701786034369526609680993429591977911304810695130485689008
4599364131003915783164223278468592950590533634401668968574079388141851
069809468480532614755.

Using Theorem 3.2 we get Y (a 240 bit integer) as follows.
1743981747042138853816214839550693531666858348727675018411981662762169
193.

We also get $pu + v$ (a 552 bit integer) as follows.
1455711706935944763346481078909022402000177527246411623972976816582783
4476154028615829072068297700713978442098223807592204425878660324320671
085270850022954001141596160.

In this case u, v are same as in Example 3.5 and hence p can be found using the idea of Theorem 3.4. It is to note that in this case X is a 241 bit integer
1766847064778384329583297500742918515827483896875618958121606 2012
92619790.

Note that $\lceil \frac{N}{pu+v} \rceil$ is a 448-bit integer. Now $2XY$ should be less than 448 bit for a success using the technique of Lemma 3.1, which is not possible as X, Y are

241 and 240 bit integers respectively. Thus the idea of continued fraction method can not be applied here.

Now we like to point out that the weak key of Example 3.6 is not covered by the works of [9,31,130]. In this case, the decryption exponent d is a 999-bit integer and hence the bound of [130] that $d < \frac{1}{3}N^{\frac{1}{4}}$ will not work here.

Here $p - q > N^{0.48}$ and according to [31, Section 6] one can consider $\beta = 0.48$. If $d = N^{\delta}$, then according to [31, Section 6] the bound of δ will be $2-4\beta < \delta < 1-\sqrt{2\beta - \frac{1}{2}}$ for RSA to be insecure. Putting $\beta = 0.48$, one can get $0.08 < \delta < 0.3217$, which is much smaller than 0.999 (in Example 3.6, $d \approx N^{0.999}$) and hence the weak keys of [31] does not cover our result.

In [9, Theorem 4, Section 4], it has been shown that p, q can be found in polynomial time for every N, e satisfying $ex + y = k\phi(N)$, with $0 < x \leq \frac{1}{3}\sqrt{\frac{\phi(N)}{e}\frac{N^{\frac{3}{4}}}{p-q}}$ and $|y| \leq \frac{p-q}{\phi(N)N^{\frac{1}{4}}}ex$. According to [9], convergents of the CF expression of $\frac{e}{N-\lfloor 2\sqrt{N}\rfloor}$ will provide $\frac{k}{x}$. For the parameters in Example 3.6, we calculated all the convergents with $x \leq \frac{1}{3}\sqrt{\frac{\phi(N)}{e}\frac{N^{\frac{3}{4}}}{p-q}}$ and we find that for each such k, x, $|ex - k\phi(N)| > \frac{p-q}{\phi(N)N^{\frac{1}{4}}}ex$. As $y = ex - k\phi(N)$, the bound on $|y|$ is not satisfied. Thus the weak key presented in Example 3.6 is not covered by the work of [9].

Next, we attempt different cases with $e = N^{\alpha}$ having varying α given the same p, q in Example 3.3. As the strategy works for the values $u = 2^{52}$ and $v = 2^{175}$ (given in Example 3.5), we get $N^{\tau} = pu + v$, which implies $\tau = 0.552$. In the first row of Table 3.3, we present the values of α where $e = N^{\alpha}$; the second row provides the theoretical upper bound on γ according to Theorem 3.2, given the values of α in the first row and $\tau = 0.552$; the third row presents the experimental values of γ, which is obtained from the bitsize of Y as found in the experiments. The lattice used in these cases has the parameters $m = 7, t = 3, w = 60$.

α	1	1.1	1.2	1.3	1.4	1.5	1.6	1.7	1.72
γ (theory)	0.250	0.311	0.374	0.438	0.504	0.570	0.638	0.706	0.720
γ (exp.)	0.240	0.300	0.362	0.426	0.491	0.555	0.621	–	–

Table 3.3: Values of γ for $\tau = 0.552$, 100 bit N, and p, q as in Example 3.3.

In experiments, we do not get the solutions for these p, q values when $\alpha \geq 1.7$ as $1 + \gamma - \alpha$ becomes very close to zero and hence X does not exist given the bound of Y. The maximum e for which we get a valid X is of size 1653 bits in this example. Thus the maximum value of α for which our method works in this example is 1.653. The value of γ in such a case is 0.656 as Y is a 656-bit integer.

We like to point out that one can exploit the techniques using sublattices given in [15] for improvement in the bound of γ than in Theorem 3.2 (where we use the idea of lattices following [14]). In practice, the idea of sublattices helps in getting the same result with less lattice dimension. During actual execution, for fixed N, e, u, v, Y, consider that t_1 is the time in seconds to run the LLL algorithm, t_2 is the time in seconds to calculate the resultant and t_3 is the time in seconds to find the integer root of the resultant; and let us refer this as a tuple $\langle (l_N, l_e, l_u, l_v, l_Y), t_1, t_2, t_3\rangle^a$, where l_N, l_e, l_u, l_v, l_Y are the bitsizes of N, e, u, v, Y respectively; and $a = L$ for full rank lattice and $a = S$ for sublattice. Our examples are with lattice parameters $m = 7, t = 3$ and thereby giving the dimension 60 for full rank lattice (following the idea of [14]) and dimension 43 for sublattice (exactly following [15] the dimension should be 45, but due to the upper bounds X_1, Y_1 in Theorem 3.2, we get lower sublattice dimension). The examples are as follows:
$\langle (1000, 1000, 52, 175, 240), 20, 373, 4\rangle^L$, $\langle (1000, 1000, 52, 175, 240), 14, 377, 4\rangle^S$, $\langle (2000, 1995, 104, 350, 465), 79, 1074, 16\rangle^L$, $\langle (2000, 1995, 104, 350, 465), 68, 1075, 15\rangle^S$, $\langle (9999, 9999, 520, 1750, 2350), 4722, 5021, 248\rangle^L$, $\langle (9999, 9999, 520, 1750, 2350), 4426, 5028, 198\rangle^S$.
As long as t_1 is much less than t_2, using sublattices (following [15]) instead of lattices (following [14]) will not provide significant improvement in total execution time. However, when t_1 becomes dominant, then the implementation using sublattices will provide faster execution.

3.3.2 Estimation of Weak Keys

In this section, we estimate the number of exponents for which our method works. We first present a simple analysis.

Lemma 3.7. *Consider RSA with $N = pq$, where p, q are primes such that $q < p < 2q$. Let e be the public encryption exponent that satisfies $eX - (N - pu - v)Y = 1$. Then for $X = Y = 1$, N can be factorized in* poly(log N) *time from the knowledge of N, e when u is not a multiple of q and $|v| < N^{\frac{1}{4}}$. The number of such weak keys e, such that $e < N$ is $N^{\frac{3}{4}-\epsilon}$, where $\epsilon > 0$ is arbitrarily small for suitably large N.*

Proof. Given the equation $eX - (N - pu - v)Y = 1$, we consider the scenario when $X = 1, Y = 1$, i.e., when $e = N - pu - v + 1$. In such a case, from e, we will immediately get $pu + v$ and then following Theorem 3.4, one can get p in $O(\text{poly}(\log \text{N}))$ time when $|v| < N^{\frac{1}{4}}$ and u is not a multiple of q. Considering $1 < e < \phi(N)$, we get that $pu + v < N$. Considering p, q are of same bitsize, i.e., $q < p < 2q$, one may find $\sqrt{N} < p < \sqrt{2N}$ and $\sqrt{\frac{N}{2}} < q < \sqrt{N}$. As, $|v| < N^{\frac{1}{4}}$ and v may be a negative integer, a conservative upper bound of u is $\sqrt{\frac{N}{2}}$ and clearly in such a case u is not a multiple of q. The total number of options of u, v pairs when $0 < u < \sqrt{\frac{N}{2}}$ and $|v| < N^{\frac{1}{4}}$ is $\sqrt{\frac{N}{2}} \times 2N^{\frac{1}{4}} = \sqrt{2}N^{\frac{3}{4}}$. As we have to consider those e's which are co-prime to $\phi(N)$, we can only consider those u, v pairs such that $\gcd(N - pu - v + 1, \phi(N)) = 1$. Similar to the arguments of [9, Lemma 13] and [96, Theorem 6], the number of such u, v pairs is $N^{\frac{3}{4}-\epsilon}$, where $\epsilon > 0$ is arbitrarily small for suitably large N. ■

The result of Lemma 3.7 will actually work in a similar manner for any X, Y which are bounded by a small constant as one can search those values of X, Y pairs to guess $pu + v$. Now we discuss a more general scenario.

Consider $\tau \leq 1 - \epsilon_1$ for some arbitrarily small positive constant ϵ_1. We have $\sqrt{N} < p < \sqrt{2N}$, $pu + v = N^{\tau}$ and $|v| < N^{\frac{1}{4}}$. Thus, it is enough to consider $u \leq \frac{1}{\sqrt{2}}N^{\frac{1}{2}-\epsilon_1}$. In such a case, $N - pu - v$ will be $c_1 N$ for some constant $0 < c_1 < 1$. Considering $e = c_2 N$, with $0 < c_2 < 1$, a constant, we find that X, Y are of the same order. As we consider $e = c_2 N$, we can estimate α as 1. Following Theorem 3.2 and putting $\alpha = 1$, we find that as τ goes towards 1 (i.e., ϵ_1 goes to 0), the value of

$$4\alpha\tau\left(\frac{1}{4\tau} + \frac{1}{12\alpha} - \sqrt{\left(\frac{1}{4\tau} + \frac{1}{12\alpha}\right)^2 + \frac{1}{2\alpha\tau}\left(\frac{1}{12} + \frac{\tau}{24\alpha} - \frac{\alpha}{8\tau}\right)}\right)$$

goes towards 0. Now, γ is less than this bound, and $|Y| = N^{\gamma}$. Hence, given that X, Y are of the same order, this puts a constraint on X as well, and we need to consider $X < N^{\epsilon_3}$ where $\alpha = 1$, $\tau = 1 - \epsilon_1$, and

$$\epsilon_3 = 4\alpha\tau\left(\frac{1}{4\tau} + \frac{1}{12\alpha} - \sqrt{\left(\frac{1}{4\tau} + \frac{1}{12\alpha}\right)^2 + \frac{1}{2\alpha\tau}\left(\frac{1}{12} + \frac{\tau}{24\alpha} - \frac{\alpha}{8\tau}\right)}\right).$$

Let us provide some computational results in this direction. Table 3.4 shows some numerical values of ϵ_3 (i.e., the bound of γ) following Theorem 3.2, corresponding to different values of τ when $\alpha = 1$.

τ	0.5	0.6	0.7	0.75	0.8	0.85	0.9	0.95	0.97	0.99
ϵ_3	0.284	0.220	0.160	0.130	0.100	0.077	0.051	0.025	0.015	0.005

Table 3.4: Numerical values of ϵ_3 following Theorem 3.2 where $\alpha = 1$.

Let us refer to the following result from [41] due to Ford and Tenenbaum.

Theorem 3.8 (from [41]). *Consider a large integer N that has a divisor d_1 in an interval $(y, z]$ for a large y. Then N has exactly one divisor (almost certainly) in this interval if $z \approx y + \frac{y}{(\log y)^{\log 4 - 1}}$.*

Now we can present a technical result required for counting the weak keys.

Lemma 3.9. *Let $N = pq$ with $q < p < 2q$ and $0 \leq u, u' \leq \frac{1}{\sqrt{2}} N^{\frac{1}{2} - \epsilon_1}$. Let X be an integer with $1 \leq X < N^{\epsilon_3}$ such that $\gcd(X, N - pu - v) = 1$ and $\gcd(X, N - pu' - v') = 1$, where $|v|, |v'| < N^{\frac{1}{4}}$. Let $e \equiv X^{-1} \pmod{N - pu - v}$ and $e' \equiv X^{-1} \pmod{N - pu' - v'}$. If $e = e'$ then almost surely $u = u', v = v'$.*

Proof. We have $e = e'$. So $N - pu - v$ and $N - pu' - v'$ both divides $eX - 1$. Since $u, u' \leq \frac{1}{\sqrt{2}} N^{\frac{1}{2} - \epsilon_1}$, and $|v|, |v'| < N^{\frac{1}{4}}$ so $N - pu - v$ and $N - pu' - v'$ are in the interval $I = \left[N - N^{1-\epsilon_1} - N^{\frac{1}{4}}, N + N^{\frac{1}{4}}\right]$. Let $y = N - N^{1-\epsilon_1} - N^{\frac{1}{4}}$ and $y' = N + N^{\frac{1}{4}}$. One can check that $y' < y + \frac{y}{(\log y)^{\log 4 - 1}}$ for a fixed ϵ_1 and a large N. Thus following Theorem 3.8, $N - pu - v = N - pu' - v'$ holds almost surely.

Given $N - pu - v = N - pu' - v'$, we get $pu + v = pu' + v'$ if and only if $p(u - u') = v' - v$. Since p is $O(\sqrt{N})$ and v, v' are $O(N^{\frac{1}{4}})$, we get $u = u'$ and $v = v'$. ■

Thus if any one (or both) of the pairs u, u' and v, v' are distinct, then e, e' are almost surely distinct once X is fixed. The number of such distinct u, v pairs is $\frac{1}{\sqrt{2}} N^{\frac{3}{4} - \epsilon_1}$. Fixing X, for each pair of values of u, v, the exponent e need to be co-prime to $\phi(N)$. Similar to the arguments of [9, Lemma 13] and [96, Theorem 6], the number of such u, v pairs is $\frac{1}{\sqrt{2}} N^{\frac{3}{4} - \epsilon_1 - \epsilon_2}$, where $\epsilon_2 > 0$ is arbitrarily small for suitably large N. Once again, we like to highlight the constraint that $X < N^{\epsilon_3}$, where the value of ϵ_3 goes towards 0 as ϵ_1 goes to zero. Following Table 3.4, we get that when ϵ_1 is 0.01, then $\epsilon_3 = 0.005$. Thus, in this case, for any $X < N^{0.005}$, we get approximately $\frac{1}{\sqrt{2}} N^{0.74 - \epsilon_2}$ many weak keys.

Note that, the problem becomes more complicated when we consider the set of weak keys for two unequal values of X, say X', X''. Let $H_{X'}$ and $H_{X''}$ be two different sets of weak keys for $X' \neq X''$. However, it is not clear what is the intersection between $H_{X'}$ and $H_{X''}$. If it can be shown that for distinct values of X, the corresponding sets H_X does not have quite a large intersection, then the total number of weak keys will be much higher than what demonstrated in this section. Further in this section we have only considered $e < N$. As the result of Theorem 3.2 shows that our technique can be applied for $e > N$ too, that will provide much larger class of weak keys.

3.4 Conclusion

In this chapter we present some new weak keys of RSA. We study the public exponents e $(= N^{\alpha})$ when $eX - ZY = 1$ under the constraint $|N - Z| = N^{\tau}$. We show that the LLL algorithm can be efficiently applied to get $Z = \psi(p, q, u, v)$ when $|Y| = N^{\gamma}$ and

$$\gamma < 4\alpha\tau \left(\frac{1}{4\tau} + \frac{1}{12\alpha} - \sqrt{\left(\frac{1}{4\tau} + \frac{1}{12\alpha}\right)^2 + \frac{1}{2\alpha\tau}\left(\frac{1}{12} + \frac{\tau}{24\alpha} - \frac{\alpha}{8\tau}\right)} \right).$$

Some specific forms of $\psi(p, q, u, v)$ are then studied. We improve the results of [96] taking $\psi(p, q, u, v) = (p-u)(q-v)$. Further, we consider $\psi(p, q, u, v) = N - pu - v$ to provide a new class of weak keys in RSA. A very preliminary analysis of this class shows that the number of weak keys is at least $N^{0.75-\epsilon}$ and it seems that a better analysis will provide a better bound than this. Further, the study of different forms of $\psi(p, q, u, v)$ may provide additional weak keys that are not known till date.

Following a similar line of thought, one may wonder if there exist any weaknesses associated with the RSA decryption exponent. A lot of research in cryptanalysis of RSA has been done in this direction, and there exist several results. In the next chapter, we study a special vulnerability of RSA in the case when more than one decryption exponents are used with the same modulus.

Chapter 4

Cryptanalysis of RSA with more than one Decryption Exponent

From the work of Boneh and Durfee [14], we know that one can factor N in polynomial time when $d < N^{0.292}$. Instead of one decryption exponent, consider that n many decryption exponents $(d_1, \ldots, d_n)$ are used with the same N. Let $(e_1, e_2, \ldots, e_n)$ be their corresponding public exponents. As explained in [60, Page 121] such a situation can arise if a person is using the same RSA modulus N, but different exponents d_i to sign different messages. It has been shown by Howgrave-Graham and Seifert [62] that in case of n many decryption exponents, one can factor N efficiently when $d_i < N^{\delta}$, for $1 \leq i \leq n$, where

$$\delta \; < \begin{cases} \dfrac{(2n+1)\cdot 2^n - (2n+1)\binom{n}{n/2}}{(2n-2)\cdot 2^n + (4n+2)\binom{n}{n/2}} & \text{if } n \text{ is even.} \\[2ex] \dfrac{(2n+1)\cdot 2^n - 4n\cdot\binom{n-1}{(n-1)/2}}{(2n-2)\cdot 2^n + 8n\cdot\binom{n-1}{(n-1)/2}} & \text{if } n \text{ is odd.} \end{cases} \tag{4.1}$$

However, Hinek et al [55, Section 5] proved that one needs to satisfy another condition for the idea of [62] to work. That condition makes the upper bound of decryption exponents $d_i < \sqrt{N}$ for $1 \leq i \leq n$.

We show in this chapter that if n many decryption exponents $(d_1, \ldots, d_n)$ are used with the same N, then RSA is insecure when $d_i < N^{\frac{3n-1}{4n+4}}$, for $1 \leq i \leq n$ and $n \geq 2$. Our result improves the bound of Howgrave-Graham and Seifert [62]. The time complexity of our technique as well as that of [62] is polynomial in the bitsize of N and exponential in the number of decryption exponents n. Putting $n = 2$, we find $\delta < 0.416$. However, for this special case of $n = 2$, we show that our strategy can extend the bound till $\delta < 0.422$. Our result has another component that it takes care of the case when some of the most significant bits (MSBs) of the decryption exponents are same (but unknown). This implicit information increases the bounds on the decryption exponents even further. We present experimental results to support our claim. As explained in the introduction of [62], we also agree that studying this kind of cryptanalysis may not have direct impact to RSA used in practice. However, there are few issues for which this problem is interesting.

- This shows how one can find further weaknesses of RSA with additional public information – in this case more than one encryption exponents.
- Moreover, this shows how one can extend the ideas of [15,130], where a single encryption exponent is considered, to more than one exponents.

4.1 Theoretical Result

We need the following technical result that will be used later. A general treatment in this direction is available in [63, Theorem 1, Page 230].

Lemma 4.1. *For any fixed positive integer $r \geq 1$, and a large integer m,*

$$\sum_{t=1}^{m} t^r = \frac{m^{r+1}}{r+1} + o(m^{r+1}).$$

Proof. Let $S = 1^r + 2^r + \ldots + m^r$. Then we have

$$\begin{aligned} \int_0^m x^r\,dx \ &< \ S \ < \ \int_1^{m+1} x^r\,dx \\ \Rightarrow \ \frac{m^{r+1}}{r+1} \ &< \ S \ < \ \frac{(m+1)^{r+1}-1}{r+1} \\ \Rightarrow \ \frac{m^{r+1}}{r+1} \ &< \ S \ < \ \frac{(m+1)^{r+1}}{r+1}. \end{aligned}$$

Now, $\frac{(m+1)^{r+1}}{r+1} - \frac{m^{r+1}}{r+1}$ contains the terms m^i for $i \leq r$. Thus, for a fixed r and large m, one can write

$$S = \frac{m^{r+1}}{r+1} + o(m^{r+1}).$$

Hence the required result. ∎

Now we may proceed to our main result for this section.

Theorem 4.2. *Suppose that $n \geq 2$ and $(e_1, \ldots, e_n)$ are n RSA encryption exponents with common modulus N. Also suppose that $\gcd(e_i, e_j) = 1$ for all $i \neq j$ where $1 \leq i, j \leq n$. Let $d_1, \ldots, d_n$ be the corresponding decryption exponents with $d_1, \ldots, d_n < N^\delta$. Then, under Assumption 1, one can factor N in* $\mathrm{poly}\{\log \mathrm{N}, \exp(\mathrm{n})\}$ *time if*

$$\delta < \begin{cases} 0.422 & \text{for } n = 2 \\ \dfrac{3n-1}{4n+4} & \text{for } n \geq 3 \end{cases}$$

Proof. We have, $e_1 d_1 = 1 + k_1(N + r)$, $e_2 d_2 = 1 + k_2(N + r)$, …, $e_n d_n = 1 + k_n(N+r)$, where $r = -p-q+1$. Let $\prod_{i=1}^n e_i = E$. Multiplying the n equations by $\frac{E}{e_1}, \frac{E}{e_2}, \ldots, \frac{E}{e_n}$ respectively, and then subtracting all the other equations from the first one, we get

$$E\left(d_1 - \sum_{i=2}^n d_i\right) = \frac{E}{e_1} - \sum_{j=2}^n \frac{E}{e_j} + (N+r)\left(\frac{E}{e_1}k_1 - \sum_{j=2}^n \frac{E}{e_j}k_j\right).$$

Now, we want to find a solution $(d_1 - d_2 - \cdots - d_n, k_1, \ldots, k_n, r)$ of the polynomial

$$f(x_1, x_2, \ldots, x_{n+2}) = Ex_1 - \left(\frac{E}{e_1} - \sum_{j=2}^n \frac{E}{e_j}\right) - (N + x_{n+2})\left(\frac{E}{e_1}x_2 - \sum_{j=2}^n \frac{E}{e_j}x_{j+1}\right).$$

Note that $f(x_1, x_2, \ldots, x_{n+2})$ is an irreducible polynomial as the encryption exponents are relatively prime. Since $d_i < N^\delta$ for $1 \leq i \leq n$, $|d_1 - \sum_{i=2}^n d_i| < N^\delta$, treating n as a constant and neglecting it. Also, we have $|r| < \left(1+\sqrt{2}\right)N^{\frac{1}{2}}$ and $k_i < N^\delta$ for $1 \leq i \leq n$. Let $X_1 = X_2 = \cdots = X_{n+1} = N^\delta$ and $X_{n+2} = N^{\frac{1}{2}}$. Then $X_1, X_2, \ldots, X_{n+2}$ are the upper bounds of $\left(d_1 - \sum_{i=2}^n d_i\right), k_1, \ldots, k_n, r$ respectively, neglecting constant terms. Using the extended strategy of Section 2.6.2, we define

$$S = \bigcup_{0 \leq j \leq t} \{x_1^{i_1} x_2^{i_2} \cdots x_{n+2}^{i_{n+2}+j} : x_1^{i_1} x_2^{i_2} \cdots x_{n+2}^{i_{n+2}} \text{ is a monomial of } f^m\}$$

$$M = \{\text{monomials of } x_1^{i_1} x_2^{i_2} \cdots x_{n+2}^{i_{n+2}} \cdot f : x_1^{i_1} x_2^{i_2} \cdots x_{n+2}^{i_{n+2}} \in S\},$$

where t is a non-negative integer. Thus we get the following:

$$x_1^{i_1}x_2^{i_2}\cdots x_{n+2}^{i_{n+2}} \in S \Leftrightarrow \begin{cases} i_1 = 0,\ldots,m \\ i_2 = 0,\ldots,m-i_1 \\ \vdots \\ i_{n+1} = 0,\ldots,m-i_1-\cdots-i_n \\ i_{n+2} = 0,\ldots,i_2+\cdots+i_{n+1}+t \end{cases}$$

$$x_1^{i_1}x_2^{i_2}\cdots x_{n+2}^{i_{n+2}} \in M \Leftrightarrow \begin{cases} i_1 = 0,\ldots,m+1 \\ i_2 = 0,\ldots,m+1-i_1 \\ \vdots \\ i_{n+1} = 0,\ldots,m+1-i_1-\cdots-i_n, \\ i_{n+2} = 0,\ldots,i_2+\cdots+i_{n+1}+t. \end{cases}$$

Apart from f, we need to find at least $n+1$ more polynomials $f_1, f_2, \ldots f_{n+1}$ that share the same root $(d_1 - d_2 - \cdots - d_n, k_1, \ldots, k_n, r)$ over the integers.

Considering a small fixed n, we know that these polynomials can be found by LLL [77] algorithm in poly(log N) time if

$$X_1^{s_1}X_2^{s_2}\cdots X_{n+2}^{s_{n+2}} < W^s$$

for $s_j = \sum_{x_1^{i_1}\cdots x_{n+2}^{i_{n+2}} \in M\setminus S} i_j$ with $j = 1,\ldots,n+2$, $s = |S|$, and

$$W = ||f(x_1X_1,\ldots x_{n+2}X_{n+2})||_\infty \geq Ne_2\cdots e_nX_2 \approx N^{n+\delta},$$

assuming the e_i's are of full bitsize for $2 \leq i \leq n$. From the structure of the polynomial f, it is clear that $s_2,\ldots,s_{n+1}$ are equal. Thus we only need to calculate[1]

[1] The detailed calculations for $s, s_1, \ldots, s_{n+2}$ are quite tedious and these are presented a little later, not to hamper the continuity of this proof.

s, s_1, s_2 and s_{n+2}. We get,

$$\begin{aligned}
s &\approx \frac{1}{(n-1)!} \cdot \frac{m^{n+2}}{(n+1)(n+2)} + \frac{t}{(n-1)!} \cdot \frac{m^{n+1}}{n(n+1)}, \\
s_1 &\approx \frac{m^{n+2}}{(n-1)! \cdot (n+2)(n+1)} + \frac{tm^{n+1}}{(n-1)! \cdot n(n+1)}, \\
s_2 = \cdots = s_{n+1} &\approx \frac{m^{n+2}}{n! \cdot (n+2)} + \frac{tm^{n+1}}{(n-2)! \cdot n(n-1)(n+1)}, \\
s_{n+2} &\approx \frac{m^{n+2}}{(n-1)! \cdot 2(n+2)} + t \cdot \frac{m^{n+1}}{(n-1)! \cdot (n+1)} + t^2 \cdot \frac{m^n}{(n-1)! \cdot 2n}.
\end{aligned}$$

Consider $t = \tau m$, where $\tau \geq 0$ is a real number. Putting the values of $X_1, X_2, \ldots, X_{n+2},\ s_1, \ldots, s_{n+2}, s,$ and the lower bound of W in the condition $X_1^{s_1} X_2^{s_2} \ldots X_{n+2}^{s_{n+2}} < W^s$, we get

$$n^2\tau^2 + 4n^2\tau\delta - 2n^2\tau + 3n\tau^2 + 4n^2\delta + 8n\tau\delta - 3n^2 - 4n\tau + 2\tau^2 + 4n\delta + n < 0. \quad (4.2)$$

The optimal value of τ to maximize δ is $\frac{(1-2\delta)n}{1+n}$. One may note that $\tau \leq 0$ when the maximum value of δ is greater than $\frac{1}{2}$. For the cases $n \geq 3$, we get that the upper bound of δ greater than $\frac{1}{2}$ for $\tau = 0$. Thus in these cases, it is enough to consider $\tau = 0$, i.e., $t = 0$. In these cases, putting $\tau = 0$ in (4.2), we get

$$\delta < \frac{3n-1}{4n+4}.$$

For the cases $n \geq 3$, extra shifts over the variable x_{n+2} does not provide any improvement in the theoretical bound. Thus, it is enough to consider $i_{n+2} = 0, \ldots, i_2 + \cdots + i_{n+1}$ instead of $i_{n+2} = 0, \ldots, i_2 + \cdots + i_{n+1} + t$. For the case $n = 2$ though, the extra shifts over x_4 provide theoretical improvements. Putting $\tau = \frac{(1-2\delta)n}{1+n}$ in (4.2), we get $\delta < 0.422$, which provides a better bound compared to $\frac{3\times 2-1}{4\times 2+4} \approx 0.416$.

Using the strategy of Section 2.6, one can construct a lattice L from S, M. The bitsize of the entries of L is poly(log N), and

$$\dim(L) = |M| = \frac{1}{(n-1)!} \cdot \frac{(m+1)^{n+2}}{(n+1)(n+2)} + \frac{t}{(n-1)!} \cdot \frac{(m+1)^{n+1}}{n(n+1)} + o((m+1)^{n+2}).$$

The running time of our algorithm is dominated by the LLL algorithm run on L, which takes time polynomial in the dimension of the lattice and in the bitsize of the entries. Since the lattice dimension in our case is exponential in n, so the total running time for this method is $\mathrm{poly}\{\log \mathrm{N}, \exp(\mathrm{n})\}$. ■

Remark 4.3. If the encryption exponents are not relatively prime, as assumed in Theorem 4.2, then $f(x_1, x_2, \ldots, x_{n+2})$ will not be an irreducible polynomial. However, the GCD of any two encryption exponents will be small in general, and hence $f(x_1, x_2, \ldots, x_{n+2})$ will be of the form $cg(x_1, x_2, \ldots, x_{n+2})$, where c is a small constant. In this case, one needs to find the root of $g(x_1, x_2, \ldots, x_{n+2})$ instead.

	$n = 2$	$n = 3$	$n = 4$	$n = 5$	$n = 6$
Method of [62]	0.357	0.400	0.441	0.468	0.493
Our Method	0.422	0.500	0.550	0.583	0.607

Table 4.1: Comparison of our theoretical bounds with that of [62].

Our bound in Theorem 4.2 is clearly better than that of Howgrave-Graham and Seifert [62]. Our approach works when upper bound of d_i is less than $N^{0.75}$ for $1 \leq i \leq n$ as $n \to \infty$, whereas the bound is $N^{0.5}$ in [62]. In Table 4.1, we present comparative upper bounds of d_i for different values of n.

Theorem 4.2 may be extended when some of the most significant bits (MSBs) of the decryption exponents are same (but unknown). This implicit information increases the bound of decryption exponents even further, as follows.

Corollary 4.4. *Let $e_1, \ldots, e_n$ (where $n \geq 2$) be RSA encryption exponents with common modulus N, and suppose that $d_1, \ldots, d_n$ are the corresponding decryption exponents. Also suppose that $\gcd(e_i, e_j) = 1$ for all $i \neq j$ where $1 \leq i, j \leq n$. Let $d_1, d_2, \ldots, d_n < N^\delta$ and $|d_u - d_v| < N^\beta$ for $u \neq v \in [1, n]$. Then one can factor N in $\mathrm{poly}\{\log \mathrm{N}, \exp(\mathrm{n})\}$ time when $\tau = \max\{0, \frac{-2n\delta+n-2\beta+2\delta}{n+1}\}$ and*

$$\begin{aligned} n^2\tau^2 + 4n^2\tau\delta - 2n^2\tau + 3n\tau^2 + 4n\tau\beta + 4n^2\delta \\ +4n\tau\delta - 3n^2 - 4n\tau + 2\tau^2 + 4n\beta + 8\tau\beta - 8\tau\delta + n < 0. \end{aligned}$$

Proof. First consider the case when n is even. If $E = \prod_{i=1}^n e_i$, we have

$$E \cdot \sum_{i=1}^{n} (-1)^{i+1} d_i = \sum_{j=1}^{n} (-1)^{j+1} \frac{E}{e_j} + (N + r) \cdot \left(\sum_{j=1}^{n} (-1)^{j+1} \frac{E}{e_j} k_j \right).$$

We want to find a solution $(d_1 - d_2 + d_3 - \cdots - d_n, k_1, \ldots, k_n, r)$ of the polynomial

$$f\left(x_1, x_2, \ldots, x_{n+2}\right) = Ex_1 - \sum_{j=1}^{n} (-1)^{j+1} \frac{E}{e_j} - (N + x_{n+2}) \left(\sum_{j=1}^{n} (-1)^{j+1} \frac{E}{e_j} x_{j+1}\right).$$

In this case we have $|d_1 - d_2 + d_3 - \cdots - d_n| < N^{\beta}$, assuming that n is a fixed small integer, negligible compared to N^{β}. Let $X_1 = N^{\beta}, X_2 = \cdots = X_{n+1} = N^{\delta}$ and $X_{n+2} = N^{\frac{1}{2}}$. Then $X_1, X_2, \ldots, X_{n+2}$ are the upper bounds of $d_1 - d_2 + d_3 - \cdots - d_n, k_1, \ldots, k_n, r$ respectively, neglecting constant terms. Now proceeding as in the proof of Theorem (4.2), we get the claimed bound. The situation can be handled in a similar manner for odd n as $|2d_1 - d_2 + d_3 - \cdots - d_{n-1} - d_n| = |(d_1 - d_2 + d_3 - \cdots - d_{n-1}) + (d_1 - d_n)|$ can be bounded above by N^{β}. ■

Detailed Calculations related to Theorem 4.2

Calculation of s

One may note that s is the number of solutions of $0 \leq i_1 + \cdots + i_{n+1} \leq m$, $0 \leq i_{n+2} \leq i_2 + \cdots + i_{n+1} + t$. Using Lemma 4.1 and neglecting lower order terms,

$$\begin{aligned} s &= \sum_{r=0}^{m} (1 + r + t)(m + 1 - r) \binom{r + n - 1}{r} \\ &\approx \sum_{r=0}^{m} (1 + r + t)(m + 1 - r) \frac{r^{n-1}}{(n-1)!} \end{aligned}$$

which gives the following:

$$\begin{aligned} s &\approx \sum_{r=0}^{m} (r + t)(m - r) \frac{r^{n-1}}{(n-1)!} \\ &\approx \frac{1}{(n-1)!} \cdot \frac{m^{n+2}}{(n+1)(n+2)} + \frac{t}{(n-1)!} \cdot \frac{m^{n+1}}{n(n+1)}. \end{aligned}$$

Calculation of s_1

$$\begin{aligned} s_1 = &\sum_{i_1=0}^{m+1}\sum_{r=0}^{m-i_1+1}\binom{r+n-1}{r}(r+t+1)\cdot i_1 \\ &\qquad - \sum_{i_1=0}^{m}\sum_{r=0}^{m-i_1}\binom{r+n-1}{r}(r+t+1)\cdot i_1 \end{aligned}$$

Now, using Lemma 4.1 and neglecting the lower order terms, we obtain

$$\begin{aligned} &\sum_{i_1=0}^{m+1}\sum_{r=0}^{m-i_1+1}\binom{r+n-1}{r}(r+t+1)\cdot i_1 \\ \approx &\sum_{i_1=0}^{m+1}\sum_{r=0}^{m-i_1+1}\frac{r^{n-1}}{(n-1)!}(r+t)\cdot i_1 \\ \approx &\sum_{i_1=0}^{m+1}\frac{(m-i_1+1)^{n+1}}{(n-1)!\cdot(n+1)}\cdot i_1 + \sum_{i_1=0}^{m+1}\frac{(m-i_1+1)^{n}}{(n-1)!\cdot n}\cdot ti_1 \\ \approx &\sum_{T=0}^{m+1}\frac{T^{n+1}(m-T+1)}{(n-1)!\cdot(n+1)} + \sum_{T=0}^{m+1} t\cdot\frac{T^{n}(m-T+1)}{(n-1)!\cdot n}, \quad \text{where } T = m-i_1+1 \\ \approx &\frac{(m+1)^{n+3}}{(n-1)!\cdot(n+3)(n+2)(n+1)} + \frac{t}{(n-1)!}\cdot\frac{(m+1)^{n+2}}{n(n+1)(n+2)}. \end{aligned}$$

Thus, the final expression for s_1 is as follows.

$$\begin{aligned} s_1 &\approx \frac{(m+1)^{n+3}}{(n-1)!\cdot(n+3)(n+2)(n+1)} + \frac{t}{(n-1)!}\cdot\frac{(m+1)^{n+2}}{n(n+1)(n+2)} \\ &\quad - \frac{m^{n+3}}{(n-1)!\cdot(n+3)(n+2)(n+1)} - \frac{t}{(n-1)!}\cdot\frac{m^{n+2}}{n(n+1)(n+2)} \\ &\approx \frac{m^{n+2}}{(n-1)!\cdot(n+2)(n+1)} + \frac{tm^{n+1}}{(n-1)!\cdot n(n+1)} \end{aligned}$$

Calculation of s_2

$$s_2 = \sum_{i_1=0}^{m+1} \sum_{i_2=0}^{m-i_1+1} \sum_{r=0}^{m+1-i_1-i_2} \sum_{i_{n+2}=0}^{r+t+i_2} \binom{r+n-2}{r} \cdot i_2 - \sum_{i_1=0}^{m} \sum_{i_2=0}^{m-i_1} \sum_{r=0}^{m-i_1-i_2} \sum_{i_{n+2}=0}^{r+t+i_2} \binom{r+n-2}{r} \cdot i_2.$$

Now, using Lemma 4.1 and neglecting lower order terms, we obtain

$$\begin{aligned}
&\sum_{i_1=0}^{m+1} \sum_{i_2=0}^{m-i_1+1} \sum_{r=0}^{m+1-i_1-i_2} \sum_{i_{n+2}=0}^{r+t+i_2} \binom{r+n-2}{r} \cdot i_2 \\
=&\sum_{i_1=0}^{m+1} \sum_{i_2=0}^{m-i_1+1} \sum_{r=0}^{m+1-i_1-i_2} \binom{r+n-2}{r} \cdot i_2 \cdot (r+t+i_2+1) \\
\approx&\sum_{i_1=0}^{m+1} \sum_{i_2=0}^{m-i_1+1} \sum_{r=0}^{m+1-i_1-i_2} \left(\frac{r^{n-1}}{(n-2)!} \cdot i_2 + \frac{r^{n-2}}{(n-2)!}(i_2^2 + ti_2) \right) \\
\approx&\frac{1}{(n-2)!} \sum_{i_1=0}^{m+1} \sum_{i_2=0}^{m-i_1+1} \left(i_2 \cdot \frac{(m+1-i_1-i_2)^n}{n} + (i_2^2 + ti_2)\frac{(m+1-i_1-i_2)^{n-1}}{n-1} \right) \\
=&\frac{1}{(n-2)!} \sum_{r_1=0}^{m+1} \sum_{i_2=0}^{r_1} \left(i_2 \cdot \frac{(r_1-i_2)^n}{n} + (i_2^2 + ti_2)\frac{(r_1-i_2)^{n-1}}{n-1} \right) \\
=&\frac{1}{(n-2)!} \sum_{r_1=0}^{m+1} \sum_{T=0}^{r_1} \left((r_1-T)\frac{T^n}{n} + (r_1-T)^2 \frac{T^{n-1}}{n-1} + t \cdot (r_1-T)\frac{T^{n-1}}{n-1} \right) \\
\approx&\frac{1}{n! \cdot (n+2)} \cdot \frac{(m+1)^{n+3}}{n+3} + \frac{t}{(n-2)!} \cdot \frac{(m+1)^{n+2}}{(n-1)n(n+1)(n+2)},
\end{aligned}$$

denoting $T = r_1 - i_2$. Hence, we get the expression for s_2 as follows.

$$s_2 \approx \frac{m^{n+2}}{n! \cdot (n+2)} + \frac{tm^{n+1}}{(n-2)! \cdot n(n-1)(n+1)}$$

Calculation of s_{n+2}

$$s_{n+2} = \sum_{i_1=0}^{m+1} \sum_{r=0}^{m-i_1+1} \sum_{i_{n+2}=0}^{r+t} \binom{r+n-1}{r} \cdot i_{n+2} - \sum_{i_1=0}^{m} \sum_{r=0}^{m-i_1} \sum_{i_{n+2}=0}^{r+t} \binom{r+n-1}{r} \cdot i_{n+2}$$

Then, using Lemma 4.1 and neglecting lower order terms, we obtain

$$\sum_{i_1=0}^{m+1}\sum_{r=0}^{m-i_1+1}\sum_{i_{n+2}=0}^{r+t}\binom{r+n-1}{r}\cdot i_{n+2}$$
$$=\sum_{i_1=0}^{m+1}\sum_{r=0}^{m-i_1+1}\binom{r+n-1}{r}\frac{(r+t)(r+t+1)}{2}$$
$$\approx\frac{1}{2(n-1)!}\sum_{i_1=0}^{m+1}\left(\frac{(m-i_1+1)^{n+2}}{n+2}+\frac{2t\cdot(m-i_1+1)^{n+1}}{n+1}+\frac{t^2\cdot(m-i_1+1)^{n}}{n}\right)$$
$$\approx\frac{(m+1)^{n+3}}{(n-1)!\cdot 2(n+2)(n+3)}+\frac{2t\cdot(m+1)^{n+2}}{(n-1)!\cdot 2(n+2)(n+1)}+\frac{t^2\cdot(m+1)^{n+1}}{(n-1)!\cdot 2(n+1)n}.$$

Hence, the final expression for s_{n+2} is as follows.

$$s_{n+2}\approx\frac{m^{n+2}}{(n-1)!\cdot 2(n+2)}+t\cdot\frac{m^{n+1}}{(n-1)!\cdot(n+1)}+t^2\cdot\frac{m^{n}}{(n-1)!\cdot 2n}$$

4.2 Experimental Results

Let us now present the experimental results. As the lattice used in [62] is of smaller dimension, the time required was of the order of a few seconds. However, using our strategy, one requires a lattice of higher dimension than that of [62], and thus the time required is of the order of a few hours. We get substantially better experimental results than [62] for $m = 3, t = 0$, i.e., when the lattice dimension is 105, in the presence of two decryption exponents ($n = 2$). We list the experimental results to show that they improve the bounds presented in [62].

l_N	l_{d_i} (Theory of [62])	l_{d_i} (Expt. of [62])	l_{d_i} (Thm. (4.2))	l_{d_i} (Our expt.)
500	178	178	208	192
700	250	249	291	267
1000	357	-	416	383

Table 4.2: Comparison of theoretical and experimental results for $n = 2$.

In Theorem 4.2, we have considered Assumption 1 for finding common root of polynomials, as mentioned earlier in Section 2.5.1. Let us now clarify how it actually worked in our experiments. In the proof of Theorem 4.2, we considered that we will be able to get at least three polynomials f_0, f_1, f_2 along with f, that share the integer root. In experiments, we found more than 4 polynomials (other than f) after the LLL algorithm that share the root. Let us call them f_0, f_1, f_2, f_3. We calculate $f_4 = R(f, f_0)$, $f_5 = R(f, f_1)$, $f_6 = R(f, f_2)$, $f_7 = R(f, f_3)$ and then iterate the process to obtain $f_8 = R(f_4, f_5)$, $f_9 = R(f_6, f_7)$. In all the experiments, we observe that $x_3^4 x_4^4 = \gcd(f_8, f_9)$. Thus we calculate

$$f_{10} = R\left(\frac{f_8}{x_3^4 x_4^4}, \frac{f_9}{x_3^4 x_4^4}\right)$$

and we find that f_{10} is a polynomial in only x_4, which corresponds to k_2 in the proof of Theorem 4.2. Since $d_1, d_2 < N^{0.416}$ and $p + q < \left(\sqrt{2} + \frac{1}{\sqrt{2}}\right) N^{0.5}$, we have $p + q < e_2$. Thus, we can find $p + q$ by calculating $\left(N + 1 + k_2^{-1}\right) \bmod e_2$ and this immediately provides the factorization of N.

We have also studied the case $n = 3$. In this case, we take $m = 2$ which makes the lattice dimension 98. We consider 1000-bit N and three decryption exponents d_1, d_2, d_3, each of 420 bits. We could factor N under Assumption 1 (Section 2.5.1) in time 12390 seconds. The previous work of [62] could only achieve the solution with three decryption exponents of 400 bits each. Thus we get an experimental advantage over [62] as well for the case $n = 3$. We could not attempt experiments for $n > 3$ as in those cases the lattice dimension becomes quite high.

4.3 Conclusion

In this chapter we prove that if n decryption exponents are used with the same RSA modulus N, then RSA becomes insecure when $d_i < N^{\frac{3n-1}{4n+4}}$ for $1 \leq i \leq n$ and $n \geq 2$. This improves the current best known bound [62]. We have experimentally demonstrated better results than that of [62] for the cases $n = 2$ and $n = 3$.

In Chapters 3 and 4, we have presented an analysis of the vulnerabilities of RSA due to weak keys. Apart from these, RSA may be vulnerable in case certain information regarding the bits of RSA primes are compromised due to some side-channel leakage. In the next chapter, we study this and present a discussion on RSA prime reconstruction if certain random bits of the primes are known.

Chapter 5

Reconstruction of Primes given few of its Bits

An extensive amount of research has been done in RSA factorization and we refer the reader to the survey papers by Boneh [11] and May [84] for a complete account. One major class of RSA attacks exploit partial knowledge of the RSA secret keys or the primes. Rivest and Shamir [109] pioneered these attacks using Integer Programming and factored RSA modulus given two-third of the LSBs of a factor. Later, a seminal paper [24] by Coppersmith proved that factorization of the RSA modulus can be achieved given half of the MSBs of a factor. His method used LLL [77] lattice reduction technique to solve for small solutions to modular equations. This method triggered a host of research in the field of lattice based factorization, e.g., the works by Howgrave-Graham [59], Jochemsz and May [65].

These results require knowledge of contiguous blocks of bits of the RSA secret keys or the primes. However, in an actual practical scenario of side-channel attacks, it is more likely that an adversary will gain the knowledge of random bits of the RSA parameters instead of contiguous blocks. In fact, the cold-boot attack proposed by Halderman et al [46] in 2009 was mounted to recover random bits of RSA secret parameters exploiting data remanence in the computer memory. Thus the motivation comes from side channel attack on RSA where some bits of p and q are revealed but not the entire key. In this model, the application of the earlier factorization methods prove insufficient, and one requires a way to extract more information out of the random bits obtained via the side channel attacks. In [51], it has been shown how N can be factored with the knowledge of a random subset of the bits (distributed over small contiguous blocks) in one of the primes. Later, a similar result has been studied by Heninger and Shacham [50] to reconstruct the RSA private keys given a certain fraction of the bits, distributed at random. This is the work [50] where the random bits of both the primes are considered unlike the earlier works (e.g., [16, 24, 51]) where knowledge of the bits of a single prime have been exploited.

This chapter studies how the least (respectively most) significant halves of the RSA primes can be completely recovered from some amount of randomly chosen bits from the least (respectively most) significant halves of the same. Thereafter one can exploit the existing lattice based results towards factoring the RSA modulus $N = pq$ when p, q are of the same bitsize. It is possible to factor N in any one of the following cases in $\mathrm{poly}(\log \mathrm{N})$ time: (i) when the most significant half of any one of the primes is known [24, Theorem 4], (ii) when the least significant half of any one of the primes is known [16, Corollary 2.2].

5.1 LSB Case: Combinatorial Analysis of Existing Work

In this section, we analyze the reconstruction algorithm by Heninger and Shacham [50, Section 3] from combinatorial point of view. Though the algorithm extends to all relations among the RSA secret keys, we shall concentrate our attention to the primary relation $N = pq$ for the sake of factorization. The algorithm is a smart brute-force method on the total search space of unknown bits of p and q, which prunes the solutions those are infeasible given the knowledge of N and some random bits of the primes.

5.1.1 The Reconstruction Algorithm

Definition 5.1. Let us define $X[i]$ to be the i-th bit of X with $X[0]$ being the LSB. Also define X_i to be the partial approximation of X through the bits 0 to i.

Then Algorithm 7 creates all possible pairs (p_i, q_i) by appending $(p[i], q[i])$ to the partial solutions (p_{i-1}, q_{i-1}) and prunes the incorrect ones by checking the validity of the available relation. A formal outline of Algorithm 7, which retrieves the least significant t many bits of both p, q, is as follows. It is easy to see that the correct partial solution till the t many LSBs will exist in the set of all pairs (p_{t-1}, q_{t-1}) found from Algorithm 7.

```
   **Input**: $N, t$ and $p[i], q[j]$, for some random values of $i, j$
   **Output**: Contiguous $t$ many LSBs of $p, q$
1  Initialize: $i = 1$ and $p_0 = p[0] = 1$, $q_0 = q[0] = 1$ (as both are odd);
2  **for** *all* $(p_{i-1}, q_{i-1})$ **do**
3      **for** *all possible* $(p[i], q[i])$ **do**
4          $p_i :=$ APPEND$(p[i], p_{i-1})$;
5          $q_i :=$ APPEND$(q[i], q_{i-1})$;
6          **if** $N \equiv p_i q_i \pmod{2^{i+1}}$ **then**
7              ADD the pair $(p_i, q_i)$ at level $i$;
           **end**
       **end**
   **end**
8  **if** $i < t - 1$ **then**
9      $i := i + 1$;
10     GOTO Step 2;
   **end**
11 REPORT all $(p_{t-1}, q_{t-1})$ pairs;
```
Algorithm 7: The search algorithm.

As one may notice, there are at most 4 possible choices for $(p[i], q[i])$ branches at any level i. Algorithm 7 works with all possible combinations of the bits $p[i], q[i]$ at level i and hence one may want to obtain a relation between $p[i]$ and $q[i]$ in terms of the known values of N, p_{i-1}, q_{i-1} so that it poses a constraint on the possibilities. Heninger and Shacham [50, Section 4] uses Multivariate Hensel's Lemma to obtain such a relation

$$p[i] + q[i] \equiv (N - p_{i-1}q_{i-1})\,[i] \pmod{2}. \tag{5.1}$$

Now, this linear relation between $p[i], q[i]$ restricts the possible choices for the bits. Thus, at any level i, instead of 4 possibilities, the number cuts down to 2.

If we construct the search tree, then these possibilities for the bits at any level give rise to new branches in the tree. The tree at any level i contains all the partial solutions p_i, q_i up to the i-th LSB (the correct partial solution is one among them). It is quite natural to restrict the number of potential candidates (i.e., the partial solutions) at any level so that the correct one can be found easily by exhaustive search among all the solutions and the space to store all these solutions is within certain feasible limit. This calls for restricting the width of the search tree at each level. Let us denote the width of the tree at level i by W_i. Now we take a look at the situations (depending on the knowledge of the random bits of the primes) that control the branching behavior of the tree.

5.1.2 Growth of the Search Tree

Consider the situation where we have a pair of partials (p_{i-1}, q_{i-1}) and do not have any information of $(p[i], q[i])$ in Step 3 of Algorithm 7. Naturally there are 4 options, $(0,0), (0,1), (1,0)$ and $(1,1)$ for getting (p_i, q_i). However, Equation (5.1) and the knowledge of N, p_{i-1}, q_{i-1} impose a linear dependence between $p[i], q[i]$ and hence restrict the number of choices to exactly 2. If $(N - p_{i-1}q_{i-1})\,[i] = 0$ then we have $p[i] + q[i] \equiv 0 \pmod 2$ and $(N - p_{i-1}q_{i-1})\,[i] = 1$ implies $p[i] + q[i] \equiv 1 \pmod 2$. Hence the width of the tree at this level will be twice the width of the tree at the previous one, as shown in Figure 5.1.

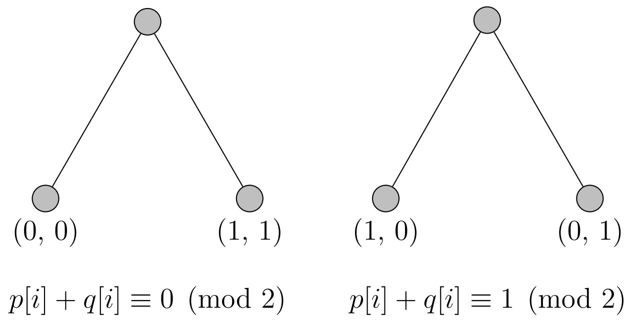
(0, 0)
(1, 1)
(1, 0)
(0, 1)
$p[i] + q[i] \equiv 0 \pmod 2$
$p[i] + q[i] \equiv 1 \pmod 2$

Figure 5.1: Branching when both the bits $p[i], q[i]$ are unknown.

Next, let us have a look at the situation when exactly one of $p[i], q[i]$ is known. First, the number of branches restricts to 2 by Equation (5.1), as discussed before. Moreover, the knowledge of one bit fixes the other in this relation. For example, if one knows the value of $p[i]$ along with N, p_{i-1}, q_{i-1} in Equation (5.1), then $q[i]$ gets determined. Thus the number of choices for $p[i], q[i]$ and hence the number of p_i, q_i branches reduces to a single one in this case. This branching, which keeps the tree-width fixed, may be illustrated as in Figure 5.2 ($p[i] = 0$ is known, say).

Though the earlier two cases are easy to understand, the situation is not so simple when both $p[i], q[i]$ are known. In this case, the validity of Equation (5.1) comes under scrutiny. If we fit in all the values $p[i], q[i], N, p_{i-1}, q_{i-1}$ in Equation (5.1) and it is satisfied, then we accept the new partial solution p_i, q_i at level i

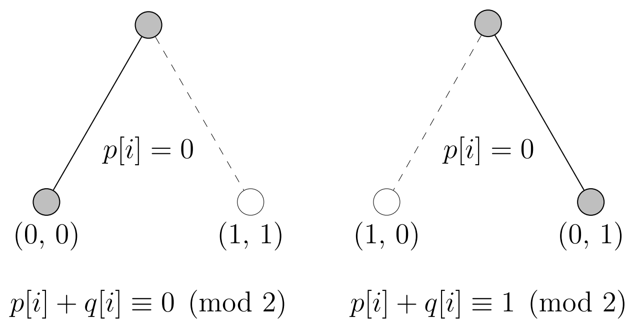
$p[i] = 0$
$p[i] = 0$
(0, 0)
(1, 1)
(1, 0)
(0, 1)
$p[i] + q[i] \equiv 0 \pmod 2$
$p[i] + q[i] \equiv 1 \pmod 2$

Figure 5.2: Branching when exactly one bit of $p[i], q[i]$ is known.

and otherwise we do not. In the case where neither of the possibilities for p_i, q_i generated from p_{i-1}, q_{i-1} satisfy the relation, we discard the whole subtree rooted at p_{i-1}, q_{i-1}. Thus, the pruning procedure not only discards the wrong ones at level i, but also discards subtrees from level $i-1$, thereby narrowing down the search tree. An example case ($p[i] = 0$ and $q[i] = 1$ are known, say) may be presented as in Figure 5.3.

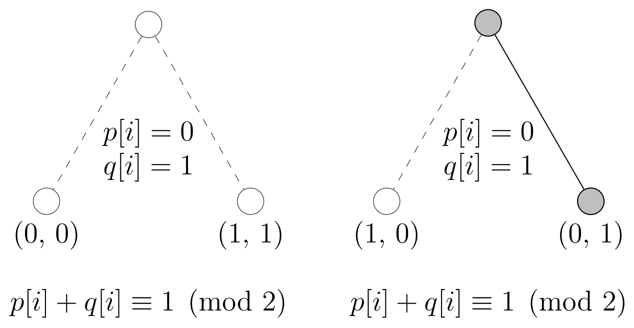

Figure 5.3: Branching when both the bits $p[i], q[i]$ are known.

Based on our discussion so far, let us try to model the growth of the search tree following Algorithm 7. As both p, q are odd, we have $p[0] = 1$ and $q[0] = 1$. Thus the tree starts from $W_0 = 1$ and the expansion or contraction of the tree at each level can be modeled as follows.

- $p[i]$ = UNKNOWN, $q[i]$ = UNKNOWN: $W_i = 2W_{i-1}$.
- $p[i]$ = KNOWN, $q[i]$ = UNKNOWN: $W_i = W_{i-1}$.
- $p[i]$ = UNKNOWN, $q[i]$ = KNOWN: $W_i = W_{i-1}$.
- $p[i]$ = KNOWN, $q[i]$ = KNOWN: $W_i = \gamma_i W_{i-1}$.

Here, we assume that the tree narrows down to a γ_i fraction ($0 < \gamma_i \leq 1$) from the earlier level if both the bits of the primes are known. One may note that Heninger and Shacham [50, Conjecture 4.3] conjectures the average value of γ_i (call it γ) to be $\frac{1}{2}$. We shall discuss this in more details later.

Suppose that randomly chosen α fraction of bits of p and β fraction of bits of q are known (by some side channel attack, e.g., cold boot). Then the joint probability distribution table for the bits of the primes will be as follows.

$\downarrow q[i]$, $p[i] \rightarrow$	UNKNOWN	KNOWN
UNKNOWN	$(1-\alpha)(1-\beta)$	$\alpha(1-\beta)$
KNOWN	$(1-\alpha)\beta$	$\alpha\beta$

As shown before, the growth of the search tree depends upon the knowledge of the bits in the primes. Hence, we can model the growth of the tree as a recursion on the level index i:

$$\begin{aligned} W_i &= (1-\alpha)(1-\beta)2W_{i-1} + \alpha(1-\beta)W_{i-1} + (1-\alpha)\beta W_{i-1} + \alpha\beta\gamma_i W_{i-1} \\ &= (2 - \alpha - \beta + \alpha\beta\gamma_i)\, W_{i-1}. \end{aligned}$$

If we want to restrict W_i (that is the growth of the tree) as a polynomial of i (that is the number of level), we would like (roughly speaking) the value of $(2-\alpha-\beta+\alpha\beta\gamma_i)$ close to 1 on an average. Considering the average value γ (instead of γ_i at each level), we get, $2 - \alpha - \beta + \alpha\beta\gamma \approx 1$ which implies $1 - \alpha - \beta + \alpha\beta\gamma \approx 0$. If we assume that the same fraction of bits are known for p and q, then $\alpha = \beta$ and we get $1 - 2\alpha + \alpha^2\gamma \approx 0 \quad \Rightarrow \quad \alpha \approx \frac{1-\sqrt{1-\gamma}}{\gamma}$. If we assume [50, Conjecture 4.3], then $\gamma \approx 0.5$ and hence $\alpha \approx 2 - \sqrt{2} \approx 0.5858$, as obtained in [50, Section 4.4]. One may note that our idea is simpler compared to the explanation in [50]. This simplification is achieved here by using average value for γ_i in the recurrence relation of W_i.

The most natural strategy is to first apply Algorithm 7 to retrieve the least significant half of any one of the primes and then apply the result of Boneh et al [16, Corollary 2.2] to factorize N. One may note that [50] utilizes their prime reconstruction algorithm to reconstruct the whole primes p, q whereas our idea is to use lattice based results after reconstructing just one half of any prime. This is more practical as it requires the knowledge of lesser number of random bits of the primes, namely, just about $0.5858 \times 0.5 \approx 0.3$ fraction of bits (from the LSB half) instead of 0.5858 fraction of the primes as explained in [50]. Moreover, factorization being the main objective, one need not reconstruct the primes completely, but just requires to obtain enough information that suffices for factoring the product N based on the existing efficient techniques. In this direction, let us first present the following result.

5.1.3 Known Prime Bits: Complementary Sets for p, q

Theorem 5.2. *Let $N = pq$, when p, q are primes of same bitsize. Let $S = \{0, \ldots, \lceil l_N/4 \rceil\}$. Consider $U \subseteq S$ and $V = S \setminus U$. Assume that $p[i]$'s for $i \in U$ and $q[j]$'s for $j \in V$ are known. Then one can factor N in* $\mathrm{poly}(\log \mathrm{N})$ *time.*

Proof. Let us apply Algorithm 7 in this case to retrieve the bits of the primes at each level. We shall use induction on the index of levels in this case.

For level 0, we know that $p[0] = 1$ and $q[0] = 1$. Hence, the width of the search tree is $W_0 = 1$ and we have a single correct partial (p_0, q_0). Let us suppose that we have possible pairs of partials (p_{i-1}, q_{i-1}) at level $i-1$, generated by Algorithm 7. At level i, two cases may arise. If $i \in U$ then we know $p[i], N, p_{i-1}, q_{i-1}$ which restricts the branching to a single branch and keeps the width of the tree fixed $(W_i = W_{i-1})$. Else one must have $i \in V$ $(V = S \setminus U)$ and we know $q[i], N, p_{i-1}, q_{i-1}$. This restricts the branching to a single branch as well and keeps the width fixed $(W_i = W_{i-1})$. Hence, by induction on i, $W_i = W_{i-1}$ for $i = 0, \ldots, \lceil l_N/4 \rceil$. As $W_0 = 1$, this boils down to $W_i = 1$ for $i \leq \lceil l_N/4 \rceil$.

Thus we obtain a single correct partial pair p_i, q_i at level $i = \lceil l_N/4 \rceil$ using Algorithm 7 in $O(\log^3 N)$ time ($\lceil l_N/4 \rceil$ iterations and $O(\log^2 N)$ computing time for each iteration) and $O(l_N/2)$ space (actually we need space to store a single partial pair at the current level). This provides us with the least significant half of both the primes and using any one of those two, the lattice based method of [16, Corollary 2.2] completes the factorization of $N = pq$ in $\mathrm{poly}(\log \mathrm{N})$ time. ∎

It is interesting to analyze the implications of this result in a few specific cases. An extreme case may be $U = S$, that is we know all the bits in the least significant half of a single prime p and do not know any such bits for q. In this scenario, one need not apply Algorithm 7 at all and the lattice based method in [16, Corollary 2.2] suffices for factorization. Second case is when $|U| = |S| - x$, i.e., missing x bits of p at random positions. In such a case, one can use a brute force search for these missing bits and apply lattice based factoring method [16, Corollary 2.2] for all of the 2^x possibilities, if x is small. However, for large x, e.g., $x \approx |U|$, i.e., around half of the random bits from the least significant halves of p as well as q are known, then the brute force strategy fails, and one must use Algorithm 7 before applying the lattice based method in [16, Corollary 2.2].

5.1.4 Known Prime Bits: Distributed at Random

Here we consider the case when random bits of p, q are available, lifting the constraint $V = S \setminus U$. That is, here we only consider U, V to be random subsets of S. For 512-bit primes, we observed that knowledge of randomly chosen half of the bits from least significant halves of p, q is sufficient to recover the complete least significant halves of p as well as q using Algorithm 7.

Now let us present a select few of our experimental results in Table 5.1. The first column represents the size of the RSA primes and the second column gives the fraction of bits known randomly from the least significant halves of the primes (call these α_p, β_q respectively). The value of t in the third column is the target level we need to run Algorithm 7 for, and is half the size of the primes. W_t is the final width of the search tree at the target level t. This denotes the number of possible partial solutions for p, q at the target bit level t, whereas the next column gives us the maximum width of the tree observed during the run of Algorithm 7. The last column depicts the average value of the shrink ratio γ, as we have defined earlier.

A few crucial observations can be made from the data presented in Table 5.1. We have run the experiments for different sizes of RSA keys, and though the theoretical requirement for the fraction of known bits (α, β) is 0.5858, we have obtained better results when $l_N \leq 2048$. For 512-bit N, the knowledge of just 0.45 fraction of random bits from the least significant halves of the primes proves to be sufficient for Algorithm 7 to retrieve the halves, whereas for 1024 and 2048 bit N, we require about 0.5 fraction of such bits. The main reason is that the growth of the search tree increases with increasing size of the target level t. As we have discussed before, the growth will be independent of the target if we know 0.5858 fraction of bits instead. One may also note that for 1024-bit N, we have obtained successful results when the fraction of bits known is not the same for the two primes. For such skewed cases, the average requirement of known bits stay the same, i.e, 0.5 fraction of the least significant halves. The examples for (0.7,

Size $\lvert p\rvert, \lvert q\rvert$	Known α_p, β_q	Target t	Final W_t	$\max_{i=1}^{t} W_i$	Average γ
256, 256	0.5, 0.5	128	30	60	0.56
256, 256	0.5, 0.5	128	2816	5632	0.52
256, 256	0.47, 0.47	128	106	1508	0.54
256, 256	0.45, 0.45	128	6144	6144	0.49
512, 512	0.5, 0.5	256	352	928	0.53
512, 512	0.5, 0.5	256	8	256	0.55
512, 512	0.5, 0.5	256	716	3776	0.53
512, 512	0.5, 0.5	256	152	2240	0.59
512, 512	0.55, 0.45	256	37	268	0.51
512, 512	0.55, 0.45	256	64	334	0.51
512, 512	0.6, 0.4	256	1648	13528	0.55
512, 512	0.6, 0.4	256	704	5632	0.56
512, 512	0.7, 0.3	256	158	1344	0.53
512, 512	0.7, 0.3	256	47	4848	0.52
1024,1024	0.55, 0.55	512	1	352	0.53
1024,1024	0.53, 0.53	512	16	764	0.53
1024,1024	0.51, 0.51	512	138	15551	0.54
1024,1024	0.51, 0.5	512	17	4088	0.52

Table 5.1: Experimental results corresponding to Algorithm 7.

0.3) in such skewed cases provides interesting results compared to the result by Herrmann and May [51]. Knowing about 70% of the bits of one prime is sufficient for their method to factorize N, but the runtime is exponential in the number of blocks over which the bits are distributed. By knowing 35% of one prime (70% from the least significant half) and 15% of the other (30% of the least significant half), Algorithm 7 can produce significantly better results in the same direction.

Another important point to note from the results is the average value of the shrink ratio γ. It is conjectured in [50] that $\gamma = 0.5$. However, our experiments clearly show that the value of γ is more than 0.5 in *most* (17 out of 18) of the cases. A theoretical explanation of this anomaly may be of interest.

5.1.5 Known Prime Bits: Distributed in a Pattern

In addition to these results, some interesting cases appear when we consider the knowledge of the bits to be periodic in a systematic pattern, instead of being totally random. Suppose that the bits of the primes p, q are available in the following pattern: none of the bits is known over a stretch of U bits, only $q[i]$ is known for Q bits, only $p[i]$ is known for P bits and both $p[i], q[i]$ are known for K bits. This pattern of length $U + P + Q + K$ repeats over the total number of bits. In such a case, one may expect the growth of the tree to obey the following heuristic model – grows in doubles for U bits, stays the same for $Q + P$ length and shrinks thereafter (approximately by halves, considering $\gamma = 0.5$) for a stretch of K bits. If this model is followed strictly, one expects the growth of the tree by a factor of $2^U 2^{-K} = 2^{U-K}$ over each period of the pattern. The total number of occurrences of this pattern over the stretch of T bits is roughly $\frac{T}{U+Q+P+K}$. Hence the width of the tree at level T may be roughly estimated by

$W_T \approx \left[2^{U-K}\right]^{\frac{T}{U+Q+P+K}} = 2^{\frac{T(U-K)}{U+Q+P+K}}$. A closer look reveals a slightly different observation. We have expected that the tree shrinks in half if both bits are known, which is based on the conjecture that $\gamma \approx 1/2$ on an average. But in practical scenario, this is not the case. So, the width W_T at level T, as estimated above, comes as an underestimate in most of the cases.

Let us consider a specific example for such a band-LSB case. The pattern followed is $[U = 5,\ Q = 3,\ P = 3,\ K = 5]$. Using the estimation formula above, one expects the final width of the tree at level 256 to be 1, as $U = K$. But in this case, the final width turns out to be 8 instead. The reason behind this is that the average value of γ in this experiment is 0.55 instead of 0.5.

It is natural for one to notice that the fraction of bits to be known in this band-LSB case is $(P+K)/(U+Q+P+K)$ for the prime p and $(Q+K)/(U+Q+P+K)$ for the prime q. If we choose $Q = P$ and $U = K$, then this fraction is 0.5. Thus, by knowing 50% of the bits from the least significant halves of the primes, that is, knowing just 0.25 fraction of bits in total, Algorithm 7 can factorize $N = pq$ in this case. One may note that the result by Herrmann and May [51] requires the knowledge of about 70% of the bits distributed over arbitrary number of small blocks of a single prime. Thus, in terms of total number of bits to be known (considering both the primes), our result is clearly better.

An extension of this idea may be applied in case of MSBs. Though we can retrieve information about the primes from random bits at the least significant side, we could not exploit similar information from the most significant part. But we could do better if bands of bits are known instead of isolated random bits. A novel idea for reconstructing primes based on such knowledge is presented in Section 5.3.

5.2 LSB Case: Lattice Based Technique

Consider the scenario when a long run (length u) of $p[i], q[i]$ is not known, for $k < i \leq k + u$ say, starting at the $(k + 1)$-th bit level. In such a case, Algorithm 7 will require large memory as the width of the tree will be at least 2^u at the u-th level. If u is large, say $u \geq 50$, then it will be hard to accommodate the number of options, which is greater than 2^{50}. We describe a lattice based method below to handle such situation.

Now we will state and prove the main result of this section.

Theorem 5.3. *Let $N = pq$ where p, q are of same bitsize. Suppose τl_N many least significant bits (LSBs) of p, q are unknown but the subsequent ηl_N many LSBs of both p, q are known. Then, under Assumption 1, one can recover the τl_N many unknown LSBs of p, q in* $\mathrm{poly}(\log \mathrm{N})$ *time, if $\tau < \frac{\eta}{2}$.*

Proof. Let p_0 correspond to the known ηl_N many bits of p and q_0 correspond to the known ηl_N bits of q. Let p_1 correspond to the unknown τl_N many bits of p and q_1 correspond to the unknown τl_N bits of q. Then we have $(2^{\tau l_N} p_0 + p_1)(2^{\tau l_N} q_0 + q_1) \equiv N \pmod{2^{(\tau+\eta) l_N}}$. Let $T = 2^{(\tau+\eta) l_N}$. Hence we are interested to find the root (p_1, q_1) of $f(x, y) = \left(2^{\tau l_N} p_0 + x\right)\left(2^{\tau l_N} q_0 + y\right) - N$ over $\mathbb{Z}_T$.

Let us take $X = 2^{\tau l_N}$ and $Y = 2^{\tau l_N}$. One may note that X, Y are the upper bounds of p_1 and q_1 respectively, neglecting small constants. For a non negative integer m, we define two sets of polynomials

$$\begin{aligned} g_{i,j}(x, y) &= x^i f^j(x, y) T^{m-j}, \text{ where } j = 0, \ldots, m,\ i = 0, \ldots, m - j, \text{ and} \\ h_{i,j}(x, y) &= y^i f^j(x, y) T^{m-j}, \text{ where } j = 0, \ldots, m,\ i = 1, \ldots, m - j. \end{aligned}$$

Note that $g_{i,j}(p_1, q_1) \equiv 0 \pmod{T^m}$ and $h_{i,j}(p_1, q_1) \equiv 0 \pmod{T^m}$. We call $g_{i,j}$ the x-shift and $h_{i,j}$ the y-shift polynomials, as per their respective constructions following the idea described in Section 2.5.

Next, we form a lattice L by taking the coefficient vectors of the shift polynomials $g_{i,j}(xX, yY)$ and $h_{i,j}(xX, yY)$ as basis. One can verify that the dimension of the lattice L is $\omega = (m+1)^2$. The matrix containing the basis vectors of L is lower triangular and has diagonal entries of the form $X^{i+j} Y^j T^{m-j}$, for $j = 0, \ldots, m$ and $i = 0, \ldots, m - j$, and $X^j Y^{i+j} T^{m-j}$ for $j = 0, \ldots, m$ and $i = 1, \ldots, m - j$, coming from $g_{i,j}$ and $h_{i,j}$ respectively. Thus, one can calculate

$$\det(L) = \left[\prod_{j=0}^{m}\prod_{i=0}^{m-j} X^{i+j}Y^{j}T^{m-j}\right]\left[\prod_{j=0}^{m}\prod_{i=1}^{m-j} X^{j}Y^{i+j}T^{m-j}\right] = X^{s_1}Y^{s_2}T^{s_3}$$

where $s_1 = \frac{1}{2}m^3 + m^2 + \frac{1}{2}m$, $s_2 = \frac{1}{2}m^3 + m^2 + \frac{1}{2}m$, and $s_3 = \frac{2}{3}m^3 + \frac{3}{2}m^2 + \frac{5}{6}m$.

To utilize resultant techniques and Assumption 1, we need two polynomials $f_1(x, y)$, $f_2(x, y)$ which share the root (p_1, q_1) over integers. From Theorem 2.23, we know that one can find such $f_1(x, y), f_2(x, y)$ using LLL lattice reduction algorithm [77] over L when $\det(L) < T^{m\omega}$, neglecting the small constants. Given $\det(L)$ and ω as above, the condition becomes $X^{s_1}Y^{s_2}T^{s_3} < T^{m((m+1)^2)}$, i.e., $X^{s_1}Y^{s_2} < T^{s_0}$, where $s_0 = m\left((m+1)^2\right) - s_3 = \frac{1}{3}m^3 + \frac{1}{2}m^2 + \frac{1}{6}m$. Putting the values of the bounds $X = Y = 2^{\tau l_N}$, and neglecting $o(m^3)$ terms, we get $\frac{\tau}{2} + \frac{\tau}{2} < \frac{\tau+\eta}{3}$ and thus get the required bound for τ. Now, one can find the root (p_1, q_1) from f_1, f_2 under Assumption 1. ■

This lattice based technique complements Algorithm 7 by overcoming one of its limitations. As we discussed before, the search tree grows two-folds each time we do not have any information about the bits of the primes. Hence in a case where an initial chunk of LSBs is unknown for both the primes, one can not use Algorithm 7 for reconstruction as it would require huge storage space for the search tree. This lattice based technique provides a feasible solution in such a case. We present a few experimental results in Table 5.2 to illustrate the operation of this technique.

# of Unknown bits (τl_N)	# of Known bits (ηl_N)	Time in Seconds		
		LLL Algorithm	Resultant	Root Extraction
40	90	36.66	25.67	< 1
50	110	47.31	35.20	< 1
60	135	69.23	47.14	< 1
70	155	73.15	58.04	< 1

Table 5.2: Experimental runs of the Lattice Based Technique with dimension 64.

The limitation of this technique is that it asks for double or more the number of missing bits for both the primes. If one misses 60 LSBs for the primes say, this method requires the next 120 or more bits of both the primes to be known to reconstruct all 60 + 120 = 180 LSBs. In the practical scenario, the requirement of bits to be known is 135 (shown in Table 5.2), instead of 120, as we use limited lattice dimensions in the experiments. In all the cases mentioned above, we miss the first τl_N LSBs of the primes. If we miss the information of the bits of the prime in a contiguous block of size τl_N somewhere in the middle, after the i-th level say, then this method offers similar solution if we have $\eta > 2\tau + 2i/l_N$.

5.3 MSB Case: Our Method and its Analysis

In this section, we put forward a novel idea of reconstructing the most significant half of the primes p, q given the knowledge of some blocks of bits. To the best of our knowledge, this has not been studied in a disciplined manner in the existing literature. As before, $N = pq$, and the primes p, q are of the same bitsize. For this section of MSB reconstruction, let us propose the following definition to make notations simpler.

Definition 5.4. Let us define $X[i]$ to be the i-th most significant bit of X with $X[0]$ being the MSB. Also define X_i to be the partial approximation of X where X_i shares the most significant bits 0 through i with X.

5.3.1 The Reconstruction Idea

The idea for reconstructing the most significant halves of the primes is quite simple. We shall use the basic relation $N = pq$. If one of the primes, p say, is known, the other one is easy to find by $q = N/p$. Now, if a few MSBs of one of the primes, p say, is known, then we may obtain an approximation p' of p. This allows us to construct an approximation $q' = \lceil N/p' \rceil$ of the other prime q as well. Our idea is to use the known blocks of bits of the primes in a systematic order to repeat this approximation process until we recover half of one of the primes. A few obvious questions may be as follows.

- How accurate are the approximations?
- How probable is the success of the reconstruction process?
- How many bits of the primes do we need to know?

We answer these questions by describing the reconstruction algorithm in Section 5.3.2 and analyzing the same in Section 5.3.3. But first, let us present an outline of our idea.

Suppose that we have the knowledge of MSBs $\{0,\ldots,a\}$ of prime p. This allows us to construct an approximation p_a of p, and hence an approximation $q' = \lceil N/p_a \rceil$ of q. Lemma 5.5, discussed later in Section 5.3.3, tells us that q' matches q through MSBs $\{0,\ldots,a-t-1\}$, i.e, $q' = q_{a-t-1}$, with some probability depending on t. Now, if one knows the MSBs $\{a-t,\ldots,2a\}$ of q, then a better approximation q_{2a} may be constructed using q_{a-t-1} and these known bits. Again, q_{2a} facilitates the construction of a new approximation $p' = \lceil N/q_{2a} \rceil$, which by Lemma 5.5, satisfies $p' = p_{2a-t-1}$ with some probability depending on t. With the knowledge of MSBs $\{2a-t,\ldots,3a\}$ of p, it is once again possible to construct a better approximation p_{3a} of p. This process of constructing approximations is recursed until one reconstructs the most significant half of one of the primes. A graphical representation of the reconstruction process is illustrated in Figure 5.4.

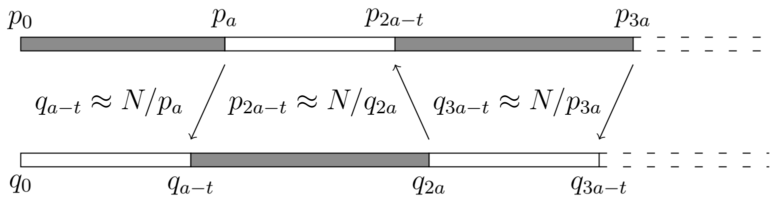

Figure 5.4: The feedback mechanism in MSB reconstruction.

5.3.2 The Reconstruction Algorithm

Let $S = \{0,\ldots,T\}$ denote the set of bit indices from the most significant halves of the primes. Let us assume that $k = \lfloor T/a \rfloor$ is odd in this case. Consider $U, V \subseteq S$ such that $U = \{0,\ldots,a\} \cup \{2a-t,\ldots,3a\} \cup \cdots \cup \{(k-1)a-t,\ldots,ka\}$,
$V = \{a-t,\ldots,2a\} \cup \{3a-t,\ldots,4a\} \cup \cdots \cup \{ka-t,\ldots,T\}$. Also consider that $p[i]$'s are known for $i \in U$ and $q[j]$'s are known for $j \in V$. Then, Algorithm 8 reconstructs T many contiguous most significant bits of the prime q.

The subroutine CORRECT used in Algorithm 8 (and Algorithm 10 later) takes as input a partial approximation Y of X and a set of contiguous known bits, $X[i]$ for $i \in \Sigma$, say. It outputs a better approximation Z of X by correcting the bits of the partial approximation Y using the knowledge of the known bits. Formally,

```
Input: $N, T$ and $p[i], q[j]$ for all $i \in U$ and $j \in V$
Output: Contiguous $T$ many MSBs of $q$
1  Initialize: $p_0 := 2^{l_p-1}$, $q_0 := 2^{l_q-1}$;
2  $p_a :=$ CORRECT($p_0$, $p[j]$ for $j \in \{1,\ldots,a\} \subset U$);
3  $q_{a-t} := \lceil \frac{N}{p_a} \rceil$;
4  for $i$ from 2 to $k-1$ in steps of 2 do
5      $q_{ia} :=$ CORRECT($q_{(i-1)a-t}$, $q[j]$ for $j \in \{(i-1)a-t,\ldots,ia\} \subset V$);
6      $p_{ia-t-1} := \lceil \frac{N}{q_{ia}} \rceil$;
7      $p_{(i+1)a} :=$ CORRECT($p_{ia-t-1}$, $p[j]$ for $j \in \{ia-t,\ldots,(i+1)a\} \subset U$);
8      $q_{(i+1)a-t-1} := \lceil \frac{N}{p_{(i+1)a}} \rceil$;
   end
9  $q_T :=$ CORRECT($q_{ka-t-1}$, $q[j]$ for $j \in \{ka-t,\ldots,T\} \subset V$);
10 REPORT $q_T$;
```

Algorithm 8: The MSB reconstruction algorithm [k odd].

the subroutine works as described is Algorithm 9.

```
Input: $Y$ and $X[i]$ for $i \in \Sigma$
Output: $Z$, the correction of $Y$
1  for $j$ from 0 to $l_X$ do
2      if $j \in \Sigma$ then $Z[j] = X[j]$;   // Correct $j$-th MSB if $X[j]$ is known
       else $Z[j] = Y[j]$;                  // Keep $j$-th MSB as $X[j]$ is not known
   end
3  REPORT $Z$;
```

Algorithm 9: Subroutine CORRECT.

In the case where $k = \lfloor T/a \rfloor$ is even, Algorithm 8 needs to be tweaked a little to work as expected. One may consider a slightly changed version of $U, V \subseteq S$ such that $U = \{0,\ldots,a\} \cup \{2a-t,\ldots,3a\} \cup \cdots \cup \{ka-t,\ldots,T\}$ and $V = \{a-t,\ldots,2a\} \cup \{3a-t,\ldots,4a\} \cup \cdots \cup \{(k-1)a-t,\ldots,ka\}$. As before, $p[i]$'s are known for $i \in U$ and $q[j]$'s are known for $j \in V$. Then, Algorithm 10 reconstructs T many contiguous most significant bits of the prime p.

5.3.3 Analysis of the Reconstruction Algorithm

Algorithm 8 and Algorithm 10 follow the same basic idea of reconstruction as discussed in Section 5.3.1, and differs only in a minor issue regarding the practical

```
Input: $N, T$ and $p[i], q[j]$ for all $i \in U$ and $j \in V$
Output: Contiguous $T$ many MSBs of $p$
1  Initialize: $p_0 := 2^{l_p - 1}$, $q_0 := 2^{l_q - 1}$;
2  $p_a :=$ CORRECT($p_0$, $p[j]$ for $j \in \{1, \ldots, a\} \subset U$);
3  for $i$ from 1 to $k-3$ in steps of 2 do
4      $q_{ia-t-1} := \lceil \frac{N}{p_{ia}} \rceil$;
5      $q_{(i+1)a} :=$ CORRECT($q_{ia-t-1}$, $q[j]$ for $j \in \{ia-t, \ldots, (i+1)a\} \subset V$);
6      $p_{(i+1)a-t-1} := \lceil \frac{N}{q_{(i+1)a}} \rceil$;
7      $p_{(i+2)a} :=$ CORRECT($p_{(i+1)a-t-1}$, $p[j]$ for
       $j \in \{(i+1)a-t, \ldots, (i+2)a\} \subset U$);
   end
8  $q_{(k-1)a-t-1} := \lceil \frac{N}{p_{(k-1)a}} \rceil$;
9  $q_{ka} :=$ CORRECT($q_{(k-1)a-t-1}$, $q[j]$ for $j \in \{(k-1)a-t, \ldots, ka\} \subset V$);
10 $p_{ka-t-1} := \lceil \frac{N}{q_{ka}} \rceil$;
11 $p_T :=$ CORRECT($p_{ka-t-1}$, $p[j]$ for $j \in \{ka-t, \ldots, T\} \subset U$);
12 REPORT $p_T$;
```

Algorithm 10: The MSB reconstruction algorithm [k even].

implementation. We have stated both the algorithms in Section 5.3.2 for the sake of completeness. But in case of the analysis and the experimental results, we shall consider only one of them, Algorithm 8 say, without loss of generality.

Algorithm 8 requires the knowledge of at most $(T - ka + 1) + k(a + t + 1) \leq k(a + t) + (k + a) \leq T(1 + \frac{t}{a}) + (k + a)$ many bits of p and q to (probabilistically) reconstruct T contiguous MSBs of one prime. The runtime of Algorithm 8 is linear in terms of the number of known blocks, i.e, linear in terms of $k = \lfloor T/a \rfloor$. If we set the target $T = l_N/4$, then Algorithm 8 outputs the most significant half of one of the primes in $O(k)$ steps with some probability of success depending on a and t. In this context, we propose Theorem 5.6 to estimate the probability of success of Algorithm 8. Before that, let us introduce the following technical result (Lemma 5.5) which is necessary to prove Theorem 5.6.

Lemma 5.5. *If X and X' are two integers with same bitsize and $|X - X'| < 2^H$, then the probability that X and X' share $l_X - H - t$ many MSBs for some $0 \leq t \leq H$ is at least $P_t = 1 - \frac{1}{2^t}$.*

Proof. We know that $|X - X'| < 2^H$, i.e, $X = X' + Y$ or $X' = X + Z$ where $0 \leq Y, Z < 2^H$, say. Let us consider the case $X = X' + Y$ first, and the other case will follow by symmetry between X and X'.

Let us split $X' = 2^H X_0 + X_1$. Then, clearly $X = 2^H X_0 + (X_1 + Y)$. The addition of $Y < 2^H$ affects the lower part X_1 directly and the carry from the sum $(X_1 + Y)$ affects the first half X_0 up to a certain level. Our goal is to find out the probability that the carry affects less than or equal to t bits of X_0 from the lower side. Let us assume that the probability of $(X_1 + Y)$ generating a carry bit is p_c. We also know that this carry bit will propagate through the lower bits of X_0 until it hits a 0, and we can assume any bit of X_0 to be 0 or 1 randomly with equal probabilities of 1/2 each. Then, the probability of the carry bit to propagate less than or equal to t bits of X_0 from the lower side is
$P[\text{carry propagation} \leq t]$

$$= P[\text{no carry}] + \sum_{i=1}^{t} P[\text{carry}] \cdot P[\text{carry propagation} = i]$$

$$= P[\text{no carry}] + \sum_{i=1}^{t} P[\text{carry}] \cdot P[\text{first 0 occurs at } i\text{-th LSB of } X_0]$$

$$= (1 - p_c) + \sum_{i=1}^{t} p_c \cdot \frac{1}{2^i} = 1 - \frac{p_c}{2^t}.$$

Now, one may expect the probability of carry generated from the sum $(X_1 + Y)$ to be $p_c \approx 1/2$. A careful statistical modelling of the difference Y will reveal a better estimate of p_c. As we do not assume any distribution of Y here, we consider the trivial bound $p_c \leq 1$. Thus the probability of X and X_0, and hence X and X', sharing $l_X - H - t$ many MSBs is $1 - \frac{p_c}{2^t} \geq 1 - \frac{1}{2^t}$. ■

At this point, we can state and prove the main result of this section, the following theorem.

Theorem 5.6. *Let $S = \{0, \ldots, T\}$ and $k = \lfloor T/a \rfloor$ is odd. Suppose $U, V \subseteq S$ such that $U = \{0, \ldots, a\} \cup \{2a - t, \ldots, 3a\} \cup \cdots \cup \{(k-1)a - t, \ldots, ka\}$, $V = \{a - t, \ldots, 2a\} \cup \{3a - t, \ldots, 4a\} \cup \cdots \cup \{ka - t, \ldots, T\}$, where $p[i]$'s are known for $i \in U$ and $q[j]$'s are known for $j \in V$, as discussed before. Then, Algorithm 8 reconstructs T many contiguous most significant bits of one of the primes correctly in $O(k)$ steps with probability at least $P_{a,t}(T) = \left(1 - \frac{1}{2^t}\right)^{\lfloor T/a \rfloor}$.*

Proof. The success of Algorithm 8 relies on the correct construction of the approximations at various levels. The CORRECT function produces correct approximations with probability 1 given the known sets of bits U, V, as mentioned before. Hence, success probability of Algorithm 8 depends on the correctness of $\{q_{a-t-1}, p_{2a-t-1}, q_{3a-t-1}, \ldots, p_{(k-1)a-t-1}, q_{ka-t-1}\}$.

Let us first consider the case $p > q$. We know that in such a case, as p, q are of the same bitsize, one must have $\sqrt{N/2} < q < \sqrt{N} < p < \sqrt{2N}$. Suppose that there exists an approximation p_{ha} of p, sharing the MSBs $\{0, \ldots, ha\}$ for some $1 \leq h \leq k$. In this case, $|p - p_{ha}| < 2^{l_p - ha}$. Using p_{ha}, one constructs an approximation $q' = \lceil N/p_{ha} \rceil$ of q. Then we have $|q - q'| \approx \left| \frac{N}{p} - \frac{N}{p_{ha}} \right| = \frac{N}{pp_{ha}} |p - p_{ha}| < |p - p_{ha}| < 2^{l_p - ha}$, as $p, p_{ha} > \sqrt{N}$. If $p_{ha} < \sqrt{N}$ from the initial approximation, we reassign $p_{ha} = \lceil \sqrt{N} \rceil$ as a better approximation to p. The case $p < q$ produces an approximation q' of q with $|q - q'| < 2|p - p_{ha}| < 2^{l_p - ha + 1}$.

Then, we know for sure that $|q - q'| < 2^{l_p - ha + 1}$. Thus, by Lemma 5.5, setting $H = l_p - ha + 1$, we get that q and q' share the first $l_p - (l_p - ha + 1) - t = ha - t - 1$ MSBs with probability at least $P_t = 1 - \frac{1}{2^t}$. In other words, the probability that q' correctly represents q_{ha-t-1} is at least $P_t = 1 - \frac{1}{2^t}$. The probability of correctness is the same in case of the approximations of p by p_{ga-t-1} for all $1 < g < k$.

Now, the k approximations of p, q at different bit levels, as mentioned above, can be considered independent. Hence, the probability of success of Algorithm 8 in constructing T many contiguous MSBs of q (or p in another case) is at least $P_{a,t} = P_t^k = \left(1 - \frac{1}{2^t}\right)^k = \left(1 - \frac{1}{2^t}\right)^{\lfloor T/a \rfloor}$. ■

Once the most significant half of any one of the primes is known using Algorithm 8, one may use a lattice based method of to factorize $N = pq$. In this context, let us present the following result for factoring the RSA modulus N using Algorithm 8.

Corollary 5.7. *Let $S = \{0, \ldots, l_N/4\}$ and $k = \lfloor l_N/4a \rfloor$ is odd. Suppose $U, V \subseteq S$ such that $U = \{0, \ldots, a\} \cup \{2a - t, \ldots, 3a\} \cup \cdots \cup \{(k-1)a - t, \ldots, ka\}$, $V = \{a - t, \ldots, 2a\} \cup \{3a - t, \ldots, 4a\} \cup \cdots \cup \{ka - t, \ldots, l_N/4\}$, where $p[i]$'s are known for $i \in U$ and $q[j]$'s are known for $j \in V$, as discussed before. Then, one can factor N in* $\mathrm{poly}(\log \mathrm{N})$ *time with probability at least $P_{a,t} = \left(1 - \frac{1}{2^t}\right)^{\lfloor l_N/4a \rfloor}$.*

Proof. By setting $T = l_N/4$ in Theorem 5.6 we obtain that Algorithm 8 is able to recover contiguous $l_N/4$ many MSBs of one of the primes p, q in $O(l_N/4)$ steps with probability at least $P_{a,t} = \left(1 - \frac{1}{2^t}\right)^{\lfloor l_N/4a \rfloor}$. Since one call of CORRECT costs $O(l_N)$, thus the total time complexity is $O(\log^2 N)$.

Once we get these $l_N/4$ MSBs, that is the complete most significant half, of one of the primes, one can use the existing lattice based method [24] by Coppersmith to factor $N = pq$ in $\text{poly}(\log \text{N})$ time. ■

5.3.4 Experimental Results

We present some experimental results in Table 5.3 to support the claim in Theorem 5.6. The blocksize for known bits, i.e, a, and the approximation offset t are varied to obtain these results for $l_N = 1024$. The target size for reconstruction is $T = 256$ as the primes are 512 bits each. We have run the experiment 1000 times for each pair of fixed parameters a, t. The first value in each cell represents the experimental percentage of success in these cases and illustrate the practicality of our method. The second value in each cell is the theoretical probability of success obtained from Theorem 5.6.

a	$t = 1$	$t = 2$	$t = 3$	$t = 4$	$t = 5$
10	0, 0	2.5, 0.07	16.8, 3.55	41.5, 19.9	64.5, 45.2
20	1.8, 0.02	18.7, 3.17	44.5, 20.1	65.7, 46.1	*81.9, 68.3*
40	15.5, 1.6	42.8, 17.8	66.7, 44.9	*81.8, 67.9*	*90.8, 82.7*
60	29.1, 6.3	55.6, 31.6	*75.7, 58.6*	*86.6, 77.2*	*91.7, 88.1*
80	41.9, 12.5	66.4, 42.2	*82.9, 67.0*	*91.0, 82.4*	*95.7, 90.9*
100	50.6, 25.0	*74.4, 56.2*	*86.6, 76.6*	*93.7, 87.9*	*97.1, 93.8*

a	$t = 6$	$t = 7$	$t = 8$	$t = 9$	$t = 10$
10	82.1, 67.5	90.6, 82.2	95.0, 90.7	97.2, 95.2	-
20	*90.6, 82.8*	*94.8, 91.0*	*97.5, 95.4*	98.5, 97.7	99.3, 97.6
40	*95.2, 91.0*	*97.8, 95.4*	*98.6, 97.7*	*99.3, 98.8*	*99.9, 99.4*
60	*95.3, 93.9*	*97.4, 96.9*	*98.9, 98.4*	*99.5, 99.2*	*99.9, 99.7*
80	*98.3, 95.4*	*99.1, 97.7*	*99.4, 98.8*	*99.7, 99.4*	*100, 99.7*
100	*98.8, 96.9*	*99.6, 98.4*	*99.8, 99.2*	*99.9, 99.6*	*100, 99.8*

Table 5.3: Percentage of success of Algorithm 8 with 512-bit p, q, i.e., $l_N = 1024$.

One may note that our theoretical bounds on the probability (second value in each cell) is an underestimate compared to the experimental evidences (first value in each cell) in all the cases. This is because we have used the bound on the probability of carry loosely as $p_c \leq 1$ in Lemma 5.5, whereas a better estimate should have been $p_c \approx \frac{1}{2}$. As an example, let us check the case with $a = 40, t = 3$.

Here, the theoretical bound on the probability with $p_c \leq 1$ is 44.9% whereas with $p_c = 0.5$, the same bound comes as 67.9%. The experimental evidence of 66.7% is quite clearly closer to the second one. But we could not correctly estimate the value of p_c and hence opted for a safe (conservative) margin.

The results in *italic font* are of special interest. In these cases one can factorize N in $\mathrm{poly}(\log \mathrm{N})$ time, with probability greater than $\frac{1}{2}$ by knowing less than 70% of the bits of both the primes combined, that is, by knowing approximately just 35% of the bits of each prime p, q. Note that the result by Herrmann and May [51] requires about 70% of the bits of one of the primes in a similar case where the known bits are distributed over small blocks. Their result factorizes N in time exponential in the number of such blocks, whereas our method produces the same result in time polynomial in the number of blocks.

5.4 Conclusion

Our work discusses the factorization of RSA modulus N by reconstructing the primes from randomly known bits. The reconstruction method exploits the known bits to prune wrong branches of the search tree and reduces the total search space. We have revisited the work of Heninger and Shacham [50] in Crypto 2009 and provided a combinatorial model for the search where certain bits of the primes are known at random. This, combined with existing lattice based techniques, can factorize N given the knowledge of randomly chosen prime bits in the least significant halves of the primes. We also explain a lattice based strategy to remedy one of the shortcomings of the reconstruction algorithm. Moreover, we study how N can be factored given the knowledge of some blocks of bits in the most significant halves of the primes. We propose an algorithm that recovers the most significant halves of one or both the primes exploiting the known bits.

In this chapter, we have assumed the knowledge of certain fraction of bits of the RSA primes. However, one may not have such an explicit information regarding the primes, but may gain certain implicit knowledge instead. In the next chapter, we discuss RSA factorization in the light of such an implicit knowledge about the RSA primes. This problem is aptly termed as the 'implicit factorization problem'.

Chapter 6

Implicit Factorization

In PKC 2009, May and Ritzenhofen [86] presented interesting problems related to factoring large integers with some implicit hints. Consider two integers N_1, N_2 such that $N_1 = p_1q_1$ and $N_2 = p_2q_2$ where p_1, q_1, p_2, q_2 are primes and p_1, p_2 share t least significant bits (LSBs). It has been shown in [86] that when q_1, q_2 are primes of bitsize $\alpha \log_2 N$, then N_1, N_2 can be factored simultaneously if $t \geq 2\alpha$. This bound on t has further been improved when $N_1 = p_1q_1, N_2 = p_2q_2, \ldots, N_k = p_kq_k$ and all the p_i's share t many LSBs. The motivation of this problem comes from oracle based complexity of factorization problems. Prior to the work of [86], the main assumption in this direction was that an oracle explicitly outputs certain amount of bits of one prime. The idea of [86] deviates from this paradigm in the direction that none of the bits of the prime will be known, but some implicit information can be available regarding the prime. That is, an oracle, on input to N_1, outputs a different N_2 as described above. One application of implicit factorization is malicious key generation of RSA moduli, i.e. the construction of backdoored RSA moduli. A nice motivation towards the importance of this problem is presented in the introduction of [86].

In this chapter we first assume that p_1, p_2 share either t many MSBs or t many LSBs or total t many bits considering LSBs and MSBs together. Then we consider the same scenario for three primes p_1, p_2 and p_3. Further, we consider the case when the primes share certain amount of bits at the middle. Our approach in solving the problem is different from that of [86].

All the theoretical results are supported by experiments. In all the experiments we have performed, each of the Gröbner Basis calculation requires less than a second and we could successfully collect the root.

6.1 Implicit Factoring of Two Large Integers

Here we present the exact conditions on p_1, q_1, p_2, q_2 under which N_1, N_2 can be factored efficiently. Throughout this chapter, we will consider p_1, p_2 are primes of same bitsize and q_1, q_2 are primes of same bitsize. Also $N_1 = p_1q_1$ and $N_2 = p_2q_2$ are of same bitsize. We use N to represent an integer of same bitsize as of N_1, N_2.

6.1.1 The General Result

We first consider the case where some amount of LSBs as well as MSBs of p_1, p_2 are same. Based on this, we present the following generalized theorem.

Theorem 6.1. *Let $N_1 = p_1q_1$ and $N_2 = p_2q_2$, where p_1, q_1, p_2, q_2 are primes. Let $q_1, q_2 \approx N^\alpha$. Consider that $\gamma_1 \log_2 N$ many MSBs and $\gamma_2 \log_2 N$ many LSBs of p_1, p_2 are same. Let $\beta = 1 - \alpha - \gamma_1 - \gamma_2$. Under Assumption 1, one can factor N_1, N_2 in polynomial time if $1 - \frac{3}{2}\beta - 2\alpha \geq 0$ and*

$$-4\alpha^2 - 2\alpha\beta - \frac{1}{4}\beta^2 + 4\alpha + \frac{5}{3}\beta - 1 < 0.$$

Proof. It is given that $\gamma_1 \log_2 N$ many MSBs and $\gamma_2 \log_2 N$ many LSBs of p_1, p_2 are same. Thus, we can write $p_1 = N^{1-\alpha-\gamma_1}P_0 + N^{\gamma_2}P_1 + P_2$ and $p_2 = N^{1-\alpha-\gamma_1}P_0 + N^{\gamma_2}P_1' + P_2$. Thus, $p_1 - p_2 = N^{\gamma_2}(P_1 - P_1')$. Since $N_1 = p_1q_1$ and $N_2 = p_2q_2$, putting $p_1 = \frac{N_1}{q_1}$ and $p_2 = \frac{N_2}{q_2}$, we get $N^{\gamma_2}(P_1 - P_1') \cdot q_1q_2 - N_1q_2 + N_2q_1 = 0$. Thus we need to solve $f'(x, y, z) = N^{\gamma_2}xyz - N_1x + N_2y = 0$ whose root corresponding to x, y, z are $q_2, q_1, P_1 - P_1'$ respectively. Since there is no constant term in f', we define a new polynomial as follows.

$$f(x, y, z) = f'(x - 1, y, z) = N^{\gamma_2}xyz - N^{\gamma_2}yz - N_1x + N_1 + N_2y$$

The root (x_0, y_0, z_0) of f is $(q_2 + 1, q_1, P_1 - P_1')$.

Let X, Y, Z be upper bounds of $q_2 + 1, q_1, P_1 - P_1'$ respectively. As given in the statement of this theorem, we can take $X = N^\alpha, Y = N^\alpha, Z = N^\beta$. Following the extended strategy of Section 2.6.2, we get

$$S = \bigcup_{0 \le k_1 \le t} \{x^i y^j z^{k+k_1} : x^i y^j z^k \text{ is a monomial of } f^m\},$$
$$M = \{\text{monomials of } x^i y^j z^k f : x^i y^j z^k \in S\}.$$

It follows from the above definitions that

$$x^i y^j z^{k+k_1} \in S \Leftrightarrow \begin{cases} k = 0, \ldots, m, \\ j = k, \ldots, m, \\ i = 0, \ldots, m+k-j, \\ k_1 = 0, \ldots, t \end{cases}$$

$$x^i y^j z^k \in M \Leftrightarrow \begin{cases} k = 0, \ldots, m+1, \\ j = k, \ldots, m+1, \\ i = 0, \ldots, m+k-j+1, \\ k_1 = 0, \ldots, t \end{cases}$$

We exploit t many extra shifts of z where t is a non-negative integer. Our aim is to find two more polynomials f_0, f_1 that share the root $(q_2+1, q_1, P_1 - P_1')$ over the integers. From Section 2.6.2, we know that these polynomials can be found by lattice reduction if

$$X^{s_1} Y^{s_2} Z^{s_3} < W^s, \tag{6.1}$$

where $s = |S|$, $s_j = \sum_{x^{i_1} y^{i_2} z^{i_3} \in M \setminus S} i_j$ for $j = 1, 2, 3$, and

$$W = \|f(xX, yY, zZ)\|_\infty \ge N_1 X.$$

One can quite easily check the following.

$$s_1 = \frac{1}{2}m^3 + \frac{5}{2}m^2 + 4m + 2 + 2t + \frac{3}{2}m^2 t + \frac{7}{2}mt,$$
$$s_2 = \frac{5}{6}m^3 + 4m^2 + \frac{37}{6}m + 3 + 2t + \frac{3}{2}m^2 t + \frac{7}{2}mt,$$
$$s_3 = \frac{1}{2}m^3 + \frac{5}{2}m^2 + 4m + 2 + \frac{3}{2}t^2 + \frac{7}{2}t + \frac{3}{2}m^2 t + mt^2 + \frac{9}{2}mt,$$
$$s = \frac{1}{3}m^3 + \frac{3}{2}m^2 + \frac{13}{6}m + 1 + t + m^2 t + 2mt$$

Let $t = \tau m$, where τ is a nonnegative real number. Neglecting the lower order terms, from (6.1), we get the condition as

$$X^{\frac{m^3}{2}+\frac{3}{2}m^2t}Y^{\frac{5}{6}m^3+\frac{3}{2}m^2t}Z^{\frac{m^3}{2}+\frac{3}{2}m^2t+mt^2} < W^{\frac{m^3}{3}+m^2t}.$$

If we neglect the $o(m^3)$ terms after putting $t = \tau m$, the required condition becomes

$$\tau^2\beta + (2\alpha + \frac{3}{2}\beta - 1)\tau + (\alpha + \frac{\beta}{2} - \frac{1}{3}) < 0. \tag{6.2}$$

The optimal value of τ, to minimize the left hand side of (6.2), is $\frac{1-\frac{3}{2}\beta-2\alpha}{2\beta}$. Putting this optimal value, the required condition turns into

$$-64\alpha^2 - 32\alpha\beta - 4\beta^2 + 64\alpha + \frac{80}{3}\beta - 16 < 0.$$

Since $\tau \geq 0$, we need $1 - \frac{3}{2}\beta - 2\alpha \geq 0$. ∎

Remark 6.2. In the proof of Theorem 6.1, we have applied extra shifts over z. In fact, we tried with extra shifts on x, y too. However, we noted that the best theoretical as well as experimental results are achieved using extra shifts on z.

Looking at Theorem 6.1, it is clear that the efficiency of this factorization technique depends on the total amount of bits that are equal considering the most and least significant parts together. Let us present an example as follows.

Example 6.3. Let us consider 750-bit primes p_1 and p_2 as follows.
3804472805395186392319221660578496208300951856349524536490291627689678
4508873994603764416042481638726883020251099785398270595309011413652074
0662982896963184145937357387807661916268890545112759642350996744984148
6470692918256969 and
3804472805395186392319221660578496208300951856349524536490291627689216
6107608916018165804358808795724349647533346298650637180633006710173703
4446620984517063526577265988384407769443498510140109413281971152495463
7781487537874249.

Note that p_1, p_2 share 222 many MSBs and 220 many LSBs, i.e., 442 many bits in total. Further, q_1, q_2 are 250-bit primes
1788684495317470472835032661187758515078190921640698934821176591562967
327967, and
1706817658439540390758485693495273025642629127144779879402852507986344
279931 respectively.

Given N_1, N_2, with only the implicit information, we can factorize both of them efficiently. We use lattice of dimension 105 (parameters $m = 3, t = 2$) and the lattice reduction takes 6227.76 seconds.

In Theorem 6.1, we have considered that given the conditions, we can find f_0, f_1 by lattice reduction. However, in practice, one may get more polynomials. In our experiments, we use four polynomials f_0, f_1, f_2, f_3 that come after lattice reduction. Let J be the ideal generated by $\{f, f_0, f_1, f_2, f_3\}$ and let the corresponding Gröbner Basis be G. We studied the first three elements of G and found that one of them is of the form $y^a(x - \frac{q_2}{q_1}y - 1)$, where a is a small positive integer. We observed $a = 0, 1, 2$, considering the experiments listed in Sections 6.1 and 6.2. Note that $x_0 = q_2 + 1, y_0 = q_1$ is the root of this polynomial.

6.1.2 The MSB Case

The study when p_1, p_2 share some MSBs has not been considered in [86], which we present in this section. The following result arrives from Theorem 6.1, noting $\beta = 1 - \alpha - \gamma_1$.

Corollary 6.4. *Let $N_1 = p_1q_1$ and $N_2 = p_2q_2$, where p_1, q_1, p_2, q_2 are primes. Let $q_1, q_2 \approx N^\alpha$ and $|p_1 - p_2| < N^\beta$. Under Assumption 1, one can factor N_1, N_2 in polynomial time if* $1 - \frac{3}{2}\beta - 2\alpha \geq 0$ *and*

$$-4\alpha^2 - 2\alpha\beta - \frac{1}{4}\beta^2 + 4\alpha + \frac{5}{3}\beta - 1 < 0.$$

Thus fixing the bitsize of N, if the bitsize of q_1, q_2 (i.e., α) increases, then the equality of the MSBs of p_1, p_2 should increase (i.e., β should decrease) for efficient factorization of N_1, N_2. We have performed detailed experiments for this MSB case, but we skip the experimental results as the results are similar to what is explained in Section 6.1.3 for the LSB case. We present the experimental evidences for the LSB case in detail as we can compare the results with that of [86].

6.1.3 The LSB Case

Let us first explain the ideas presented in [86]. Let $N_1 = p_1q_1$ and $N_2 = p_2q_2$. In [86, Section 3], it has been explained that if $q_1, q_2 \approx N^\alpha$, then for efficient factorization of N_1, N_2, the primes p_1, p_2 need to share at least $2\alpha \log_2 N$ many LSBs. Our strategy is different from the strategy of [86] and we follow the result of Theorem 6.1.

Corollary 6.5. *Let $N_1 = p_1q_1$ and $N_2 = p_2q_2$, where p_1, q_1, p_2, q_2 are primes. Let $q_1, q_2 \approx N^\alpha$. Consider that $\gamma \log_2 N$ many LSBs of p_1, p_2 are same, i.e., $p_1 \equiv p_2 \pmod{N^\gamma}$. Let $\beta = 1 - \alpha - \gamma$. Under Assumption 1, one can factor N_1, N_2 in polynomial time if $1 - \frac{3}{2}\beta - 2\alpha \geq 0$ and*

$$-4\alpha^2 - 2\alpha\beta - \frac{1}{4}\beta^2 + 4\alpha + \frac{5}{3}\beta - 1 < 0.$$

The numerical values related to the theoretical result of [86] and Corollary 6.5 as well as the experimental results are presented in Table 6.1. By NUB we mean "Numerical Upper Bound" for the parameters. The experimental results in each row are based on one run where N_1, N_2 are 1000-bit integers. The experiments in Table 6.1 are performed with lattice dimension 46 (parameters $m = 2, t = 1$) and each lattice reduction takes around 30 seconds.

To explain the results of Table 6.1, let us concentrate on the first row. As $\alpha = 0.23$, we have q_1, q_2 are of bitsize $0.23 \times 1000 = 230$. Thus, p_1, p_2 are of bitsize $1000 - 230 = 770$. Now, the numerical value from Corollary 6.5 tells that $770 - 0.255 \times 1000 = 515$ many LSBs of p_1, p_2 need to be equal to have efficient factorization of N_1, N_2 simultaneously. However, the the experimental result is more encouraging which shows that only $770 - 0.336 \times 1000 = 434$ many LSBs of p_1, p_2 need to be equal.

α	NUB of β following [86]	NUB of β following Corollary 6.5	Results achieved for β from experiments
0.23	0.31	0.255	0.336
0.24	0.28	0.239	0.314
0.25	0.25	0.225	0.296
0.26	0.22	0.210	0.268
0.27	0.19	0.196	0.251

Table 6.1: Values of α, β for which N_1, N_2 can be factored efficiently.

Remark 6.6. From Table 6.1 it is clear that we get much better results in the experiments than the theoretical bounds. This is because for the parameters we consider here, the shortest vectors may belong to some sublattice. However, the theoretical calculation in Theorem 6.1 cannot capture that and further, identifying such optimal sublattice seems to be difficult. This kind of scenario where experimental results perform better than theoretical estimates, has earlier been observed in [66, Section 7.1], too.

However, we like to point out that the experimental results are better than our theoretical results for a certain parameter range, i.e., when $\log_N q_i < 0.35$ for lattice parameters $m = 2, t = 1$. If one looks at Figure 6.1, it can be observed that in the remaining range, the experimental results are worse than the theoretical results. We are yet to explain this phenomenon clearly. It may happen that in this range, the shortest vectors do not belong to some sublattice and the experimental results are worse due to limited lattice dimensions in practice.

In our notation, the number of MSBs in each of p_1, p_2 that are unshared is $\beta \log_2 N$. Thus $\beta = (1-\alpha) - 2\alpha = 1 - 3\alpha$, where $\alpha \log_2 N$ is the bitsize of q_1, q_2. Table 6.1 identifies that while our theoretical result is either worse or better than that of [86] based on the values of α, the experimental results that we obtain are always better than [86]. In the introduction of [86], it has been pointed out that for 250-bit q_1, q_2 and 750-bit p_1, p_2, the primes p_1, p_2 need to share 502 many LSBs. We have implemented the strategy of [86] and observed similar results.

On the other hand, our experimental results are better as evident from Table 6.1, when $\alpha = 0.25$. In fact, we experimented with a higher lattice dimension as explained in Example 6.7 and our strategy requires only 438 many LSBs to be shared in p_1, p_2. This result is better than [86], where 502 many LSBs have been shared.

Example 6.7. Here we consider 750-bit primes p_1 and p_2
5895254139679228077142387416586490039613283191466241401307494261824605
9669084690420722716275439075281566487074700579275565739610880278518405
2727673670100332217332947627771123511694759914704886336601966226161930
4575961682668297 and
4392119049423447468690947059559090008016802774014559696547174955333794
4652342861564934625350120675407265601224878945969002652471346685040069
8503016817420142894918107629408891591088684705545955400539206624614659
4876423472933641 respectively.

Note that p_1, p_2 share 438 many LSBs. Further, q_1, q_2 are 250-bit primes
9160109778146430106669507839679796567724449698019266905896747910430591

04197 and
1587061752065032326280290326014711341044827082150757395718254111544994
945759 respectively.
Given N_1, N_2, with only the implicit information, we can factorize both of them efficiently. We use lattice of dimension 105 (parameters $m = 3, t = 2$) and the lattice reduction takes 7273.52 seconds.

We now present a detailed discussion how our strategy compares with that of [86]. It is indeed clear from Table 6.1, that our experimental results provide better performance than the theoretical results presented in this chapter as well as in [86]. Moreover, we explain how the technique of [86] and our strategy perform in terms of theoretical results.

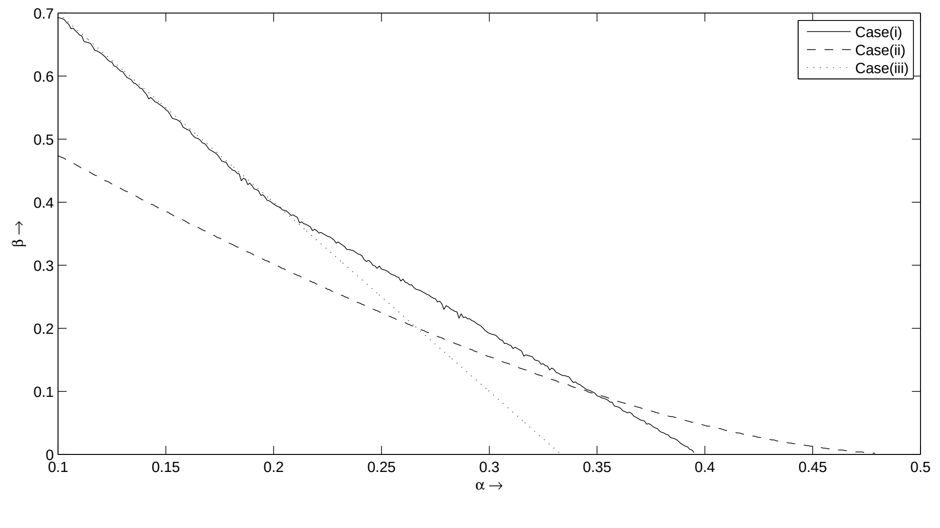

Figure 6.1: Comparison of our experimental (case (i)) and theoretical results (case (ii)) with that of [86] (case (iii)).

Let us first concentrate on the formula $\beta = 1-3\alpha$, that characterizes the bound on the primes for efficient factoring in [86]. When $\alpha = \frac{1}{3}$, β becomes zero, implying that p_1, p_2 need to have all the bits shared. Thus, the upper bound on the smaller primes q_1, q_2 is $N^{\frac{1}{3}}$, where shared LSBs in p_1, p_2 help in efficient factoring.

However, in our case, the bound on the primes is characterized by $-4\alpha^2 - 2\alpha\beta - \frac{1}{4}\beta^2 + 4\alpha + \frac{5}{3}\beta - 1 < 0$ provided $1 - \frac{3}{2}\beta - 2\alpha \geq 0$. We find that β becomes zero when $\alpha = \frac{1}{2}$. Thus in our case, the upper bound on smaller primes q_1, q_2 is $N^{\frac{1}{2}}$, where sharing of LSBs in p_1, p_2 helps in efficient factoring.

Theoretically, our method starts performing better, i.e., β in our case is greater than that of [86], when $\alpha \geq 0.266$. Thus for $q_1, q_2 \geq N^{0.266}$, our method will require less number of LSBs of p_1, p_2 to be equal than that of [86]. This is also presented in Figure 6.1. The numerical values of the theoretical results are generated using the formulae $\beta = 1 - 3\alpha$ for [86] and Corollary 6.5 for our case. The experimental results are generated by one run in each case with lattice dimension 46 (parameters $m = 2, t = 1$) for 1000 bits N_1, N_2. The values of α are considered in $[0.1, 0.5]$, in a step of 0.01. Referring to Figure 6.1, we like to reiterate that our experimental results outperforms the theoretical results presented by us as well as in [86].

The next example considers the primes p_1, p_2 of 650 bits and q_1, q_2 of 350 bits. This is to demonstrate how our method works experimentally for larger q_1, q_2.

Example 6.8. Here we consider 650-bit primes p_1 and p_2
3137055889901096909077531458327171120014878453383152732512530257276363168292785241218747273712763711037157637711966791419526760377688029885676273831127205611509045644179511599106554189421550654601 and
2451436010930813903814310506086633020716328387757587411726661941127209321216740545001634090447037011441230660481097503555238640524767415889480913091786359014934176726120292021849927924906510931081.

Note that p_1, p_2 share 531 many LSBs. Further, q_1, q_2 are 350-bit primes
1851420588886517478939713595303492404190382112791551597798571143339516233613445774636517955322189132943773 and
2258350305148478218870025161325667637658623408855938899014758338949666508115561055599847183651567682695481 respectively.
Given N_1, N_2, with only the implicit information, we can factorize both of them efficiently. We use lattice of dimension 105 (parameters $m = 3, t = 2$) and the lattice reduction takes 15016.42 seconds.

Though our result does not generalize for the case where $N_1, N_2, \ldots, N_k$, we like to compare the result of Example 6.8 with [86, Table 1, Section 6.2] when $\alpha = 0.35$ and N is of 1000 bits. This is presented in Table 6.2. One may note that the idea of [86] requires 10 many N_i's as the input where $N_i = p_i q_i$, $1 \leq i \leq 10$. In such a case, 391 many LSBs need to be same for $p_1, \ldots, p_{10}$. On the other hand, we require higher number of LSBs, i.e., 531 to be same, but only N_1, N_2 are needed.

The analysis of our results related to LSBs, presented in this section, will apply similarly for our analysis related to MSBs or LSBs and MSBs taken together as

Reference	l_{p_i}, l_{q_i}	# of N_i's required	# of shared bits
[86, Table 1, Section 6.2]	650, 350	10	391
Our Example 6.8	650, 350	2	531

Table 6.2: Comparison of experimental results when $\alpha = 0.35$.

explained earlier.

6.1.4 Implicit Factorization problem when $k = 3$

We have already introduced the general problem for k many RSA moduli in the introduction, where $N_1 = p_1q_1, N_2 = p_2q_2, \ldots, N_k = p_kq_k$, where $p_1, p_2, \ldots, p_k$ and $q_1, q_2, \ldots, q_k$ are primes. The primes $p_1, p_2, \ldots, p_k$ are of the same bitsize and so are $q_1, q_2, \ldots, q_k$. It is considered that certain portions of bit pattern in $p_1, p_2, \ldots, p_k$ are common. Under such a situation, it is studied when it is possible to factor $N_1, N_2, \ldots, N_k$ efficiently. Here we consider the case for $k = 3$.

Theorem 6.9. *Let N be of the same bitsize as the RSA moduli N_1, N_2, N_3. Let $q_1, q_2, q_3 \approx N^\alpha$. Consider that $\gamma_1 \log_2 N$ many MSBs and $\gamma_2 \log_2 N$ many LSBs of p_1, p_2, p_3 are the same. Let $\beta = 1 - \alpha - \gamma_1 - \gamma_2$. Then, under Assumption 1, one can factor N_1, N_2, N_3 in polynomial time if* $10\alpha + 5\beta - 4 \leq 0$.

Proof. It is given that $\gamma_1 \log_2 N$ many MSBs and $\gamma_2 \log_2 N$ many LSBs of p_1, p_2, p_3 are the same. Thus, we can write $p_1 = N^{1-\alpha-\gamma_1}P_0 + N^{\gamma_2}P_1 + P_2$, $p_2 = N^{1-\alpha-\gamma_1}P_0 + N^{\gamma_2}P_1' + P_2$ and $p_3 = N^{1-\alpha-\gamma_1}P_0 + N^{\gamma_2}P_1'' + P_2$. Thus, $p_1 - p_2 = N^{\gamma_2}(P_1 - P_1')$. Since $N_1 = p_1q_1$ and $N_2 = p_2q_2$, putting $p_1 = \frac{N_1}{q_1}$ and $p_2 = \frac{N_2}{q_2}$, we get

$$N_1q_2 - N_2q_1 = N^{\gamma_2}(P_1 - P_1') \cdot q_1q_2. \tag{6.3}$$

Similarly, we have

$$N_1q_3 - N_3q_1 = N^{\gamma_2}(P_1 - P_1'') \cdot q_1q_3. \tag{6.4}$$

Now, multiplying Equation (6.3) by N_3 and Equation (6.4) by N_2 and then subtracting, we get $N_1N_3q_2 - N_1N_2q_3 - N^{\gamma_2}(P_1 - P_1') \cdot q_1q_2N_3 + N^{\gamma_2}(P_1 - P_1'') \cdot q_1q_3N_2 = 0$. Thus we need to solve $f'(x, y, z, v, t) = N_1N_3y - N_1N_2z - N_3N^{\gamma_2}xyv + N_2N^{\gamma_2}xzt = 0$ whose root corresponding to x, y, z, v, t are $q_1, q_2, q_3, P_1 - P_1', P_1 - P_1''$ respectively. Since there is no constant term in f', we define a new polynomial

$f(x,y,z,v,t) = f'(x,y+1,z,v,t) = N_1N_3 + N_1N_3y - N_1N_2z - N_3N^{\gamma_2}xyv - N_3N^{\gamma_2}xv + N_2N^{\gamma_2}xzt$. The root (x_0,y_0,z_0,v_0,t_0) of f is $(q_1, q_2-1, q_3, P_1-P_1', P_1-P_1'')$. The idea of modifying the polynomial with a constant term was introduced in [28, Appendix A] and later used in [65] which we follow here.

Let X,Y,Z,V,T be the upper bounds of $q_1, q_2-1, q_3, P_1-P_1', P_1-P_1''$ respectively. As given in the statement of this theorem, one can take $X = N^\alpha, Y = N^\alpha, Z = N^\alpha, V = N^\beta, T = N^\beta$. Following the basic strategy of Section 2.6.2 ,

$$\begin{aligned} S &= \{x^{i_1}y^{i_2}z^{i_3}v^{i_4}t^{i_5} : x^{i_1}y^{i_2}z^{i_3}v^{i_4}t^{i_5} \text{ is a monomial of } f^m\}, \\ M &= \{\text{monomials of } x^{i_1}y^{i_2}z^{i_3}v^{i_4}t^{i_5}f : x^{i_1}y^{i_2}z^{i_3}v^{i_4}t^{i_5} \in S\}. \end{aligned}$$

It follows that,

$$x^{i_1}y^{i_2}z^{i_3}v^{i_4}t^{i_5} \in S \Leftrightarrow \begin{cases} i_3 = 0,\ldots,m, \\ i_5 = 0,\ldots,i_3, \\ i_4 = 0,\ldots,m-i_3, \\ i_2 = 0,\ldots,m-i_3, \\ i_1 = i_4+i_5, \end{cases}$$

$$x^{i_1}y^{i_2}z^{i_3}v^{i_4}t^{i_5} \in M \Leftrightarrow \begin{cases} i_3 = 0,\ldots,m+1, \\ i_5 = 0,\ldots,i_3, \\ i_4 = 0,\ldots,m+1-i_3, \\ i_2 = 0,\ldots,m+1-i_3, \\ i_1 = i_4+i_5. \end{cases}$$

From [65], we know that these polynomials can be found by lattice reduction if

$$X^{s_1}Y^{s_2}Z^{s_3}V^{s_4}T^{s_5} < W^s, \tag{6.5}$$

where $s = |S|$, $s_j = \sum_{x^{i_1}y^{i_2}z^{i_3}v^{i_4}t^{i_5} \in M\setminus S} i_j$,
for $j = 1,2,3,4,5$, and $W = \|f(xX,yY,zZ,vV,tT)\|_\infty \geq N_1N_3Y \approx N^{2+\alpha}$.

Thus, we have the following.

$$s = \sum_{i_3=0}^{m}\sum_{i_5=0}^{i_3}\sum_{i_4=0}^{m-i_3}\sum_{i_2=0}^{m-i_3} 1,$$

$$s_1 = \sum_{i_3=0}^{m+1}\sum_{i_5=0}^{i_3}\sum_{i_4=0}^{m+1-i_3}\sum_{i_2=0}^{m+1-i_3}(i_4+i_5) - \sum_{i_3=0}^{m}\sum_{i_5=0}^{i_3}\sum_{i_4=0}^{m-i_3}\sum_{i_2=0}^{m-i_3}(i_4+i_5),$$

$$s_j = \sum_{i_3=0}^{m+1}\sum_{i_5=0}^{i_3}\sum_{i_4=0}^{m+1-i_3}\sum_{i_2=0}^{m+1-i_3} i_j - \sum_{i_3=0}^{m}\sum_{i_5=0}^{i_3}\sum_{i_4=0}^{m-i_3}\sum_{i_2=0}^{m-i_3} i_j, \text{ for } j = 2,3,4,5.$$

Simplifying, one can check that

$$s = s_5 = \frac{1}{12}m^4 + \frac{2}{3}m^3 + \frac{23}{12}m^2 + \frac{7}{3}m + 1,$$
$$s_1 = \frac{5}{24}m^4 + \frac{7}{4}m^3 + \frac{127}{24}m^2 + \frac{27}{4}m + 3,$$
$$s_2 = s_4 = \frac{1}{8}m^4 + \frac{13}{12}m^3 + \frac{27}{8}m^2 + \frac{53}{12}m + 2,$$
$$s_3 = \frac{1}{6}m^4 + \frac{4}{3}m^3 + \frac{23}{6}m^2 + \frac{14}{3}m + 2.$$

Neglecting the lower order terms and putting the values of X, Y, Z, V, T as well as the lower bound of W, from (6.5), we get the condition as

$$\frac{5}{12}\alpha + \frac{5}{24}\beta - \frac{1}{6} < 0 \text{ i.e., } 10\alpha + 5\beta - 4 < 0. \tag{6.6}$$

That is, when this condition holds, according to [65], we get four polynomials f_0, f_1, f_2, f_3 such that $f_0(x_0, y_0, z_0, v_0, t_0) = f_1(x_0, y_0, z_0, v_0, t_0)$ $= f_2(x_0, y_0, z_0, v_0, t_0) = f_3(x_0, y_0, z_0, v_0, t_0) = 0$. Under Assumption 1, we can extract x_0, y_0, z_0, v_0, t_0 in polynomial time. ■

In Theorem 6.9, we consider the Assumption 1. Let us now clarify how it actually works. In the proof of Theorem 6.9, we consider that we will be able to get at least four polynomials f_0, f_1, f_2, f_3 along with f, that share the integer root. In experiments we found more than 4 polynomials (other than f) after the LLL algorithm that share the same root. We calculate $f_4 = R(f, f_0), f_5 = R(f, f_1)$ and then $f_6 = R(f_4, f_5)$. We always find a factor

$$-\frac{t_0}{\gcd(t_0, v_0)}v + \frac{v_0}{\gcd(t_0, v_0)}t$$

of f_6, though we do not have a theoretical proof for that. In all the cases, we find $\gcd(t_0, v_0) \leq 2$. After getting t_0, v_0, we define a new polynomial $f_7(y, z) = f_4(y, z, v_0, t_0)$. We always find a factor $q_3y - q_2z + q_3$ of f_7. From this we can find $(y_0, z_0) = (q_2 - 1, q_3)$. Finally, putting the values of y_0, z_0, v_0, t_0 in f, we obtain

$x_0 = q_1$.

α	γ_1	γ_2	β (Theory)	Time	β (Experimental Values)			
			our	(in sec.)	our	[86]	[112]	[40]
0.25	0.2	0.175	0.3	1.64	0.375	0.372	0.383	0.372
0.3	0.225	0.225	0.2	1.55	0.25	0.248	0.269	0.246
0.35	0.28	0.26	0.1	1.52	0.11	0.125	0.151	0.123

Table 6.3: Theoretical and experimental values of α, β for which N_1, N_2, N_3 can be factored efficiently. Results using our technique are obtained with the lattice dimension 20.

From Table 6.3 it may be noted that we get much better results in the experiments than the theoretical bounds. This is because, for the parameters we consider here, the shortest vectors may belong to some sublattice. However, the theoretical calculation in Theorem 6.9 cannot capture that and further, identifying such optimal sublattice seems to be difficult. This kind of scenario, where experimental results perform better than theoretical estimates, has earlier been observed in [66, Section 7.1], too.

Our experimental results show that the total number of bits to be shared in the primes for implicit factorization is of similar order to the requirements in [40,86] for three RSA moduli, i.e., $k = 3$. Following [86] (respectively [40]), the experiments are done when the least (respectively most) significant bits of the primes are same. However, in our case, the experiments are done considering some portion of least significant bits as well as some portion of most significant bits are same, which is not covered in [40, 86].

6.2 Two Primes with Shared Contiguous Portion of Bits at the Middle

Now we consider the case when p_1, p_2 share a contiguous portion of bits at the middle. Theorem 6.10 can be generalized to primes having two blocks of unshared bits at any position with a similar kind of analysis. However, only the case of shared bits in the middle of the primes is considered here.

Theorem 6.10. *Let $N_1 = p_1q_1$ and $N_2 = p_2q_2$, where p_1, q_1, p_2, q_2 are primes. Let $q_1, q_2 \approx N^{\alpha}$. Consider that a contiguous portion of bits of p_1, p_2 are same* *at the middle leaving the $\gamma_1 \log_2 N$ many MSBs and $\gamma_2 \log_2 N$ many LSBs. Then under Assumption 1, we can factor both N_1, N_2 if there exist $\tau_1, \tau_2 \geq 0$ for which $h(\tau_1, \tau_2, \alpha, \gamma_1, \gamma_2) < 0$ where $\gamma = \max\{\gamma_1, \gamma_2\}$ and*

$$\begin{aligned} h(\tau_1, \tau_2, \alpha, \gamma_1, \gamma_2) = & \left(3\tau_1\tau_2 + \frac{7}{3}\tau_1 + \frac{7}{3}\tau_2 + \frac{17}{24}\right)\alpha \\ & + \left(\tau_1^2\tau_2 + \frac{3}{2}\tau_1\tau_2 + \frac{3}{4}\tau_1^2 + \frac{2}{3}\tau_1 + \frac{2}{3}\tau_2 + \frac{1}{6}\right)\gamma_1 \\ & + \left(\tau_1\tau_2^2 + \frac{3}{2}\tau_1\tau_2 + \frac{3}{4}\tau_2^2 + \frac{2}{3}\tau_1 + \frac{2}{3}\tau_2 + \frac{1}{6}\right)\gamma_2 \\ & - \left(\tau_1\tau_2 + \frac{\tau_1}{2} + \frac{\tau_2}{2} + \frac{1}{8}\right)(1+\gamma) < 0. \end{aligned}$$

Proof. We can write $p_1 = N^{1-\alpha-\gamma_1}p_{10} + N^{\gamma_2}p_{11} + p_{12}$, and $p_2 = N^{1-\alpha-\gamma_1}p_{20} + N^{\gamma_2}p_{11} + p_{22}$. So $p_1 - p_2 = N^{1-\alpha-\gamma_1}(p_{10} - p_{20}) + (p_{12} - p_{22})$. Since $p_1 = \frac{N_1}{q_1}$ and $p_2 = \frac{N_2}{q_2}$ we have $N_1q_2 - N_2q_1 - N^{1-\alpha-\gamma_1}(p_{10} - p_{20})q_1q_2 - (p_{12} - p_{22})q_1q_2 = 0$. Thus we need to solve $f'(x, y, z, v) = N_1x - N_2y - N^{1-\alpha-\gamma_1}xyz - xyv = 0$ whose root corresponding to x, y, z, v are $q_2, q_1, p_{10} - p_{20}, p_{12} - p_{22}$ respectively. Since there is no constant term in f', we define a new polynomial

$$\begin{aligned} f(x, y, z, v) &= f'(x-1, y, z, v) \\ &= N_1x - N_2y - N_1 - N^{1-\alpha-\gamma_1}xyz \\ &\quad + N^{1-\alpha-\gamma_1}yz - xyv + yv. \end{aligned}$$

The root (x_0, y_0, z_0, v_0) of f is $(q_2 + 1, q_1, p_{10} - p_{20}, p_{12} - p_{22})$. Let $X = N^\alpha, Y = N^\alpha, Z = N^{\gamma_1}, V = N^{\gamma_2}$. Then we can take X, Y, Z, V as the upper bound of x_0, y_0, z_0, v_0 respectively.

Following the extended strategy of Section 2.6.2, we have the following definitions of S, M, where m, t_1, t_2 are non-negative integers.

$$S = \bigcup_{0 \leq j_1 \leq t_1, 0 \leq j_2 \leq t_2} \{x^{i_1}y^{i_2}z^{i_3+j_1}v^{i_4+j_2} : x^{i_1}y^{i_2}z^{i_3}v^{i_4} \text{ is a monomial of } f^m\},$$

$$M = \{\text{monomials of } x^{i_1}y^{i_2}z^{i_3}v^{i_4}f : x^{i_1}y^{i_2}z^{i_3}v^{i_4} \in S\}.$$

It follows from the above definitions that

$$x^{i_1}y^{i_2}z^{i_3+j_1}v^{i_4+j_2} \in S \Leftrightarrow \begin{cases} i_4 = 0,\ldots,m, \\ i_3 = 0,\ldots,m-i_4, \\ i_2 = i_3+i_4,\ldots,m, \\ i_1 = 0,\ldots,m-i_2+i_3+i_4, \\ j_1 = 0,\ldots,t_1, \\ j_2 = 0,\ldots,t_2 \end{cases}$$

$$x^{i_1}y^{i_2}z^{i_3+j_1}v^{i_4+j_2} \in M \Leftrightarrow \begin{cases} i_4 = 0,\ldots,m+1, \\ i_3 = 0,\ldots,m+1-i_4, \\ i_2 = i_3+i_4,\ldots,m+1, \\ i_1 = 0,\ldots,m+1-i_2+i_3+i_4, \\ j_1 = 0,\ldots,t_1, \\ j_2 = 0,\ldots,t_2 \end{cases}$$

From Section 2.6.2, we know that these polynomials can be found by lattice reduction if

$$X^{s_1}Y^{s_2}Z^{s_3}V^{s_4} < W^s, \tag{6.7}$$

where $s = |S|$, $s_j = \sum_{x^{i_1}y^{i_2}z^{i_3}v^{i_4} \in M \setminus S} i_j$ for $j = 1,\ldots,4$, and

$$W = \|f(xX, yY, zZ)\|_\infty \geq \max\{N_1X, N_2Y\} = N^{1+\gamma}.$$

One can easily check the following.

$$\begin{aligned} s_1 &= \frac{3}{2}t_1t_2m^2 + t_1m^3 + t_2m^3 + \frac{1}{4}m^4 + o(m^4), \\ s_2 &= \frac{3}{2}t_1t_2m^2 + \frac{4}{3}t_1m^3 + \frac{4}{3}t_2m^3 + \frac{11}{24}m^4 + o(m^4), \\ s_3 &= t_1^2t_2m + \frac{3}{2}t_1t_2m^2 + \frac{3}{4}t_1^2m^2 + \frac{2}{3}t_1m^3 + \frac{2}{3}t_2m^3 + \frac{1}{6}m^4 + o(m^4), \\ s_4 &= t_1t_2^2m + \frac{3}{2}t_1t_2m^2 + \frac{3}{4}t_2^2m^2 + \frac{2}{3}t_1m^3 + \frac{2}{3}t_2m^3 + \frac{1}{6}m^4 + o(m^4), \\ s &= t_1t_2m^2 + \frac{1}{2}t_1m^3 + \frac{1}{2}t_2m^3 + \frac{1}{8}m^4 + o(m^4). \end{aligned}$$

For a given integer m, let $t_1 = \tau_1 m$ and $t_2 = \tau_2 m$. Then substituting the values of X, Y, Z, V and lower bound of W in (6.7) and neglecting the lower order terms of s_j we get the required condition. ∎

When $\alpha, \gamma_1, \gamma_2$ are available, we need to take the partial derivative of h with respect to τ_1, τ_2 and equate each of them to 0 to get non-negative solutions of τ_1, τ_2. Given any pair of such non-negative solutions, if h is less than zero, then N_1, N_2 can be factored in polynomial time. When $\gamma_1 = \gamma_2 = \gamma$, one can consider t_1 to be equal to t_2. In that case we get the following result.

Corollary 6.11. *Let $N_1 = p_1q_1$ and $N_2 = p_2q_2$, where p_1, q_1, p_2, q_2 are primes. Let $q_1, q_2 \approx N^\alpha$. Consider that a contiguous portion of bits of p_1, p_2 are same at the middle leaving $\gamma \log_2 N$ many MSBs and as well as LSBs. Then under Assumption 1, we can factor both N_1, N_2 if there exists $\tau \geq 0$ for which*

$$2\gamma\tau^3 + \left(3\alpha + \frac{7}{2}\gamma - 1\right)\tau^2 + \left(\frac{14}{3}\alpha + \frac{5}{3}\gamma - 1\right)\tau + \left(\frac{17}{24}\alpha + \frac{5}{24}\gamma - \frac{1}{8}\right) < 0.$$

In Table 6.4, we present some numerical values of $\alpha, \gamma_1, \gamma_2$ following Theorem 6.10 for which N_1, N_2 can be factored in polynomial time. It is clear from

α	γ_1	γ_2
0.25	0.019	0.019
0.25	0.010	0.035
0.20	0.061	0.061
0.20	0.052	0.080
0.15	0.146	0.146
0.15	0.137	0.176
0.10	0.277	0.277
0.10	0.268	0.314

Table 6.4: Values of $\alpha, \gamma_1, \gamma_2$ for which N_1, N_2 can be factored efficiently.

Table 6.4 that the requirement of bits at the middle of p_1, p_2 to be same is quite high compared to the case presented in Section 6.1, where we have considered that MSBs and/or LSBs are same. Thus, the kind of lattice based technique we consider in this chapter works more efficiently when the bits of p_1, p_2 are same at MSBs and/or LSBs compared to the case when some contiguous bits at the middle are same. Let us present an experimental result in this direction.

Example 6.12. In this experiment, we consider 850-bit primes p_1 and p_2
6010063291745673411630355586520987527501854123507752031634410338922261
3252007908441224870562785926663459589770176998171796837046320523689364
1193687536324043967528101378987325022248122037708305617873964005637745

98643554294853982575818856214151329271376535 73

and

598482564187093158582338222096292634422067053255440393335210567557106 6
672703603259730323516347307682331134375921535484093155453663109803357 4
699693202605043239631638906892532523449394047385276940671493424035346 9
41864114078348939007323033801466200125284213 39.

Note that, p_1, p_2 share middle 504 many bits (leaving 177 bits from the least significant side). Further, q_1, q_2 are 150-bit primes
1038476608131498405684472704928794724111541861 and
1281887704228770097092001008195142506836912053 respectively.
Given N_1, N_2, with only the implicit information, we can factorize both of them efficiently. We use lattice of dimension 70 (parameters $m = 1, t_1 = 1, t_2 = 1$) and the lattice reduction takes 175.83 seconds.

Referring to Theorems 6.1,6.10 together, one may be tempted to consider the case that a few contiguous intervals of bits are same in p_1, p_2. However, in such a scenario, the polynomials contain increased number of variables as well as monomials. Thus, encouraging results cannot be obtained in this method.

6.3 Exposing a Few MSBs of One Prime

In this section we study what actually happens when a few bits of q_1 or q_2 gets exposed. Without loss of generality, consider that a few MSBs of q_2 are available. In this case, q_2 can be written as $q_{20} + q_{21}$, where q_{20} is known and it takes care of the higher order bits of q_2. In such a case we can generalize Theorem 6.1 as follows. We do not write the proof as it is similar to that of Theorem 6.1.

Theorem 6.13. *Let $N_1 = p_1q_1$ and $N_2 = p_2q_2$, where p_1, q_1, p_2, q_2 are primes. Let $q_1, q_2 \approx N^{\alpha}$. Suppose q_{20} is known such that $|q_2 - q_{20}| \leq N^{\delta}$. Consider that $\gamma_1 \log_2 N$ many MSBs and $\gamma_2 \log_2 N$ many LSBs of p_1, p_2 are same. Let $\beta = 1 - \alpha - \gamma_1 - \gamma_2$. Under Assumption 1, one can factor N_1, N_2 in polynomial time if*

$$-18\delta^2 - 12\delta\alpha - 2\alpha^2 - 20\delta\beta + 4\alpha\beta - 2\beta^2 + 24\delta + 8\alpha + \frac{40}{3}\beta - 8 < 0,$$

provided $1 - \frac{3}{2}\beta - \frac{\alpha}{2} - \frac{3}{2}\delta \geq 0$.

Remark 6.14. If one puts $\delta = \alpha$ in Theorem 6.13, then the statement of Theorem 6.1 appears immediately.

Since a few MSBs of q_2 are known, we have $\delta < \alpha$. So we get increased upper bound on β here than the bound of β in Theorem 6.1. This has two implications:

1. Knowing few MSBs of q_2 requires less number of bits to be shared in p_1, p_2.
2. Keeping the same number of bits to be shared, knowing few MSBs of q_2 may increase the bound on α.

Now we summarize the effect of knowing a few bits of q_2 in Table 6.5. We like to refer Example 6.8 (presented in Section 6.1.3) for the corresponding case when none of the bits of q_2 is known. Examples 6.15, 6.16 consider the cases when a few MSBs of q_2 are known.

Example	l_{q_i}	MSBs known in q_2	l_{p_i}	Number of shared LSBs in p_1, p_2
Example 6.8	350	None	650	531
Example 6.15	350	30	650	524
Example 6.16	360	30	640	531

Table 6.5: Effect of knowing a few MSBs of q_2.

Example 6.15. In this experiment, we consider 650-bit primes p_1 and p_2
2775767775790591046985561869233542957775274354805022086049845364080279
5440244280573885472845851243011213318288730705557808640463498077884662
12373756028939921330525120578684995831943366630235878261 and
3801183388452299712401650983522590415108473350487714213507812938100060
4705988087731376063723045978466857969967252244163227348367433438951447
21500290301718857429556533325996125827735316520359080821.

Note that, p_1, p_2 share 524 many LSBs. Further, q_1, q_2 are 350-bit primes
2209314625764053417894986194593494065883964634454958947790531142197386
120832336613378820991430378869014127 and
2204557440995913372286125950717602197195022637415287838524947867913051
465973479416134708098578709098671857 respectively.
Given N_1, N_2, with the implicit information and 30 MSBs of q_2, we can factorize both of them efficiently. We use lattice of dimension 105 and the lattice reduction takes 12889.92 seconds.

In our experiments for Examples 6.15, 6.16, we use four polynomials f_0, f_1, f_2, f_3 that are available after lattice reduction. Let J be the ideal generated by $\{f, f_0, f_1, f_2, f_3\}$ and let the corresponding Gröbner Basis be G. We studied the first three elements of G and found that one of them is of the form $y - \frac{q_1}{P_1 - P_1'} z$. Note that $y_0 = q_1, z_0 = P_1 - P_1'$ is the root of this polynomial.

Example 6.16. In this experiment, we consider 640-bit primes p_1 and p_2
2670046755820111597125983499598458226482802916962725172846821109006065
0996111051814093451331628000557452579503698243975066836110558623837620
26236414416814754812160919136728523814508982308428241
and
4213654354378478098923678873165024061763485330880715693603141201754570
6833482787889436013438027059065208431251021416732001179886518042599444
22004067683253229586251375139983258948708099714177489.

Note that, p_1, p_2 share 531 many LSBs. Further, q_1, q_2 are 360-bit primes
2286199623814759322661521537688479233374714361474052906990385379626075
664185395324016017013427591508941472939 and
1280130913887850481545063885171390143545984280360797978692995205434971
745123201014643124723774455018327835627 respectively.
Given N_1, N_2, with the implicit information and 30 MSBs of q_2, we can factorize both of them efficiently. We use lattice of dimension 105 and the lattice reduction takes 11107.04 seconds.

In a similar direction, we can extend Theorem 6.10 below, where we consider a few MSBs of q_2 are known.

Theorem 6.17. *Let $N_1 = p_1 q_1$ and $N_2 = p_2 q_2$, where p_1, q_1, p_2, q_2 are primes. Let $q_1, q_2 \approx N^{\alpha}$. Suppose q_{20} is known such that $|q_2 - q_{20}| \leq N^{\delta}$. Consider that a contiguous portion of bits of p_1, p_2 are same at the middle leaving the $\gamma_1 \log_2 N$ many MSBs and $\gamma_2 \log_2 N$ many LSBs. Then under Assumption 1, we can factor both N_1, N_2 if there exist $\tau_1, \tau_2 \geq 0$ for which $h(\tau_1, \tau_2, \alpha, \gamma_1, \gamma_2) < 0$ where $\gamma =$*

$\max\{\gamma_1, \gamma_2\}$ *and*

$$\begin{aligned} h(\tau_1, \tau_2, \alpha, \delta, \gamma_1, \gamma_2) = & \left(\frac{3}{2}\tau_1\tau_2 + \tau_1 + \tau_2 + \frac{1}{4}\right)\delta \\ & + \left(\frac{3}{2}\tau_1\tau_2 + \frac{4}{3}\tau_1 + \frac{4}{3}\tau_2 + \frac{11}{24}\right)\alpha \\ & + \left(\tau_1^2\tau_2 + \frac{3}{2}\tau_1\tau_2 + \frac{3}{4}\tau_1^2 + \frac{2}{3}\tau_1 + \frac{2}{3}\tau_2 + \frac{1}{6}\right)\gamma_1 \\ & + \left(\tau_1\tau_2^2 + \frac{3}{2}\tau_1\tau_2 + \frac{3}{4}\tau_2^2 + \frac{2}{3}\tau_1 + \frac{2}{3}\tau_2 + \frac{1}{6}\right)\gamma_2 \\ & - \left(\tau_1\tau_2 + \frac{\tau_1}{2} + \frac{\tau_2}{2} + \frac{1}{8}\right)(1+\gamma) < 0. \end{aligned}$$

Our idea does not work well when one considers that p_i, q_i are of same bitsize. The bound presented in Theorem 6.1 does not provide encouraging results as q_i's increase towards the value of p_i's. Considering some MSBs of q_2 are available, one can improve it a little bit as explained in Theorem 6.13. Even if we consider that some information regarding both q_1, q_2 are available, that does not help much. This is because under such scenario, the structure of the polynomial f in Theorem 6.1 changes and more monomials arrive. This prevents achieving a good bound.

6.4 Conclusion

In this chapter we have studied poly $(\log \mathrm{N})$ time factorization strategy when two integers N_1, N_2 (of same size) are given where $N_1 = p_1q_1, N_2 = p_2q_2$ and p_1, p_2 share certain amount of LSBs and/or MSBs taken together. We also study the same problem with three integers N_1, N_2, N_3. Further, we study the case with two integers N_1, N_2 when p_1, p_2 share some bits at the middle.

Our results extend the idea presented in [86]. For the LSB case, we obtain better results than [86] under certain conditions. We also consider an instance of the implicit factorization problem which has not been solved earlier.

However, the techniques presented here cannot immediately be extended to the generalized problem in [86] where $N_1, N_2, \ldots, N_k$ are considered. In the next chapter we will solve this problem. Chapter 7 concentrates on the approximate common divisor problem first, that paves the path to the generalization of implicit factorization.

Chapter 7

Approximate Integer Common Divisor Problem

Given any two large integers a, b (without loss of generality, take $a > b$), one can calculate $\gcd(a, b)$ efficiently in $O(\log^2 a)$ time using the well known Euclidean Algorithm [126, Page 169]. Howgrave-Graham [61] has shown that it is also possible to calculate the GCD efficiently when some approximations of a, b are available. This problem is referred to as the *approximate common divisor problem* in the literature. As an one important application of this problem, Howgrave-Graham [61] had shown that Okamoto's cryptosystem [98] is not secure. Using the idea of [61], Coron and May [29] proved deterministic polynomial time equivalence of computing the RSA secret key and factoring the RSA modulus. In this chapter, we first present two applications of *approximate common divisor problem.*

Application 1: For the first application, consider $N = pq$, where p, q are large primes and $p > q$. In a recent paper [50] presented at Crypto 2009, it has been asked how one can use $q^{-1} \bmod p$ towards factorization of N as $q^{-1} \bmod p$ is stored as a part of the secret key in PKCS #1 [99]. Using lattice based technique, we show that factoring N is deterministic polynomial time equivalent to finding $q^{-1} \bmod p$.

Application 2: Next, we consider the problem of finding smooth integers in a small interval [12, 13]. Finding smooth numbers is important for application in the well known factorization algorithms such as quadratic sieve [100] and number field sieve [76]. We study the results of [12,13] and show that slightly improved outcome could be achieved using a different strategy following the idea of [61].

Later in this chapter (Section 7.3 onwards), we focus on a couple of general extensions of the approximate common divisor problem, and relate it to *implicit factorization*, another well known problem along similar lines.

7.1 Finding $q^{-1} \bmod p \equiv$ Factorization of N

In this section, we prove the computational equivalence of finding $q^{-1} \bmod p$ and factoring $N = pq$. In this direction, we present the following result.

Theorem 7.1. *Assume $N = pq$, where p, q are primes and $p \approx N^{\gamma}$. Suppose an approximation p_0 of p is known such that $|p - p_0| < N^{\beta}$. Given $q^{-1} \bmod p$, one can factor N deterministically in* $\mathrm{poly}(\log \mathrm{N})$ *time when $\beta - 2\gamma^2 < 0$.*

Proof. Let $q_1 = q^{-1} \bmod p$. So we can write $qq_1 = 1 + k_1 p$ for some positive integer k_1. Multiplying both sides by p, we get $q_1 N = p + k_1 p^2$. That is, we have $q_1 N - p = k_1 p^2$. Let $x_0 = p - p_0$. Thus, we have $q_1 N - p_0 - x_0 = k_1 p^2$. Also we have $N^2 = p^2 q^2$. Our goal is to recover x_0 from $q_1 N - p_0$ and N^2.

Note that p^2 is the GCD of $q_1 N - p_0 - x_0$ and N^2. In this case $q_1 N - p_0$ and N^2 is known, i.e., one term N^2 is exactly known and the other term $q_1 N - p_0 - x_0$ is approximately known. This is exactly the Partially Approximate Common Divisor Problem (PACDP) [61] and we follow a similar technique to solve this as explained below. This will provide the error term $-x_0$, which added to the approximation $q_1 N - p_0$, gives the exact term $q_1 N - p_0 - x_0$.

Take $X = N^{\beta}$ as an upper bound of x_0. Then we consider the shift polynomials

$$\begin{aligned} g_i(x) &= (q_1 N - p_0 + x)^i N^{2(m-i)} \text{ for } 0 \le i \le m, \\ g'_i(x) &= x^i (q_1 N - p_0 + x)^m \text{ for } 1 \le i \le t, \end{aligned} \tag{7.1}$$

for fixed non-negative integers m, t. Clearly, $g_i(-x_0) \equiv g'_i(-x_0) \equiv 0 \pmod{p^{2m}}$.

We construct the lattice L spanned by the coefficient vectors of the polynomials $g_i(xX), g'_i(xX)$. One can check that the dimension of the lattice L is $\omega = m + t + 1$ and the determinant of L is

$$det(L) = X^{\frac{(m+t)(m+t+1)}{2}} N^{2\frac{m(m+1)}{2}} = X^{\frac{(m+t)(m+t+1)}{2}} N^{m(m+1)}. \tag{7.2}$$

Using Lattice reduction on L by LLL algorithm [77], one can find a nonzero vector

b whose norm $||\mathbf{b}||$ satisfies $||\mathbf{b}|| \leq 2^{\frac{\omega-1}{4}}(\det(L))^{\frac{1}{\omega}}$. The vector **b** is the coefficient vector of the polynomial $h(xX)$ with $||h(xX)|| = ||\mathbf{b}||$, where $h(x)$ is the integer linear combination of the polynomials $g_{ij}(x)$. Hence $h(-x_0) \equiv 0 \pmod{p^{2m}}$. To apply Lemma 2.20 and Lemma 2.22 for finding the integer root of $h(x)$, we need

$$2^{\frac{\omega-1}{4}}(det(L))^{\frac{1}{\omega}} < \frac{p^{2m}}{\sqrt{\omega}}. \tag{7.3}$$

Note that ω is the dimension of the lattice which we may consider as small constant with respect to the size of p and the elements of L. Thus, neglecting $2^{\frac{\omega-1}{4}}$ and $\sqrt{\omega}$, we can rewrite (7.3) as $det(L) < p^{2m\omega}$. Substituting the expression of $det(L)$ from Equation (7.2) and using $X = N^{\beta}, p \approx N^{\gamma}$ we get

$$\frac{(m+t)(m+t+1)}{2}\beta + m(m+1) < 2m(m+t+1)\gamma. \tag{7.4}$$

Let $t = \tau m$. Then neglecting the terms of $o(m^2)$ we can rewrite (7.4) as

$$\frac{\tau^2\beta}{2} + (\beta - 2\gamma)\tau + \frac{\beta}{2} - 2\gamma + 1 < 0. \tag{7.5}$$

Now, the optimal value of τ to minimize the left hand side of (7.5) is $\frac{2\gamma-\beta}{\beta}$. Putting this optimal value in (7.5), we get $\beta - 2\gamma^2 < 0$.

Our strategy uses LLL algorithm [77] to find $h(x)$ and then calculates the integer root of $h(x)$. Both these steps are deterministic polynomial time in $\log N$. Thus the result. ■

Corollary 7.2. *Factoring N is deterministic polynomial time equivalent to finding $q^{-1} \bmod p$, where $N = pq$ and $p > q$.*

Proof. When no approximation of p is given, then β in the Theorem 7.1 is equal to γ. Putting $\beta = \gamma$ in the condition $\beta - 2\gamma^2 < 0$, we get $\gamma > \frac{1}{2}$. This requirement forces the condition that $p > q$. Also, it is trivial to note that if the factorization of N is known then one can efficiently compute $q^{-1} \bmod p$. Thus the proof. ■

Corollary 7.3. *Factoring N is deterministic polynomial time equivalent to finding $q^{-1} \bmod p$, where $N = pq$ and p, q are of same bit size.*

Proof. The proof of the case $p > q$ is already taken care in Corollary 7.2. Now consider $q > p$. When p, q are of same bit size and $p < q$, then $p < q < 2p$, i.e.,

$\sqrt{\frac{N}{2}} < p < \sqrt{N}$ and $\sqrt{N} < q < \sqrt{2N}$.

Now if we take $p_0 = \sqrt{N}$ then $|p-p_0| < \left(1 - \frac{1}{\sqrt{2}}\right)\sqrt{N} < \frac{N^{\frac{1}{2}}}{2} = N^{\frac{1}{2}-\frac{\log 2}{\log N}}$. Also $p > N^{\frac{1}{2}-\frac{\log 2}{2\log N}}$. So in this case we can take $\beta = \frac{1}{2} - \frac{\log 2}{\log N}$ and $\gamma > \frac{1}{2} - \frac{\log 2}{2\log N}$. Thus,

$$\beta - 2\gamma^2 < \frac{1}{2} - \frac{\log 2}{\log N} - 2\left(\frac{1}{2} - \frac{\log 2}{2\log N}\right)^2 = -\frac{\log^2 2}{2\log^2 N} < 0.$$

Hence in this situation one can factor N following Theorem 7.1. ■

The only case that we do not cover is when p is significantly smaller than q. It requires further study to understand how this situation can be tackled.

Now let us describe the experimental results. Note that our result in Theorem 7.1 holds when the lattice dimension approaches infinity. Since in practice we use finite lattice dimension, we may not reach the bound presented in Theorem 7.1. For experiments, we consider that small amount of Most Significant Bits (MSBs) of p is known. In Table 7.1, we provide some practical results. In the first three experiments, we take N as 1000-bit integer with p, q of the same bit size and $p > q$. Then in the next three experiments, we swapped p, q, i.e., q becomes larger than p. Given $q^{-1} \bmod p$, we could successfully recover p in all the cases.

p ? q	# MSBs of p known	Lattice Parameters (m, t)	Lattice Dim.	Time (in sec.)
$p > q$	46	(5, 5)	11	1.41
$p > q$	24	(10, 10)	21	66.33
$p > q$	20	(11, 11)	23	119.72
$q > p$	47	(5, 5)	11	1.42
$q > p$	24	(10, 10)	21	66.56
$q > p$	20	(11, 11)	23	120.00

Table 7.1: Experimental results following Theorem 7.1. The lattice parameters are (m, t).

7.2 Finding Smooth Integers in a Short Interval

Following [12, 13], let us first define two notions of smoothness.

Definition 7.4. An integer N is called B *smooth* if N has no prime divisor greater than B. An integer N is called *strongly* B *smooth* if N is B smooth and p^m can not divide N for any m for which $p^m > B$.

Let us denote the n-th prime by p_n, e.g., $p_1 = 2, p_2 = 3$ and so on. Suppose we want to find a strongly B smooth integer N in the interval $[U, V]$. Now let us present our main result of this section.

Theorem 7.5. *Let $S = \prod_{i=1}^{n} p_i^{a_i}$ where $a_i = \lfloor \frac{\log B}{\log p_i} \rfloor$ and $p_1, \ldots, p_n$ are all distinct primes not exceeding B. Let $I = [U, V]$. One can find all strongly B smooth integers $N \in I$ for which $\gcd(N, S) > d$ in $\mathrm{poly}(\log \mathrm{S})$ time when $|I| < 2d^{\frac{\log d}{\log S}}$ and $V < 2d$.*

Proof. We will try to find N such that $\gcd(N, S) > d$. Let us take take $a_0 = \lfloor \frac{U+V}{2} \rfloor$. We consider a_0 as an approximation of N. Thus we will try to find the GCD of S, N, by knowing S exactly and some approximation of N, which is a_0 (but N is not known). Here we follow the idea of solving the Partially Approximate Common Divisor Problem (PACDP) as explained in [61].

Let $x_0 = N - a_0$. We want to calculate x_0 from a_0, S. Assume $X = d^{\beta}$ is an upper bound of x_0. Let $S = d^{\delta}$. Using the same approach as in the proof of Theorem 7.10, we get the condition as

$$\frac{(m+t)(m+t+1)}{2}\beta + \frac{m(m+1)}{2}\delta < m(m+t+1). \tag{7.6}$$

Let $t = \tau m$. Then neglecting the terms of $o(m^2)$ we can rewrite (7.6) as

$$\frac{\beta}{2}\tau^2 + (\beta - 1)\tau + \frac{\beta}{2} + \frac{\delta}{2} - 1 < 0. \tag{7.7}$$

Now, the optimal value of τ to minimize the left hand side of (7.7) is $\frac{1-\beta}{\beta}$. Putting this optimal value in (7.7), we get $\beta < \frac{1}{\delta}$. Now $\delta = \frac{\log S}{\log d}$. So x_0 should be less than $d^{\frac{\log d}{\log S}}$.

Thus, we get x_0 and hence N in $\mathrm{poly}(\log \mathrm{S})$ time. As, $V < 2d$, we have $N < 2d$ (since $U \leq N \leq V$). When $\gcd(N, S) > d$, then $\gcd(N, S) = N$ as $N < 2d$. Hence N divides S, i.e., N is strongly B smooth.

Our strategy exploits the solution of Partially Approximate Common Divisor Problem (PACDP) presented in [61]. That, all such strongly B smooth numbers will be available, follows from [61, Algorithm 12, Page 53] as in that case all the common divisors are reported. ∎

Asymptotically, our result is 8 times better than that of [12, Theorem 3.1], as that bound was $|I| < \frac{1}{4} d^{\frac{\log d}{\log S}}$. In Table 7.2, we present a few experimental results, where we find improved outcomes (in terms of execution time) using our strategy than that of [12]. We have implemented the ideas of [12] for experimental comparison. In the table, LD denotes Lattice Dimension, and the time denotes the Lattice reduction time. One should also note that the method of [12] requires the implementation of CRT on several primes, which is not included in the time mentioned in Table 7.2. Our strategy, using the idea of [61], does not require such computation.

B	$\log_2 d$	$\log_2(V-U)$	LD (Our), Time (sec.)	LD ([12]), Time (sec.)
1000	450	130	36, 15.51	32, 21.33
1000	496	156	29, 3.77	26, 8.06
1000	496	161	45, 36.88	41, 64.71

Table 7.2: Comparison of our experimental results with that of [12]. We have implemented the ideas of [12] for the comparison. LD denotes lattice dimension.

Note that finding smooth integers in an interval was also discussed in [84] considering a similar idea of lattice construction. However, we arrive at the solution following the PACDP.

7.3 Extension of Approximate Common Divisor Problem

Let us now describe the generalized version of PACDP [61]. We name this version the Extended Partially Approximate Common Divisor Problem (EPACDP).

Problem Statement 1. EPACDP.
Let $a, a_1, a_2, \ldots, a_k$ be large integers (of same bitsize) and $g = \gcd(a_1, a_2, \ldots, a_k)$, for $k \geq 2$. Consider that $\tilde{a}_2, \ldots, \tilde{a}_k$ are the approximations of $a_2, \ldots, a_k$ respectively, and $\tilde{a}_2, \ldots, \tilde{a}_k$ are of same bitsize too. Suppose that $a_2 = \tilde{a}_2 + \tilde{x}_2, \ldots, a_k = \tilde{a}_k + \tilde{x}_k$. The goal is to find $\tilde{x}_2, \ldots, \tilde{x}_k$ from the knowledge of $a_1, \tilde{a}_2, \ldots, \tilde{a}_k$.

The integer a is referred to in EPACDP as the representative element of the same bit size as $a_1, a_2, \ldots, a_k$. This 'a' will be used frequently throughout this chapter, mainly in calculating the time complexities. Similarly, we also consider N to be the representative integer of the same bitsize as $N_1, N_2, \ldots, N_k$.

One may observe that another generalization of PACDP is possible. Consider the case with two integers a_1, a_2 where neither is available exactly, but approximations of both a_1, a_2 are given. In such a case, the problem of finding $\gcd(a_1, a_2)$ is referred to as the General Approximate Common Divisor Problem (GACDP), as introduced in [61]. One can extend this notion for more than two integers, as follows. We call this generalization the Extended General Approximate Common Divisor Problem (EGACDP).

Problem Statement 2. EGACDP.
Let $a, a_1, a_2, \ldots, a_k$ be large integers of same bitsize and $g = \gcd(a_1, a_2, \ldots, a_k)$, for $k \geq 2$. Consider that $\tilde{a}_1, \ldots, \tilde{a}_k$ are the approximations of $a_1, \ldots, a_k$ respectively, and $\tilde{a}_1, \ldots, \tilde{a}_k$ are of same bitsize too. Suppose that $a_1 = \tilde{a}_1 + \tilde{x}_1, a_2 = \tilde{a}_2 + \tilde{x}_2, \ldots, a_k = \tilde{a}_k + \tilde{x}_k$. The goal is to find $\tilde{x}_1, \tilde{x}_2, \ldots, \tilde{x}_k$ from the knowledge of $\tilde{a}_1, \tilde{a}_2, \ldots, \tilde{a}_k$.

Now, our motivation is to link the two well known problems along similar lines; the approximate integer common divisor problem, and the implicit factorization problem. We have already discussed the former. Let us define the later problem before starting our discussion about the interconnection of the two.

Problem Statement 3. Implicit Factorization Problem.
Consider $N_1 = p_1q_1, N_2 = p_2q_2, \ldots, N_k = p_kq_k$, where $p_1, p_2, \ldots, p_k$ and $q_1, q_2, \ldots, q_k$ are primes. Also consider that $p_1, p_2, \ldots, p_k$ are of same bitsize and so are $q_1, q_2, \ldots, q_k$ (this is followed throughout this chapter unless otherwise mentioned). Given that certain portions of bit pattern in $p_1, p_2, \ldots, p_k$ are common, the goal is to find under what condition it is possible to factor $N_1, N_2, \ldots, N_k$ efficiently.

The results on implicit factorization, published in [86], were based on the assumption that some amount of least significant bits (LSBs) are same in $p_1, p_2, \ldots, p_k$. In Chapter 6, we considered different cases, namely (i) some portions of LSBs are same and/or some portions of MSBs are same in $p_1, p_2, \ldots, p_k$, and (ii) some portion of bits at the middle are same in $p_1, p_2, \ldots, p_k$. However, our techniques can be applied only for $k = 2, 3$. Recently, the idea published in [40] considered the case when some portions of MSBs are same in $p_1, p_2, \ldots, p_k$ for general k.

We concentrate on the implicit factorization problem considering that the primes $p_1, p_2, \ldots, p_k$ share (i) some amount of MSBs, or (ii) some amount of LSBs, or (iii) some amount of MSBs and LSBs together. Because of this fact, the work of [86] (the LSB case) as well as that of [40] (the work for MSB case) are considered in the same framework. Further, the proposed technique takes care of the new case where the primes share some portion of MSBs and LSBs together. This work is of the same quality (in general) and slightly improved (in certain cases) in comparison to that of [40,86]. We generalize the ideas of [61] for the lattice based technique that we exploit here, and our strategy is different from that of [40,86] and our ideas in Chapter 6.

Let us first describe the central idea of the link between EPACDP and implicit factorization on a small scale, and later we shall proceed to generalize the same. Consider the case with $k = 2$, where the primes p_1, p_2 share certain amount of MSBs. One can write $p_1 - p_2 = x_0$, where the bitsize of x_0 is smaller than that of p_1 or p_2. In terms of x_0, one may write $N_2 = p_2 q_2 = (p_1 - x_0) q_2$. Therefore, we have $\gcd(N_1, N_2 + x_0 q_2) = \gcd(p_1 q_1, p_1 q_2) = p_1$. Since N_2 is a known approximation of the unknown quantity $N_2 + x_0 q_2$, we can use the technique of [61] to solve the approximate common divisor problem efficiently with $a = N_1$ (known) and $b = N_2 + x_0 q_2$ (unknown), and get $\gcd(a, b) = p_1$ under certain conditions. It is very interesting that solving an approximate common divisor problem in this case gives the factorization of N_1. Additionally, when $p_1 > |x_0 q_2|$, then either $\lfloor \frac{N_2}{p_1} \rfloor$ or $\lceil \frac{N_2}{p_1} \rceil$ will provide q_2, thereby factorizing N_2 as well. In Section 7.4.1, we explain this idea in detail.

Next we generalize the PACDP given in [61]. Let $a_1, a_2, \ldots, a_k$ are integers with $\gcd(a_1, a_2, \ldots, a_k) = g$. Suppose $\tilde{a}_2, \ldots, \tilde{a}_k$ are given as approximations to $a_2, \ldots, a_k$, respectively. The goal of the generalized version of PACDP is to find g from the knowledge of $a_1, \tilde{a}_2, \ldots, \tilde{a}_k$. An immediate application of this generalization towards the implicit factorization problem is as follows.

We can write $p_1 = p_1 + y_1, \ldots, p_k = p_1 + y_k$ where $y_1 = 0$. Hence $p_1 = \gcd(N_1, N_2 - y_2 q_2, \ldots, N_k - y_k q_k)$ can be derived by solving the general PACDP with $a_1 = N_1, a_2 = N_2 - y_2 q_2, \ldots, a_k = N_k - y_k q_k$, where $N_2, \ldots, N_k$ act as the approximations of $a_2, \ldots, a_k$ respectively. As a consequence, we factor N_1, and also obtain $y_2 q_2, \ldots, y_k q_k$ under certain conditions.

In the case of implicit factorization problem, we will always get N_1 exactly, and the other terms $N_2 - y_2 q_2, \ldots, N_k - y_k q_k$ can be approximated by $N_2, \ldots, N_k$ respectively. Thus implicit factorization relates directly to EPACDP (and not to EGACDP) for any $k \geq 2$.

7.4 The General Solution for EPACDP

Towards solving the Extended Partially Approximate Common Divisor Problem (EPACDP), consider the polynomials

$$\begin{aligned} h_2(x_2,\ldots,x_k) &= \tilde{a}_2 + x_2, \\ &\vdots \\ h_k(x_2,\ldots,x_k) &= \tilde{a}_k + x_k, \end{aligned} \tag{7.8}$$

where $x_2,\ldots,x_k$ are the variables. Clearly, g (as in Problem Statement 1) divides $h_i(\tilde{x}_2,\ldots,\tilde{x}_k)$ for $2 \le i \le k$. Now let us define the shift polynomials

$$H_{\underbrace{j_2,\ldots,j_k}_{(k-1)\ many}}(x_2,\ldots,x_k) = h_2^{j_2}\cdots h_k^{j_k} a_1^{m-j_2-\cdots-j_k} \tag{7.9}$$

for non-negative integers $j_2,\ldots,j_k$, such that $j_2+\cdots+j_k \le m$, where the integer $m \ge 0$ is fixed. Further, we define another set of shift polynomials

$$H'_{\underbrace{0,\ldots,i_n,\ldots,0}_{(k-1)\ many},\underbrace{j_2,\ldots,j_k}_{(k-1)\ many}}(x_2,\ldots,x_k) = x_n^{i_n} h_2^{j_2}\cdots h_k^{j_k} \tag{7.10}$$

with the following:

1. $1 \le i_n \le t$, for $2 \le n \le k$ and a positive integer t, and
2. $j_2+\cdots+j_k = m$ when $0 \le j_2,\ldots,j_{n-1} < i_n$, with $0 \le j_n,\ldots,j_k \le m$.

Note that g^m divides $H_{j_2,\ldots,j_k}(\tilde{x}_2,\ldots,\tilde{x}_k)$, and $H'_{0,\ldots,i_n,\ldots,0,j_2,\ldots,j_k}(\tilde{x}_2,\ldots,\tilde{x}_k)$. Let $X_2,\ldots,X_k$ be the upper bounds of $\tilde{x}_2,\ldots,\tilde{x}_k$ respectively. Now define a lattice L using the coefficient vectors of

$$\begin{aligned} &H_{j_2,\ldots,j_k}(x_2X_2,\ldots,x_kX_k), \quad \text{and} \\ &H'_{0,\ldots,i_n,\ldots,0,j_2,\ldots,j_k}(x_2X_2,\ldots,x_kX_k). \end{aligned}$$

Let the dimension of L be ω. Under Assumption 1 of Chapter 2, one gets $\tilde{x}_2,\ldots,\tilde{x}_k$ using lattice reduction over L if

$$2^{\frac{\omega(\omega-1)}{4(\omega+2-k)}} \det(L)^{\frac{1}{\omega+2-k}} < \frac{g^m}{\sqrt{\omega}}.$$

The result follows from Theorem 2.23 and putting $i = k - 1$ in Lemma 2.20. Neglecting the small constants and considering $k \ll \omega$ (in fact, we will show that ω is exponential in k in our construction), we get the condition as $\det(L)^{\frac{1}{\omega}} < g^m$, i.e., $\det(L) < g^{m\omega}$. This is written formally in Theorem 7.8 later.

Before proceeding to the next discussion, we denote that $\binom{n}{r}$ is considered in its usual meaning when $n \geq r \geq 0$, and in all other cases we will consider the value of $\binom{n}{r}$ as 0.

Lemma 7.6. *Let ω be the dimension of the lattice L described as before. Then*

$$\omega = \sum_{r=0}^{m} \binom{k+r-2}{r} + \sum_{n=2}^{k} \sum_{i_n=1}^{t} \sum_{r=0}^{n-2} (-1)^r \binom{n-2}{r} \binom{k+m-ri_n-2}{m-ri_n}.$$

Proof. Let $j_2 + \cdots + j_k = r$ where $j_2, \ldots, j_k$ are non-negative integers. The number of such solutions is $\binom{k+r-2}{r}$. Hence the number of shift polynomials in Equation (7.9) is

$$\omega_1 = \sum_{r=0}^{m} \binom{k+r-2}{r}.$$

For fixed n, i_n, the number of shift polynomials in Equation (7.10) is the number of solutions of

$$j_2 + \cdots + j_k = m$$

for $0 \leq j_2, \ldots, j_{n-1} < i_n$, and $0 \leq j_n, \ldots, j_k \leq m$. The number of all such solutions is the coefficient of x^m in

$$\begin{aligned} & \left(1 + x + \cdots + x^{i_n - 1}\right)^{n-2} \left(1 + x + \cdots + x^m\right)^{k-n+1} \\ = & \left(\frac{1 - x^{i_n}}{1 - x}\right)^{n-2} \left(\frac{1 - x^{m+1}}{1 - x}\right)^{k-n+1} \\ = & \left(1 - x^{i_n}\right)^{n-2} \left(1 - x^{m+1}\right)^{k-n+1} (1 - x)^{-k+1}. \end{aligned}$$

We denote the coefficient by $c(n, i_n)$, for fixed n, i_n, which can be written as

$$c(n, i_n) = \sum_{r=0}^{n-2} (-1)^r \binom{n-2}{r} \binom{k+m-ri_n-2}{m-ri_n}.$$

Hence the number of shift polynomials in Equation (7.10) is

$$\omega_2 = \sum_{n=2}^{k} \sum_{i_n=1}^{t} c(n, i_n).$$

Finally, $\omega = \omega_1 + \omega_2$ provides the result. ■

After dealing with the dimension of lattice L, let us evaluate its determinant, as follows.

Lemma 7.7. *The determinant of L is* $\det(L) = P_1 P_2$ *where*

$$P_1 = \prod X_2^{j_2} X_3^{j_3} \cdots X_k^{j_k} a_1^{m-j_2-\cdots-j_k}$$

for non-negative integers $j_2, \ldots, j_k$ such that $j_2 + \cdots + j_k \leq m$, and

$$P_2 = \prod X_n^{i_n} X_2^{j_2} X_3^{j_3} \cdots X_k^{j_k}$$

with the following:

1. *$1 \leq i_n \leq t$, for $2 \leq n \leq k$ and a positive integer t, and*

2. *$j_2 + j_3 + \cdots + j_k = m$, when $0 \leq j_2, \ldots, j_{n-1} < i_n$, and $0 \leq j_n, \ldots, j_k \leq m$.*

Proof. The matrix (corresponding to the lattice L) containing the basis vectors is triangular and has the following two kinds of diagonal entries:

$$X_2^{j_2} X_3^{j_3} \cdots X_k^{j_k} a_1^{m-j_2-\cdots-j_k}, \tag{7.11}$$

for non-negative integers $j_2, \ldots, j_k$ such that $j_2 + \cdots + j_k \leq m$ where the integer $m \geq 0$ fixed and

$$X_n^{i_n} X_2^{j_2} X_3^{j_3} \cdots X_k^{j_k}, \tag{7.12}$$

with the following:

1. $1 \leq i_n \leq t$, for $2 \leq n \leq k$ and a positive integer t, and

2. $j_2 + j_3 + \cdots + j_k = m$, when $0 \leq j_2, \ldots, j_{n-1} < i_n$, and $0 \leq j_n, \ldots, j_k \leq m$.

Clearly P_1 is the product of the elements from Equation (7.11) and P_2 is the product of the elements from Equation (7.12). Hence, we have $\det(L) = P_1 P_2$. ■

Runtime. The running time of our algorithm is dominated by the runtime of the LLL algorithm, which is polynomial in the dimension of the lattice and in the bitsize of the entries. Since the lattice dimension in our case is exponential in k, the running time of our strategy is $\mathrm{poly}\{\log \mathrm{a}, \exp(\mathrm{k})\}$. Thus, for small fixed k our algorithm is polynomial in $\log a$.

Now, we can present the main result of this section, as follows.

Theorem 7.8. *Under Assumption 1, the EPACDP (Problem Statement 1) can be solved in* $\mathrm{poly}\{\log \mathrm{a}, \exp(\mathrm{k})\}$ *time when* $\det(L) < g^{m\omega}$*, where* $\det(L)$ *is as in Lemma 7.7 and* ω *is as in Lemma 7.6.*

One may also consider the same upper bound on the errors $\tilde{x}_2, \ldots, \tilde{x}_k$. In that case we get the following result.

Corollary 7.9. *Considering the same upper bound* X *on the errors* $\tilde{x}_2, \ldots, \tilde{x}_k$*, we have* $\det(L) = P_1 P_2$ *where*

$$P_1 = X^{\eta_1} a_1^{\eta_2} \quad \textit{and} \quad P_2 = X^{\eta_3}$$

and the exponents are

$$\begin{aligned}
\eta_1 &= \sum_{r=0}^{m} r \cdot \binom{k+r-2}{r}, \\
\eta_2 &= \sum_{r=0}^{m} (m-r) \cdot \binom{k+r-2}{r}, \quad \textit{and} \\
\eta_3 &= \sum_{n=2}^{k} \sum_{i_n=1}^{t} (i_n + m) \sum_{r=0}^{n-2} (-1)^r \binom{n-2}{r} \binom{k+m-ri_n-2}{m-ri_n}.
\end{aligned}$$

Proof. Let $X_2 = X_3 = \cdots = X_k = X$. Then from Equation (7.11), we have

$$X_2^{j_2} X_3^{j_3} \cdots X_k^{j_k} a_1^{m-j_2-\cdots-j_k} = X^{j_2+\cdots+j_k} a_1^{m-j_2-\cdots-j_k}$$

for non-negative integers $j_2, \ldots, j_k$ such that $j_2 + \cdots + j_k \leq m$. Let $j_2 + \cdots + j_k = r$ where $0 \leq r \leq m$. The total number of such representations is $\binom{k+r-2}{r}$. Hence

$$P_1 = \prod_{r=0}^{m} (X^r a_1^{m-r})^{\binom{k+r-2}{r}} = X^{\eta_1} a_1^{\eta_2}$$

where η_1, η_2 are as mentioned in the statement.

For calculating P_2, we have the following constraints:

1. $1 \leq i_n \leq t$, for $2 \leq n \leq k$ and a positive integer t, and
2. $j_2 + j_3 + \cdots + j_k = m$, when $0 \leq j_2, \ldots, j_{n-1} < i_n$, and $0 \leq j_n, \ldots, j_k \leq m$.

Thus we have the expression for P_2 as

$$\begin{aligned} P_2 &= \prod_{n=2}^{k} \prod_{i_n=1}^{t} X_n^{i_n} X_2^{j_2} X_3^{j_3} \cdots X_k^{j_k} \\ &= \prod_{n=2}^{k} \prod_{i_n=1}^{t} X^{i_n c(n,i_n)} X^{mc(n,i_n)} = X^{\eta_3} \end{aligned}$$

where η_3 is as mentioned in the statement. ∎

As the results in this section are quite involved, we present below a few cases for better understanding and comparison with existing results.

7.4.1 Analysis for $k = 2$

We write the proof of this special case in detail as this is in line with the proof of [29, Theorem 3] where the strategy to solve the Partially Approximate Common Divisor Problem (PACDP) [61] has been exploited.

As described in [24], after applying the LLL algorithm, if the output polynomials are of more than one variable, then to collect the roots from these polynomials one needs Assumption 1. However, in this case, Assumption 1 is not required since there is only one variable in the polynomial that we will consider.

Theorem 7.10. *Let $N_1 = p_1q_1$ and $N_2 = p_2q_2$, where p_1, p_2, q_1, q_2 are primes. Let $q_1, q_2 \approx N^\alpha$ and $|p_1 - p_2| < N^\beta$. Then one can factor N_1 and N_2 deterministically in* poly(log N) *time when*

$$\beta < 1 - 3\alpha + \alpha^2.$$

Proof. Let $x_0 = p_1 - p_2$. We have $N_1 = p_1q_1$ and $N_2 = p_2q_2 = (p_1 - x_0)q_2$. Our goal is to recover x_0q_2 from N_1 and N_2. Since $|x_0| < N^\beta$ and $q_2 = N^\alpha$, we can take

$X = N^{\alpha+\beta}$ as an upper bound of x_0q_2. Now we consider the shift polynomials

$$\begin{aligned} g_i(x) &= (N_2 + x)^i N_1^{m-i} \text{ for } 0 \le i \le m, \\ g'_i(x) &= x^i (N_2 + x)^m \text{ for } 1 \le i \le t, \end{aligned} \tag{7.13}$$

where m, t are fixed non-negative integers. Clearly,

$$g_i(x_0q_2) \equiv g'_i(x_0q_2) \equiv 0 \bmod (p_1^m).$$

We construct the lattice L spanned by the coefficient vectors of the polynomials $g_i(xX), g'_i(xX)$ in Equation (7.13). One can check that the dimension of the lattice L is $\omega = m + t + 1$ and the determinant of L is

$$\begin{aligned} \det(L) &= X^{\frac{(m+t)(m+t+1)}{2}} N_1^{\frac{m(m+1)}{2}} \\ &\approx X^{\frac{(m+t)(m+t+1)}{2}} N^{\frac{m(m+1)}{2}}. \end{aligned} \tag{7.14}$$

Here, $P_1 = X^{\frac{m(m+1)}{2}} N_1^{\frac{m(m+1)}{2}}$ and $P_2 = X^{mt+\frac{t(t+1)}{2}}$ (the general expressions of P_1, P_2 are presented in Lemma 7.7). Using Lattice reduction on L by the LLL algorithm [77], one can find a nonzero vector b whose norm $||b||$ satisfies

$$||b|| \le 2^{\frac{\omega-1}{4}} (\det(L))^{\frac{1}{\omega}}.$$

The vector b is the coefficient vector of the polynomial $h(xX)$ with $||h(xX)|| = ||b||$, where $h(x)$ is the integer linear combination of the polynomials $g_i(x), g'_i(x)$. Hence $h(x_0q_2) \equiv 0 \bmod (p_1^m)$. To apply Theorem 2.23 and Lemma 2.20 for finding the integer root of $h(x)$, we need

$$2^{\frac{\omega-1}{4}} (\det(L))^{\frac{1}{\omega}} < \frac{p_1^m}{\sqrt{\omega}}. \tag{7.15}$$

Neglecting small constant terms, we can rewrite (7.15) as $\det(L) < p_1^{m\omega}$. Substituting the expression of $\det(L)$ from Equation (7.14) and using $X = N^{\alpha+\beta}, p_1 \approx N^{1-\alpha}$ we get

$$\frac{(m+t)(m+t+1)}{2}(\alpha+\beta) < m\left((1-\alpha)(m+t+1) - \frac{m+1}{2}\right). \tag{7.16}$$

Now let $t = \tau m$. Neglecting the $o(m^2)$ terms of (7.16), we have

$$\psi(\alpha, \beta, \tau) = -(\alpha + \beta)\frac{\tau^2}{2} - (2\alpha + \beta - 1)\tau - \left(\frac{3\alpha}{2} + \frac{\beta}{2} - \frac{1}{2}\right) > 0. \tag{7.17}$$

The optimal value of τ, to maximize β for a fixed α is

$$\tau = \frac{1 - \beta - 2\alpha}{\alpha + \beta}.$$

Putting the optimal value of τ in (7.17), we get

$$\alpha^2 - 3\alpha - \beta + 1 > 0. \tag{7.18}$$

Once $x_0 q_2$, integer root of $h(x)$, is known, we get p_1 from $\gcd(N_1, N_2 + x_0 q_2)$.

As long as $|x_0 q_2| < p_1$, we get q_2 by calculating the floor or ceiling of $\frac{N_2}{p_1}$. As $|x_0 q_2| \leq N^{\alpha+\beta}$ and $p_1 \approx N^{1-\alpha}$, to satisfy $|x_0 q_2| \leq p_1$ we need $2\alpha + \beta \leq 1$, which is true from (7.18).

Our strategy uses the LLL [77] algorithm to find $h(x)$ and then calculates the integer root of $h(x)$. Both these steps are deterministic polynomial time in $\log N$. Thus the result. ∎

The relation presented in (7.16) provides the bound when the lattice parameters m, t are specified. The asymptotic relation, independent of the lattice parameters, has been presented in (7.18). This is the theoretical bound and may not be reached in practice as we work with low lattice dimensions. Now let us compare our results with that of Chapter 6.

1. In Corollary 6.4 of Chapter 6, it has been explained that factorization of N_1, N_2 will be successful when

$$\Psi(\alpha, \beta) = 4\alpha^2 + 2\alpha\beta + \frac{1}{4}\beta^2 - 4\alpha - \frac{5}{3}\beta + 1 > 0,$$

   provided that $1 - \frac{3}{2}\beta - 2\alpha \geq 0$. In our case, the upper bound of β is $\alpha^2 - 3\alpha + 1$. Putting this upper bound of β in Ψ we get $\alpha \leq 0.33 \Rightarrow \Psi(\alpha, \beta) < 0$. Hence our upper bound on β will be greater than that of Chapter 6 when $\alpha \leq 0.33$.

2. The result presented in Chapter 6 is a poly(log N) time heuristic (based on Assumption 1), whereas our algorithm in this chapter is deterministic

poly(log N) time.

3. The result of Chapter 6 could not be extended for $k > 3$, but our result can be extended for general k.

4. We get similar quality of experimental results as in Chapter 6 for $k = 2$ and both the experimental results (i.e., our and that of Chapter 6 for $k = 2$) almost coincide with our theoretical results. Our experimental results are same as our theoretical results following (7.16) for specific lattice dimensions, whereas the experimental results of Chapter 6 for $k = 2$ are better than their theoretical results, as explained in Remark 6.6 of Chapter 6.

We also compare our result with that of [40, 86].

- The strategy of [86] considers equality in some LSBs of p_1, p_2. One may note that our theoretical results for the MSB case is same as that of the LSB case (see Section 7.5.1 later). The strategy of [86] works when

$$\beta \leq 1 - 3\alpha.$$

  In our case, $\beta < 1 - 3\alpha + \alpha^2$. It is immediate that our upper bound is better than that of [86]. Given α, the amount of required bit sharing is $(1 - \alpha - \beta)\log_2 N$. Thus, for $k = 2$, we need smaller number of bit sharing in LSBs for implicit factorization than the number of bit sharing in LSBs achieved in [86].

- For $k = 2$, the strategy of [40] works in MSB case when

$$\beta \leq 1 - 3\alpha - \frac{3}{\log_2 N}.$$

  It is clear that our theoretical result is better than [40] for the MSB case. Later in Section 7.5, we compare our results with that of [40,86] for all $k \geq 2$.

One may refer to Figure 7.1 for the comparison of the theoretical results. The curve for the theoretical result given in [40] (MSB case) will be parallel and slightly below that of the curve for [86] (LSB case) and that is why we have not drawn it separately.

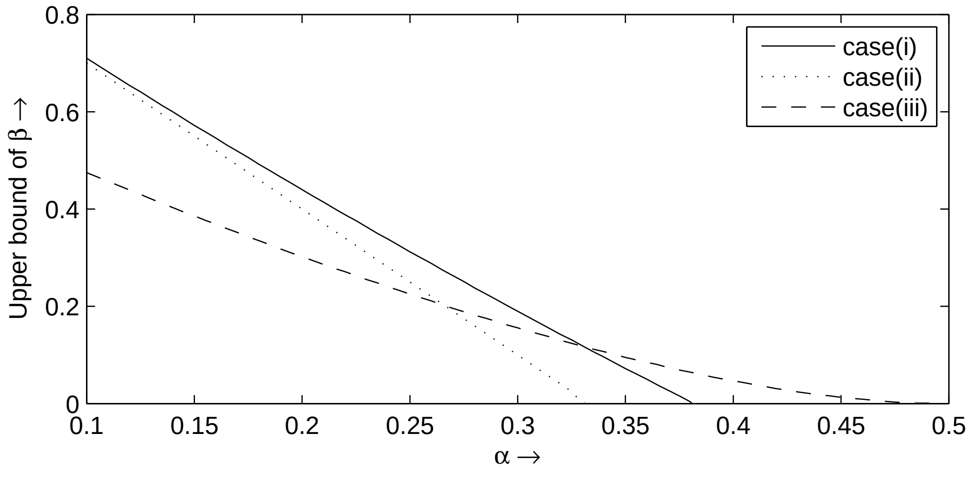

Figure 7.1: Comparison of theoretical results. Case (i): our result that works for both MSBs and LSBs. Case (ii): result of [86] for LSBs and that of [40] for MSBs. Case (iii): result of Chapter 6 for both MSBs and LSBs.

7.4.2 Analysis for $k = 3$

We now explain the case for $k = 3$ in detail.

Theorem 7.11. *Let $N_1 = p_1q_1, N_2 = p_2q_2$ and $N_3 = p_3q_3$, where p_1, p_2, p_3, and q_1, q_2, q_3 are primes. Let N, N_1, N_2, N_3 be of same bitsize and $q_1, q_2, q_3 \approx N^\alpha$, $|p_1 - p_2| < N^\beta$, $|p_1 - p_3| < N^\beta$. Then, under Assumption 1, one can factor N_1, N_2 and N_3 in* poly(log N) *time when*

$$\beta < (1-\alpha)^{\frac{3}{2}} - \alpha,$$

provided that $2\alpha + \beta \leq 1$.

Proof. Let $x_0 = p_2 - p_1$ and $y_0 = p_3 - p_1$. We have $N_1 = p_1q_1, N_2 = p_2q_2 = (x_0 + p_1)q_2, N_3 = (y_0 + p_1)q_3$. Our goal is to recover x_0q_2, y_0q_3 from N_1, N_2 and N_3. Let $X = N^{\alpha+\beta}$. Clearly X is an upper bound of x_0q_2, y_0q_3. Also we have $p_1 \approx N^{1-\alpha}$. When $k = 3$ then

$$P_1 = X^{\frac{m^3}{3}+o(m^3)} N_1^{\frac{m^3}{6}+o(m^3)}.$$

Let $t = \tau m$. To have a manageable formula for P_2, we need to assume $t \leq m + 1$.

Then

$$\begin{aligned} P_2 &= X^{m^3\tau^2+m^3\tau+\frac{m^3\tau^3}{3}+o(m^3)}, \text{ and} \\ \omega &= \frac{m^2}{2}+m^2\tau+\frac{m^2\tau^2}{2}+o(m^2). \end{aligned}$$

Neglecting the $o(m^3)$ terms, the required condition $\det(L) < p_1^{m\omega}$ implies

$$\left(\frac{1}{3}+\tau^2+\tau+\frac{\tau^3}{3}\right)(\alpha+\beta)+\frac{1}{6} < (1-\alpha)\left(\frac{1}{2}+\tau+\frac{\tau^2}{2}\right)$$

which simplifies to the following

$$-\frac{\tau^3}{3}(\alpha+\beta)-\tau^2\left(\frac{3\alpha}{2}+\beta-\frac{1}{2}\right)-\tau(2\alpha+\beta-1)-\frac{5\alpha}{6}-\frac{\beta}{\beta}+\frac{1}{3} > 0. \quad (7.19)$$

To maximize β for a fixed α, the optimal value of τ is $\frac{1-2\alpha-\beta}{\alpha+\beta}$. Putting this optimal value of τ in (7.19), we get the required condition as $-\alpha^3+2\alpha^2-2\alpha\beta-\beta^2-3\alpha+1 > 0$, i.e.,

$$\beta < \sqrt{1-3\alpha+3\alpha^2-\alpha^3}-\alpha.$$

As $\tau \geq 0$, we also need the constraint $2\alpha+\beta \leq 1$. Then under Assumption 1 (as the polynomials are of two variables), we can collect the roots successfully. ■

In Theorem 6.9 of Chapter 6, it has been explained that factorization of N_1, N_2, N_3 will be successful when $\beta < 0.8-2\alpha$. In our case, the upper bound of β is $\sqrt{1-3\alpha+3\alpha^2-\alpha^3}-\alpha$. Now, $0.8-2\alpha < \sqrt{1-3\alpha+3\alpha^2-\alpha^3}-\alpha$ when $\alpha < 0.55$. Hence, our upper bound on β will be greater than that of Theorem 6.9.

7.5 Sublattice and Generalized Bound

In this section, we study a sublattice L' of the lattice L explained in the previous section. This helps in two ways as follows.

- The dimension of the sublattice L' is less than that of L and this helps in actual experiments.
- The theoretical analysis helps us to get a generalized bound for β.

Let us define the following notation:

$$C(\alpha, k) = \frac{k^2(1-2\alpha) + k(5\alpha - 2) - 2\alpha + 1 - \sqrt{k^2(1-\alpha^2) + 2k(\alpha^2 - 1) + 1}}{k^2 - 3k + 2}.$$

We will use this notation in the following theorem as well as in later part of this chapter. Now we present the main result describing the bound on β.

Theorem 7.12. *Consider EPACDP with $g \approx a^{1-\alpha}$ and $\tilde{x}_2 \approx \cdots \approx \tilde{x}_k \approx a^{\alpha+\beta}$. Then, under Assumption 1, one can solve EPACDP in* $\mathrm{poly}\{\log \mathrm{a}, \exp(\mathrm{k})\}$ *time when*

$$\beta \quad < \quad \begin{cases} C(\alpha, k), & \text{for } k > 2 \\ 1 - 3\alpha + \alpha^2, & \text{for } k = 2 \end{cases}$$

with the constraint $2\alpha + \beta \leq 1$.

Proof. We start by explaining the shift polynomials. First we consider the following ones which are same as given in Equation (7.9) in the previous section.

$$H_{j_2,\ldots,j_k}(x_2, \ldots, x_k) = h_2^{j_2} \cdots h_k^{j_k} a_1^{m-j_2-\cdots-j_k}, \tag{7.20}$$

for non-negative integers $j_2, \ldots, j_k$ such that $j_2 + \cdots + j_k \leq m$, where the integer $m \geq 0$ fixed. Further, we define another set of shift polynomials

$$H'_{i_2,0,\ldots,0,j_2,\ldots,j_k}(x_2, \ldots, x_k) = x_2^{i_2} h_2^{j_2} \cdots h_k^{j_k}, \tag{7.21}$$

with the following:

1. $1 \leq i_2 \leq t$, for a positive integer t, and
2. $j_2 + \cdots + j_k = m$, for non-negative integers $j_2, \ldots, j_k$.

Note that this set of shift polynomials is a sub-collection of the polynomials presented in Equation (7.10).

Next, we define L' using the coefficient vectors of

$$\begin{aligned} &H_{j_2,\ldots,j_k}(x_2X_2, \ldots, x_kX_k), \text{ and} \\ &H'_{0,\ldots,i_n,\ldots,0,j_2,\ldots,j_k}(x_2X_2, \ldots, x_kX_k). \end{aligned}$$

Let $X_2 = X_3 = \cdots = X_k = X$ be the common upper bound on each co-ordinate of the root $(\tilde{x}_2, \ldots, \tilde{x}_k)$. The shift polynomials from Equation (7.20) contribute

$$P_1' = \prod_{r=0}^{m} (X^r a_1^{m-r})^{\binom{k+r-2}{r}} = X^{\eta_4} a_1^{\eta_5}$$

with $\eta_4 = \sum_{r=0}^{m} r\binom{k+r-2}{r}$, $\eta_5 = \sum_{r=0}^{m}(m-r)\binom{k+r-2}{r}$, to the determinant of L'. (Note that this P_1' is same as P_1 in Corollary 7.9). The shift polynomials from Equation (7.21) contribute

$$P_2' = \prod_{i_2=1}^{t} (X^{i_2} X^m)^{\binom{k+m-2}{m}} = X^{\eta_6}$$

with $\eta_6 = \sum_{i_2=1}^{t}(i_2+m)\binom{k+m-2}{m}$, to the determinant of L'. The dimension of L' is

$$\omega' = \sum_{r=0}^{m} \binom{k+r-2}{r} + t\binom{m+k-2}{m}.$$

Now, we have $\binom{k+r-2}{r} = \frac{r^{k-2}}{(k-2)!} + o(r^{k-2})$. Using Lemma 4.1 and neglecting lower order terms, we obtain

$$\begin{aligned}
P_1' &\approx X^{\sum_{r=0}^{m} r\frac{r^{k-2}}{(k-2)!}} a_1^{\sum_{r=0}^{m}(m-r)\frac{r^{k-2}}{(k-2)!}} \approx X^{\frac{1}{(k-2)!}\frac{m^k}{k}} a_1^{\frac{1}{(k-2)!}\frac{m^k}{k-1} - \frac{1}{(k-2)!}\frac{m^k}{k}}, \\
P_2' &\approx X^{\sum_{i_2=1}^{t}(i_2+m)\frac{m^{k-2}}{(k-2)!}} \approx X^{\frac{1}{(k-2)!}(\frac{t^2 m^{k-2}}{2} + t m^{k-1})}, \text{ and} \\
\omega' &\approx \sum_{r=0}^{m} \frac{r^{k-2}}{(k-2)!} + t\frac{m^{k-2}}{(k-2)!} \approx \frac{m^{k-1}}{(k-1)(k-2)!} + t\frac{m^{k-2}}{(k-2)!}
\end{aligned}$$

Following Theorem 2.23, the required condition is

$$\det(L') = P_1' P_2' < g^{m\omega'},$$

where g is the common divisor. Let $X = a^{\alpha+\beta}$. Then putting the values of g, X in $\det(L') = P_1' P_2' < g^{m\omega'}$, we get,

$$\left(\frac{m^k}{k} + \frac{m^{k-2}t^2}{2} + m^{k-1}t\right)(\alpha+\beta) + \frac{m^k}{k-1} - \frac{m^k}{k} < (1-\alpha)\left(m^{k-1}t + \frac{m^k}{k-1}\right). \tag{7.22}$$

Now putting $t = \tau m$, ($\tau \geq 0$ is a real number) in (7.22), we get the condition as

$$\left(\frac{1}{k} + \frac{\tau^2}{2} + \tau\right)(\alpha + \beta) + \frac{1}{(k-1)k} < (1-\alpha)\left(\tau + \frac{1}{k-1}\right). \tag{7.23}$$

To maximize β for a fixed α, the optimal value of τ is

$$\tau = \frac{1 - 2\alpha - \beta}{\alpha + \beta}.$$

Putting this optimal value in (7.23), we get the condition as

$$\begin{aligned}4\alpha^2k^2 + {}&4\alpha\beta k^2 + \beta^2k^2 - 8\alpha^2k - 10\alpha\beta k - 3\beta^2k - 4\alpha k^2\\ &- 2\beta k^2 + 2\alpha^2 + 4\alpha\beta + 2\beta^2 + 6\alpha k + 4\beta k + k^2 - 2\alpha - 2\beta - k > 0.\end{aligned}$$

From which we get the required condition as $\beta < C(\alpha, k)$ when $k > 2$, and $\beta < 1 - 3\alpha + \alpha^2$ when $k = 2$. Since $\tau \geq 0$, we also need the constraint $1 - 2\alpha - \beta \geq 0$. Then, under Assumption 1 (as the polynomials are of more than one variable), we can collect the roots successfully. ■

7.5.1 Implicit Factorization problem with shared MSBs and LSBs together

So far we continued our discussion for the MSB case for better understanding. Now we show that the same technique works as well when MSBs and LSBs are shared together. This also takes care of the case when only LSBs are shared. As before, consider $N_1 = p_1q_1, N_2 = p_2q_2, \ldots, N_k = p_kq_k$, where $p_1, p_2, \ldots, p_k$ and $q_1, q_2, \ldots, q_k$ are primes. It is also considered that $p_1, p_2, \ldots, p_k$ are of same bitsize and so are $q_1, q_2, \ldots, q_k$. We also assume that some amount of LSBs as well as some amount of MSBs of $p_1, p_2, \ldots, p_k$ are same.

Theorem 7.13. *Let $q_1, q_2, \ldots, q_k \approx N^\alpha$. Consider that $\gamma_1 \log_2 N$ many MSBs and $\gamma_2 \log_2 N$ many LSBs of $p_1, p_2, \ldots, p_k$ are same. Define $\beta = 1 - \alpha - \gamma_1 - \gamma_2$. Then, under Assumption 1, one can factor $N_1, N_2, \ldots, N_k$ in* $\mathrm{poly}\{\log \mathrm{N}, \exp(\mathrm{k})\}$ *if*

$$\beta < \begin{cases} C(\alpha, k), & \text{for } k > 2,\\ 1 - 3\alpha + \alpha^2, & \text{for } k = 2,\end{cases}$$

with the constraint $2\alpha + \beta \leq 1$.

Proof. It is given that $\gamma_1 \log_2 N$ many MSBs and $\gamma_2 \log_2 N$ many LSBs of $p_1, p_2, \ldots, p_k$ are same. We consider $\gamma_1 \log_2 N$ and $\gamma_2 \log_2 N$ as integers, so it is clear that $N^{\gamma_1}, N^{\gamma_2}$ are integers too. Thus, we can write the following equations.

$$\begin{aligned} p_2 &= p_1 + N^{\gamma_2}\tilde{x}_2, \\ p_3 &= p_1 + N^{\gamma_2}\tilde{x}_3, \\ &\vdots \\ p_k &= p_1 + N^{\gamma_2}\tilde{x}_k. \end{aligned}$$

Using the above relations, we have[1]

$$N_i q_1 - N_1 q_i = N^{\gamma_2}\tilde{x}_i q_i q_1 \text{ for } 1 < i \le k. \tag{7.24}$$

Suppose, $\mu N^{\gamma_2} \equiv 1 \bmod N_1$. Now, multiplying Equation (7.24) by μ, we get $\mu N_i q_1 - \tilde{x}_i q_i q_1 \equiv 0 \pmod{N_1}$. Let $b_i \equiv \mu N_i \pmod{N_1}$ for $1 < i \le k$. (Note that, for any i, b_i is of $O(N_1)$, i.e., $O(N)$.) Thus we have,

$$b_i q_1 - \tilde{x}_i q_i q_1 \equiv 0 \pmod{N_1} \quad \Rightarrow \quad b_i - \tilde{x}_i q_i \equiv 0 \pmod{p_1} \text{ for } 1 < i \le k.$$

Our first aim is to find $\tilde{x}_i q_i$ from the knowledge of N_1 and $b_2, \ldots, b_k$. Then, using $\tilde{x}_i q_i$, we want to find the factorization of N_i for $1 \le i \le k$. Here we have $\tilde{x}_i \approx N^{1-\alpha-\gamma_1-\gamma_2}$ for $1 \le i \le k$. Then, following the similar technique as in the proof of Theorem 7.12, the desired result is achieved. ■

We extend the results related to MSBs (as in Theorem 7.12 and earlier) in the case of LSBs as well as in the case of MSBs and LSBs taken together (as in Theorem 7.5.1). As this method works for the case where MSBs and LSBs are considered together, it also works for the case where only the LSBs are shared. In such a case, we can consider that just a single bit from the MSB side (the first MSB, which is surely 1 for all the primes) is shared.

[1] One may note that similar equations as in Equation (7.24) have been used in [40] to construct the lattice only for the case of most significant bits. However, here we use these for the case when the equal bits are spread out between most and least significant bits.

7.5.2 Comparison with the work of [40, 86, 107]

Let us now compare our result with that of [86] for the general case. The strategy of [86] considers equality in some LSBs of $p_1, p_2, \ldots, p_k$ and we consider the same here following the result in Section 7.5.1. The strategy of [86] works when

$$\beta \leq 1 - \alpha - \frac{k}{k-1}\alpha = 1 - \frac{2k-1}{k-1}\alpha.$$

As $\beta > 0$, one may note $\alpha < \frac{k-1}{2k-1}$, i.e, $\alpha < \frac{1}{2}$.

We have already discussed in Section 7.4.1 that for the case $k = 2$ our result is better than that of [86]. The results of Theorem 7.12 for $k = 2$ and Theorem 7.10 are same, since L and L' are same for $k = 2$. However, L and L' become different for $k > 2$. For $k > 2$, $C(\alpha, k) > 1 - \frac{2k-1}{k-1}\alpha$. Thus, for any k, $k \geq 2$, we need smaller amount of bit sharing in LSBs for implicit factorization than the number of bit sharing in LSBs achieved in [86]. Our upper bound on β is

$$\frac{k - 1 - \sqrt{k^2 + 2\alpha^2 k - \alpha^2 k^2 - 2k + 1}}{k^2 - 3k + 2}$$

more than the upper bound on β in [86]. Thus, the gap between our bound and that of [86] reduces as k increases. To summarize, we have the following observations.

1. Our theoretical result is better than that of [86] from the point that it requires less LSBs to be equal than the number of LSBs in case of [86].

2. Both our result as well as that of [86, Theorem 7] are of time complexity $\mathrm{poly}\{\log \mathrm{N}, \exp(\mathrm{k})\}$. However, the lattice dimension in the formulation of [86] is much smaller (exactly k) than the lattice dimension following our approach (exponential in k). Experimentally our results provide superior outcome for $k = 3$ and similar kind of outcome for $k = 4$, though we need more time than that of [86]. Experiments for large k is not possible with our strategy in this section. To overcome this, we present a technique (in Section 7.6) that provides results for larger values of k.

3. The strategy of [86] could be extended for balanced RSA moduli, which we could not achieve in our case.

Let us now present some numerical values (both theoretical as well as experimental) for comparison with [86] in Table 7.3. In the * marked rows, experimental

k	Bitsize of p_i, q_i	No. of shared LSBs [86] in p_i				No. of shared LSBs (our) in p_i			
	$(1-\alpha)\log_2 N, \alpha \log_2 N$	Theory	Expt.	LD	Time	Theory	Expt.	LD	Time
3	750, 250	375	378	3	< 10	352	367	56	41.92
* 3	700, 300	450	452	3	< 1	416	431	56	59.58
* 3	650, 350	525	527	3	< 1	478	499	56	74.54
# 3	600, 400	600	-	-	-	539	562	56	106.87
* 4	750, 250	334	336	4	< 1	320	334	65	32.87
* 4	700, 300	400	402	4	< 1	380	400	65	38.17
* 4	650, 350	467	469	4	< 1	439	471	65	39.18
* 4	600, 400	534	535	4	< 1	497	528	65	65.15

Table 7.3: For 1000 bit N, theoretical and experimental data of the number of shared LSBs in [86] and shared LSBs in our case. (Time in seconds)

data is not available from [86], and we perform the experiments following the method of [86]. In the # marked row, the method of [86] does not work as all the bits of the primes p_1, p_2, p_3 need to be same.

k	Bitsize of p_i, q_i	No. of shared MSBs [40] in p_i				No. of shared MSBs (our) in p_i			
	$(1-\alpha)\log_2 N, \alpha \log_2 N$	Theory	Expt.	LD	Time	Theory	Expt.	LD	Time
2	874, 150	303	302	2	< 1	278	289	16	1.38
2	824, 200	403	402	2	< 1	361	372	16	1.51
2	774, 250	503	502	2	< 1	439	453	16	1.78
2	724, 300	603	602	2	< 1	513	527	16	2.14
3	874, 150	231	230	3	< 1	217	230	56	29.24
3	824, 200	306	304	3	< 1	286	304	56	36.28
3	774, 250	381	380	3	< 1	352	375	56	51.04
3	724, 300	456	455	3	< 1	417	441	56	70.55
3	674, 350	531	530	3	< 1	480	505	56	87.18
3	624, 400	606	604	3	< 1	540	569	56	117.14

Table 7.4: For 1024-bit N, theoretical and experimental data of the number of shared MSBs in [40] and shared MSBs in our case. (Time in seconds)

Faugére et al [40] (independently at the same time of our work) presented a different lattice based approach for the problem of implicit factorization with shared MSBs of $p_1, p_2, \ldots, p_k$. The strategy of [40] works when

$$\begin{aligned}\beta &\leq 1 - \alpha - \frac{k}{k-1}\alpha - \frac{1}{\log_2 N} - \frac{k}{2(k-1)\log_2 N}\left(2 + \frac{\log_2 k}{k} + \log_2(\pi e)\right) \\ &= 1 - \frac{2k-1}{k-1}\alpha - \frac{1}{\log_2 N} - \frac{k}{2(k-1)\log_2 N}\left(2 + \frac{\log_2 k}{k} + \log_2(\pi e)\right).\end{aligned}$$

Similar to our comparison with that of [86] for LSB case, it can be noted that our method requires less number of MSBs to be shared compared to [40] and the gap between our bound and that of [40] reduces as k increases. The numerical data, both theoretical and experimental, are presented in Table 7.4 for a clear comparison of our work with that of [40].

Very recently, M. Ritzenhofen [107] presented a distinct lattice based approach for the problem of implicit factorization with shared MSBs and LSBs of $p_1, p_2, \ldots, p_k$. The strategy of [107, Theorem 6.1.7] works when $\beta \leq 1 - \alpha - \frac{k}{k-1}\alpha$. Similar to our comparison with that of [86] for LSB case, it can be noted that our method requires less number of bits to be shared compared to [107].

We have explained our results for the MSB case as well as LSB case and compared with state of the art literature. The experimental results in both the cases are of similar quality using our techniques. Similar results are achieved in our case if one considers sharing of MSBs and LSBs together in the primes $p_1, p_2, \ldots, p_k$. Thus, we do not repeat these results.

7.6 Improved Results for Larger k

In [128, Section 5.2], the authors studied the EPACDP for analyzing the security of their scheme. Initially this strategy has been analyzed in [73, 74]. Based on the idea presented in [128], we get Theorem 7.14. The result in Theorem 7.14 below is not exactly presented in a similar form in [128].

In case of EPACDP, one can write

$$\begin{array}{rcl} a_1 & = & gq_1, \\ \tilde{a}_2 & = & gq_2 - \tilde{x}_2, \\ & \vdots & \\ \tilde{a}_k & = & gq_k - \tilde{x}_k. \end{array}$$

Let us construct the matrix

$$M = \begin{pmatrix} 2^{\rho} & \tilde{a}_2 & \tilde{a}_3 & \ldots & \tilde{a}_k \\ 0 & -a_1 & 0 & \ldots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \ldots & -a_1 \end{pmatrix}$$

where $2^\rho \approx \tilde{x}_2$. One can note that $(q_1, q_2, \ldots, q_k) \cdot M = (2^\rho q_1, -q_1\tilde{x}_2, \ldots, q_1\tilde{x}_k) = \mathbf{b}$, say. It can be checked that

$$||\mathbf{b}|| < \sqrt{k} \cdot a^{2\alpha+\beta}. \tag{7.25}$$

Moreover, $|\det(M)| = 2^\rho a_1^{k-1} \approx a^{\alpha+\beta+k-1}$. According to Minkowski's theorem (see [106] for details), we know that there is a vector $\mathbf{v}$ in the lattice L corresponding to M such that

$$||\mathbf{v}|| < \sqrt{k} \cdot a^{\frac{\alpha+\beta+k-1}{k}}, \tag{7.26}$$

Now we consider the following assumption.

Assumption 2. The vector $\mathbf{b}$ is one of the shortest vectors, and the next shortest vector is significantly larger than $||\mathbf{b}||$.

Under this assumption, and from (7.25) and (7.26), we get $\mathbf{b}$ from L if

$$a^{2\alpha+\beta} < a^{\frac{\alpha+\beta+k-1}{k}} \quad \Leftrightarrow \quad \beta < \frac{k-1+\alpha-2\alpha k}{k-1}.$$

The running time is determined by the time to calculate a shortest vector in L which is polynomial in $\log a$ but exponential in k. Thus, we get the following.

Theorem 7.14. *Consider EPACDP with $g \approx a^{1-\alpha}$ and $\tilde{x}_2 \approx \tilde{x}_k \approx a^{\alpha+\beta}$. Then, under Assumption 2, one can solve EPACDP in* $\mathrm{poly}\{\log \mathrm{a}, \exp(\mathrm{k})\}$ *time when,*

$$\beta \;<\; 1 - \frac{2k-1}{k-1}\alpha. \tag{7.27}$$

So, the bound in (7.27) works for the case when MSBs are shared. Further, using the idea of Section 7.5.1, the bound in (7.27) can as well be used for the LSB case or in the case where we consider the MSBs and LSBs together. From the analysis presented in Section 7.5.2, it is clear that this bound is worse than the bound presented in Theorem 7.12. However, this result helps us to provide much better experimental performance for larger values of k, that could not be achieved by the method in Section 7.5.

7.6.1 Comparison with the work of [40, 86]

Theorem 7.14 states that the time complexity is $\mathrm{poly}\{\log \mathrm{a}, \exp(\mathrm{k})\}$. However, under the assumption that "the shortest vector of the lattice L can be found by the LLL algorithm", the complexity becomes poly{log a, k}. This happens in practice as observed in [86] too.

In Table 7.5, we present a comparison of our experimental results with those in [86, Table 1, Section 6.2]. One may note that both our results and the results of [86] are of similar quality. We have implemented the method of [86] for comparison, and the data is presented in Table 7.5.

α	k	Theoretical bound	Results of [86]		Our results	
		(same for both)	Experiments	Time in seconds	Experiments	Time in seconds
0.25	3	375	377	< 1	376	< 1
0.35	10	389	391	< 1	390	< 1
0.40	100	405	408	50.36	407	28.21
0.44	50	449	452	7.09	451	4.04
0.48	100	485	492	68.88	488	36.36

Table 7.5: For 1000 bit N, theoretical and experimental data of the number of shared LSBs in [86] and shared LSBs in our case.

As we have already discussed, in the approach of [40], the number of shared MSBs should be greater than or equal to $\frac{k}{k-1}\alpha\log_2 N + 6$ for $k \geq 3$. In our case, putting the upper bound of β, number of shared bits should be greater than

$$\left(1 - \alpha - \left(1 - \frac{2k-1}{k-1}\alpha\right)\right)\log_2 N = \frac{k}{k-1}\alpha\log_2 N.$$

We have implemented the method of [40] for comparison with our strategy. Note that the data presented in Table 7.6 match with those in [40, Tables 4, 5].

Advantages of our approach over [40] are as follows.

1. Our theoretical result in this section is slightly better than that of [40] in terms of number of shared MSBs.
2. In this section, the matrix corresponding to the lattice is a square one, but it is rectangular in the method of [40]. Hence, the calculation of determinant for the lattices is easier in our method.
3. In the presence of k many RSA moduli, we have to reduce a $k \times k$ matrix, whereas the size of the matrix in [40] is $k \times \frac{1}{2}k(k+1)$. Hence in practical circumstances, the matrix reduction step in case of [40] takes more time than ours. Further, from the experimental results presented in Table 7.6, it is clear that our strategy requires much less time than the method of [40].

k	Bitsize of p_i, q_i	No. of shared MSBs [40] in p_i				No. of shared MSBs (our) in p_i			
		Theory	Expt.	LD	Time (sec)	Theory	Expt.	LD	Time (sec)
10	874, 150	171	170	10	< 1	166	170	10	< 1
10	824, 200	227	225	10	< 1	220	225	10	< 1
10	774, 250	282	280	10	< 1	274	280	10	< 1
10	724, 300	338	334	10	< 1	328	332	10	< 1
10	674, 350	393	390	10	< 1	382	388	10	< 1
10	624, 400	449	446	10	< 1	435	444	10	< 1
40	874, 150	158	157	40	12.74	154	157	40	< 1
40	824, 200	209	206	40	17.42	205	206	40	< 1
40	774, 250	261	258	40	21.64	256	258	40	1.13
40	724, 300	312	309	40	24.17	307	308	40	1.26
40	674, 350	363	361	40	29.87	358	360	40	1.48
40	624, 400	414	412	40	34.69	409	410	40	1.75
100	874, 150	155	154	100	299.64	152	153	100	5.63
100	824, 200	206	205	100	525.67	202	204	100	9.36
100	774, 250	257	257	100	781.42	253	255	100	14.11
100	724, 300	307	307	100	1053.66	303	305	100	18.61
100	674, 350	358	357	100	1415.02	353	355	100	24.16
100	624, 400	408	408	100	2967.75	404	406	100	29.95

Table 7.6: For 1024-bit N, theoretical (bound for [40] and in our case) and experimental data of the number of shared MSBs in [40] and shared MSBs in our case.

In this section, we have presented our results for the MSB case as well as LSB case and compared with [40, 86]. The experimental results in both the cases are of similar quality using our techniques. Similar results are achieved in our case if one considers sharing of MSBs and LSBs together, and thus we do not repeat these results.

7.6.2 Comparing the Methods with respect to EPACDP

So far we have discussed the methods towards applying them in implicit factorization. Now, let us compare the method presented in this section with that of Section 7.5 for EPACDP itself.

Let $g \approx a^{1-\alpha}$ and $\tilde{x}_i \approx a^{\beta}$ for $i = 2$ to k. Following similar kind of calculations as in the proof of Theorem 7.12, we get

$$\beta < \begin{cases} C(\alpha, k) + \alpha & \text{for } k > 2, \\ 1 - 2\alpha + \alpha^2 & \text{for } k = 2, \end{cases} \tag{7.28}$$

The approach in this section provides the bound as

$$\beta \ < \ 1 - \left(\frac{k}{k-1}\right)\alpha.$$

When $\alpha \geq \frac{k-1}{k}$, we can not get the common divisor by the proposed method in this section. However, we get results in such situations using the results of Section 7.5. In Table 7.7, we present few such experimental outcomes for the method discussed in Section 7.5.

k	$\alpha = (k-1)/k$	Bound of β		LD	Time
		Equation (7.28)	Experimental		
3	2/3	0.1835	0.135	25	10.72
4	3/4	0.1464	0.09	28	28.75

Table 7.7: EPACDP: Experimental results for 1000-bit a.

To conclude this section, let us point out a few issues:

- Considered theoretically, the idea of Section 7.5 is always better than that of this section.
- The method of this section works better than the idea of Section 7.5 is case of practical experiments for larger values of k.
- There are some situations with small values of k, where the idea of Section 7.5 works better than the strategy presented in this section.

7.7 EGACDP

So far we have concentrated on EPACDP, i.e., we have considered that the first element is exactly known. However, the more general problem is when the first element is also approximated by some known term. This is what we refer to as Extended General Approximate Common Divisor Problem (EGACDP, as in Problem Statement 2) and we study this problem in the current section. Towards solving EPACDP, we presented two different techniques, one in Section 7.4 and another in Section 7.6. We will try similar methods in this section as Method I and Method II respectively.

In case of EGACDP, we have

$$\begin{aligned}\tilde{a}_1 &= gq_1 - \tilde{x}_1,\\ \tilde{a}_2 &= gq_2 - \tilde{x}_2,\\ &\vdots\\ \tilde{a}_k &= gq_k - \tilde{x}_k,\end{aligned}$$

where $\tilde{a}_1, \ldots, \tilde{a}_k$ are known. The goal is to find g from the knowledge of the approximates $\tilde{a}_1, \ldots, \tilde{a}_k$.

7.7.1 Method I

Towards solving the EGACDP in a manner similar to Section 7.4, consider the polynomials

$$\begin{aligned}h_1(x_1, x_2, \ldots, x_k) &= \tilde{a}_1 + x_1,\\ &\vdots\\ h_k(x_1, x_2, \ldots, x_k) &= \tilde{a}_k + x_k,\end{aligned} \tag{7.29}$$

where $x_1, x_2, \ldots, x_k$ are the variables. Clearly, g (of Problem Statement 2) divides $h_i(\tilde{x}_1, \tilde{x}_2, \ldots, \tilde{x}_k)$ for $1 \le i \le k$. Now let us define the shift polynomials

$$h_{s_1,\ldots,s_k}(x_1, x_2, \ldots, x_k) = h_1^{s_1} \cdots h_k^{s_k}, \tag{7.30}$$

for $u \le s_1 + \cdots + s_k \le m$, where u, m are fixed non-negative integers.

Let $X_1, \ldots, X_k$ be the upper bounds of $\tilde{x}_1, \ldots, \tilde{x}_k$ respectively. Now we define a lattice L using the coefficient vectors of $h_{s_1,\ldots,s_k}(x_1X_1, \ldots, x_kX_k)$. Let the dimension of L be ω. One gets $\tilde{x}_1, \ldots, \tilde{x}_k$ (under Assumption 1 and following Theorem 2.23 and Lemma 2.20) using lattice reduction over L, if $\det(L)^{\frac{1}{\omega}} < g^m$, i.e., when $\det(L) < g^{m\omega}$ (neglecting the lower order terms).

Since the lattice dimension $\omega = \sum_{s=u}^{m} \binom{k+s-1}{s}$ is exponential in k, the running time of this strategy will be $\mathrm{poly}\{\log \mathrm{a}, \exp(\mathrm{k})\}$. Thus for small fixed k, this algorithm is polynomial in $\log a$. Formally, we get the following result.

Theorem 7.15. *Under Assumption 1, the EGACDP can be solved in time* $\mathrm{poly}\{\log \mathrm{a}, \exp(\mathrm{k})\}$ *when* $\det(L) < g^{m\omega}$.

Since in this case the matrix corresponding to the lattice L is not square, finding $\det(L)$ may not be easy for general k. Further, for large k, dimension of L will be very large.

7.7.2 Method II

Here we follow the idea of Section 7.6. We have

$$\begin{aligned}\tilde{a}_1 &= gq_1 - \tilde{x}_1,\\ \tilde{a}_2 &= gq_2 - \tilde{x}_2,\\ &\vdots\\ \tilde{a}_k &= gq_k - \tilde{x}_k,\end{aligned}$$

where $\tilde{a}_1, \ldots, \tilde{a}_k$ are known and $\tilde{a}_i \approx a$ for $1 \le i \le k$. Suppose, $\tilde{x}_i \approx a^\beta$ for $1 \le i \le k$ and $g \approx a^{1-\alpha}$. Then $q_i \approx a^\alpha$ for $i \in [1, k]$. Let us construct

$$M = \begin{pmatrix} 2^\rho & \tilde{a}_2 & \tilde{a}_3 & \ldots & \tilde{a}_k \\ 0 & -\tilde{a}_1 & 0 & \ldots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \ldots & -\tilde{a}_1 \end{pmatrix}$$

where $2^\rho \approx 2\tilde{x}_1$. One can note that $(q_1, q_2, \ldots, q_k) \cdot M = (2^\rho q_1, \tilde{x}_1 q_2 - q_1\tilde{x}_2, \ldots, \tilde{x}_1 q_k - q_1 \tilde{x}_k) = \mathbf{b}$, say. It can be checked that

$$||\mathbf{b}|| < 2\sqrt{k}a^{\alpha+\beta}. \tag{7.31}$$

Moreover, $|\det(M)| = 2^\rho(\tilde{a}_1)^{k-1} \approx 2a^{\beta+k-1}$. Following Minkowski's theorem, there is a vector $\mathbf{v}$ in the lattice L corresponding to M such that

$$||\mathbf{v}|| < \sqrt{k}2^{\frac{1}{k}}a^{\frac{\beta+k-1}{k}}. \tag{7.32}$$

Under Assumption 2 (Section 7.6), and from (7.31) and (7.32), one can obtain $\mathbf{b}$ from L if

$$a^{\alpha+\beta} < a^{\frac{\beta+k-1}{k}} \quad \Leftrightarrow \quad \beta < 1 - \frac{k}{k-1}\alpha,$$

neglecting the terms 2 and $2^{\frac{1}{k}}$. With the knowledge of $\mathbf{b}$, one can find q_1, from which g is obtained if $|\tilde{x}_1| \leq q_1$. So we need

$$1 - \frac{k}{k-1}\alpha \leq \alpha \quad \Rightarrow \quad \alpha \geq \frac{k-1}{2k-1}.$$

The running time is determined by the time to calculate a shortest vector in L which is polynomial in $\log a$ but exponential in k. Thus, we get the following result.

Theorem 7.16. *Consider EGACDP with $g \approx a^{1-\alpha}$ and $\tilde{x}_1 \approx \tilde{x}_2 \approx \cdots \approx \tilde{x}_k \approx a^{\beta}$. Then, under Assumption 2, one can solve EGACDP in* $\mathrm{poly}\{\log \mathrm{a}, \exp(\mathrm{k})\}$ *time when*

$$\beta \; < \; 1 - \frac{k}{k-1}\alpha, \tag{7.33}$$

provided that $\alpha \geq \frac{k-1}{2k-1}$.

7.7.3 Experimental Results

In this section we present a few experimental results for both the methods, as shown in Table 7.8. When $k = 3$ and $1 - \alpha = 0.25$, we have $1 - \frac{k}{k-1}\alpha < 0$. In such a situation one can not get results using Method II, but Method I will succeed. For example, Method I succeeds given 250-bit g for $k = 3, 4$, whereas Method II does not provide results in such cases.

Results using Method I.				
k	g	error	LD	Time (in sec.)
3	250-bit	36-bit	31	11.72
3	500-bit	245-bit	31	2.85
4	250-bit	82-bit	65	210.91
4	500-bit	320-bit	65	63.04

Results using Method II.				
k	g	error	LD	Time (in sec.)
3	500-bit	249-bit	3	< 1
4	500-bit	331-bit	4	< 1
10	500-bit	441-bit	10	< 1
50	500-bit	487-bit	50	4.62
100	500-bit	490-bit	100	40.17

Table 7.8: EGACDP: Experimental results for 1000-bit a.

For large values of k, we can not perform experiments corresponding to Method I due to high lattice dimensions. Theorem 7.16 states that the time complexity is $\mathrm{poly}\{\log \mathrm{a}, \exp(\mathrm{k})\}$. However, under the assumption that "the shortest vector of the lattice L can be found by the LLL algorithm", the complexity becomes poly$\{\log \mathrm{a}, \mathrm{k}\}$. Thus, Method II will work successfully and we can easily obtain the experimental results using LLL up to $k \leq 100$.

7.8 Conclusion

In this chapter we first present two applications of partially approximate common divisor problem (PACDP) of [61]. Next we present a generalization of the partially approximate common divisor problem (PACDP) [61] which we term as Extended Partially Approximate Common Divisor Problem (EPACDP). We also study the extension of GACDP (General Approximate Common Divisor Problem) [61].

EPACDP immediately relates to the implicit factorization problem introduced in [86] and already elaborated in Chapter 6. We consider the case when some MSBs and/or LSBs of the primes $p_1, p_2, \ldots, p_k$ are equal (but unknown). This covers the case when the LSBs are equal in [86] and MSBs are equal in [40]. Our strategy provides new and improved theoretical as well as experimental results.

This chapter marks the end of our technical results. In the next chapter, we conclude the thesis by summarizing our results and mentioning related open problems to introduce areas of further research in similar directions.

Chapter 8

Conclusion

In this chapter, we conclude the thesis. We have studied weaknesses of RSA cryptosystem, and results related to integer factorization, throughout Chapters 3 to 7. We exploited the vulnerabilities of RSA arising due to weak encryption/decryption keys, information about the bits of RSA primes, and implicit knowledge about the same. Most of our results are based on lattice basis reduction techniques applied to finding solutions to integer and modular polynomials.

We revisit the chapters one-by-one to summarize the thesis. We mention the existing results and prior work (if any) in the direction. Most importantly, we present the crux of the chapters, that is our contributions, improvements and extensions to existing methods. Finally, we also discuss the future scope for research and potential open problems in respective field of study.

8.1 Summary of Technical Results

Chapter 1 provided the introduction to the thesis, while Chapter 2 covered some mathematical topics the reader should know before reading the thesis comfortably. The main technical results of the thesis are discussed in Chapters 3 to 7, and the highlights of these chapters are as follows.

Chapter 3: Class of Weak Encryption Exponents

Blömer and May [9] have shown that N can be factored in polynomial time if the public exponent e satisfies $ex + y \equiv 0 \pmod{\phi(N)}$, with $x \leq \frac{1}{3}N^{\frac{1}{4}}$ and $|y| = O(N^{-\frac{3}{4}}ex)$. Some extensions considering the difference $p-q$ have also been studied. The number of such weak keys has been estimated as $N^{\frac{3}{4}-\epsilon}$. In a similar direction, more weak keys are presented by Nitaj [96]. Nitaj proved that if e satisfies $eX - (p-u)(q-v)Y = 1$ with $1 \leq Y < X < 2^{-\frac{1}{4}}N^{\frac{1}{4}}$, $|u| < N^{\frac{1}{4}}$, $v = \left[-\frac{qu}{p-u}\right]$, and if all the prime factors of $p-u$ or $q-v$ are less than 10^{50}, then N can be factored from the knowledge of N, e. The number of such weak exponents is estimated as $N^{\frac{1}{2}-\epsilon}$.

We concentrate on the cases when $e\,(= N^{\alpha})$ satisfies $eX - ZY = 1$, given $|N - Z| = N^{\tau}$. Using the idea of Boneh and Durfee [14,15], we show that the LLL algorithm can be efficiently applied to get Z when $|Y| = N^{\gamma}$ and

$$\gamma < 4\alpha\tau\left(\frac{1}{4\tau} + \frac{1}{12\alpha} - \sqrt{\left(\frac{1}{4\tau} + \frac{1}{12\alpha}\right)^2 + \frac{1}{2\alpha\tau}\left(\frac{1}{12} + \frac{\tau}{24\alpha} - \frac{\alpha}{8\tau}\right)}\right).$$

This idea substantially extends the class of weak keys presented in [96] when $Z = \psi(p,q,u,v) = (p-u)(q-v)$. Further, we consider $Z = \psi(p,q,u,v) = N - pu - v$ to provide a new class of weak keys in RSA. This idea does not require any kind of factorization as in [96]. A very conservative estimate for the number of such weak exponents is $N^{0.75-\epsilon}$, where $\epsilon > 0$ is arbitrarily small for suitably large N.

Chapter 4: More than one Decryption Exponent

From the results of Howgrave-Graham et al [62] and Hinek et al [55] we know that in the presence of n many decryption exponents, one can factor N when the decryption exponents $d_i < N^{\delta}$, for $1 \leq i \leq n$, where

$$\delta < \begin{cases} \min\left\{\frac{(2n+1)\cdot 2^n - (2n+1)\binom{n}{\frac{n}{2}}}{(2n-2)\cdot 2^n + (4n+2)\binom{n}{\frac{n}{2}}}, 0.5\right\} & \text{if } n \text{ is even} \\ \\ \min\left\{\frac{(2n+1)\cdot 2^n - 4n\cdot\binom{n-1}{\frac{n-1}{2}}}{(2n-2)\cdot 2^n + 8n\cdot\binom{n-1}{\frac{n-1}{2}}}, 0.5\right\} & \text{if } n \text{ is odd} \end{cases}$$

We improved this bound by showing that if n many decryption exponents $(d_1, \ldots, d_n)$ are used with the same N, then RSA is insecure when $d_i < N^{\frac{3n-1}{4n+4}}$, for all $1 \leq i \leq n$ and $n \geq 2$. Though, the time complexity of our technique as well as the technique of [62] are polynomial in the bitsize of N, both are exponential in the number of exponents n.

Chapter 5: Prime Reconstruction given a few Bits

One major class of RSA attacks exploit partial knowledge of the RSA secret keys or the primes. Rivest and Shamir [109] pioneered these attacks using Integer Programming and factored RSA modulus given two-third of the LSBs of a factor. Later, a seminal paper [24] by Coppersmith proved that factorization of the RSA modulus can be achieved given half of the MSBs of a factor. In Crypto 2009, Heninger and Shacham [50] proposed a method to reconstruct the RSA private keys given a certain fraction of the bits, distributed at random.

In this chapter, we revisit the work of [50] and provide a combinatorial model for the search where some random bits of the primes are known. This shows how one can factorize N given the knowledge of random bits in the least significant halves of the primes. We also explain a lattice based strategy in this direction to rectify a shortcoming of the reconstruction algorithm presented in [50]. More importantly, we study how N can be factored given the knowledge of some blocks of bits in the most significant halves of the primes. We present improved theoretical result and experimental evidences in this direction.

Chapter 6: Implicit Factorization

In PKC 2009, May and Ritzenhofen [86] explained the problem of implicit factorization. Consider $N_1 = p_1q_1, N_2 = p_2q_2, \ldots, N_k = p_kq_k$, where $p_1, p_2, \ldots, p_k$ and $q_1, q_2, \ldots, q_k$ are primes. Further assume that $p_1, p_2, \ldots, p_k$ are of same bitsize and so are $q_1, q_2, \ldots, q_k$. Given that certain portions of bit pattern in $p_1, p_2, \ldots, p_k$ are common, the question is under what conditions it is possible to factor $N_1, N_2, \ldots, N_k$ efficiently. In [86], the result was based under the assumption that some amount of LSBs are same in $p_1, p_2, \ldots, p_k$.

We explore the same problem with a different lattice-based strategy for $k = 2, 3$. In a general framework, our method works when implicit information is available related to Least Significant as well as Most Significant Bits. Given $q_1, q_2, q_3 \approx N^\alpha$, we show that one can factor N_1, N_2, N_3 simultaneously in poly(log N) time when

p_1, p_2, p_3 share certain amount of MSBs and/or LSBs. We also study the case when p_1, p_2 share some contiguous bits in the middle. Our strategy presents new and encouraging results in this direction. Moreover, some of the observations in [86] get improved when we apply our ideas for the LSB case.

Chapter 7: Approximate Integer Common Divisor

In CaLC 2001, Howgrave-Graham [61] proposed two methods to find the Greatest Common Divisor (GCD) of two large integers in the following two cases.

- One of the integers is known exactly and the other is known approximately.
- Both the integers are known approximately.

In this chapter, we first present two applications of the technique of [61], as follows.

Application 1: Consider $N = pq$, where p, q are large primes and $p > q$. We show that factoring N is deterministic polynomial time equivalent to finding $q^{-1} \bmod p$.

Application 2: Consider the problem of finding smooth integers in a short interval as studied by Boneh [12] in STOC 2000. We find improved result using the idea of [61], which is different from the method proposed in [12].

Next, we analyze how to calculate the GCD of k (≥ 2) many large integers given their approximations, which in turn relates to the implicit factorization problem. Introduced by May and Ritzenhofen [86] in PKC 2009, this was studied under the assumption that some Least Significant Bits (LSBs) of certain primes are same.

Our strategy can be applied to the implicit factorization problem in a general framework considering the equality of (i) Most Significant Bits (MSBs), or (ii) Least Significant Bits (LSBs), or (iii) both the MSBs and LSBs together. We present new and improved theoretical as well as experimental results in comparison with existing works.

8.2 Overview of Root Finding Techniques

Throughout the thesis, a common approach for analysis (in most of the cases) was to solve integer integer or modular polynomials using lattice based techniques. We have thoroughly discussed the lattice based root finding methods that are present in the current literature. Our discussion covered the earlier techniques of finding small roots of modular polynomials by Coppersmith [23], as well as the general result by Jochemsz and May [65]. We have presented Coron's [28] method to find roots of bivariate integer polynomials, and have also discussed the general version of the same proposed by Jochemsz and May [65].

Case I: Modular Polynomial f_N		
Monomials of f_N	Bound	Ref.
$1,\, x, y, xy$	$X \cdot Y < N^{\frac{2}{3}}$	Sec. 5.2

Case II: Integer Polynomial f		
Monomials of f	Bound	Ref.
$1, x_1, \ldots, x_{n+1}$, $x_2x_{n+2}, \ldots, x_{n+1}x_{n+2}$	$X_1^{\frac{1}{n+1}\left(\frac{1}{n+2}+\frac{\tau}{n}\right)} \cdot (X_2 \cdots X_{n+1})^{\left(\frac{1}{n}\left(\frac{1}{n+2}+\frac{\tau}{n+1}\right)\right)}$ $X_{n+2}^{\left(\frac{1}{2(n+2)}+\frac{\tau}{n+1}+\frac{\tau^2}{2n}\right)} < W^{\left(\frac{1}{n+1}\left(\frac{1}{n+2}+\frac{\tau}{n}\right)\right)}$	Sec. 4.1
$1, x, y, yz, xyz$	$X^{\left(\frac{1}{2}+\frac{3}{2}\tau\right)} \cdot Y^{\left(\frac{5}{6}+\frac{3}{2}\tau\right)} \cdot Z^{\left(\frac{1}{2}+\frac{3}{2}\tau+\tau^2\right)} < W^{\left(\frac{1}{3}+\tau\right)}$	Sec. 6.1.1
$1, y, z, xyv, xv, xzt$	$X^{\frac{5}{24}} \cdot Y^{\frac{1}{8}} \cdot Z^{\frac{1}{6}} \cdot V^{\frac{1}{8}} \cdot T^{\frac{1}{12}} < W^{\frac{1}{12}}$	Sec. 6.1.4
$1, x, y, yz, yv, xyz, xyv$	$X^{\left(\frac{1}{4}+\tau_1+\tau_2+\frac{3}{2}\tau_1\tau_2\right)} \cdot Y^{\left(\frac{11}{24}+\frac{4}{3}(\tau_1+\tau_2)+\frac{3}{2}\tau_1\tau_2\right)}$ $\cdot Z^{\left(\frac{1}{6}+\frac{2}{3}(\tau_1+\tau_2)+\frac{3}{4}\tau_1^2+\frac{3}{2}\tau_1\tau_2+\tau_1^2\tau_2\right)}$ $\cdot V^{\left(\frac{1}{6}+\frac{2}{3}(\tau_1+\tau_2)+\frac{3}{4}\tau_2^2+\frac{3}{2}\tau_1\tau_2+\tau_1\tau_2^2\right)}$ $< W^{\left(\frac{1}{8}+\frac{1}{2}(\tau_1+\tau_2)+\tau_1\tau_2\right)}$	Sec. 6.2

Table 8.1: Bounds for finding roots of a polynomial.

Now, the general root finding methods by Jochemsz and May [65] have been the most effective ones in case of the polynomials we have encountered. In Table 8.1, we give an overview of the results obtained by using the general root finding techniques of [65] to the specific polynomials in our case. The idea of this tabular representation is motivated by similar tables presented in [64, Page 44], and the results we present here extend the array of results provided in [64]. The reader may refer back to the respective sections to obtain the relevant details.

8.3 Open Problems

In this section, we propose a few open problems related to our work. These may lead to new interesting research topics in the related field.

8.3.1 Weak Encryption Keys

One may note that Blömer and May [9] and our work in Chapter 3 present a different classes of weak encryption exponents of cardinality $N^{0.75-\epsilon}$. For fixed N, there are $O(N)$ many encryption exponents e which are less than N. Hence most of the encryption exponents are not weak based on the current knowledge in this field. It is unknown whether there is a class of weak encryption exponents of cardinality substantially greater than $N^{0.75-\epsilon}$. So, we have the following open problem.

Problem 8.1. *Is there any new class of weak encryption exponents of cardinality substantially greater than $N^{0.75-\epsilon}$?*

8.3.2 More than one Decryption Key

Since the time complexity of the work of Howgrave-Graham and Seifert [62] as well as our approach in Chapter 4 are exponential in n, we can not get better experimental results for large n. Hence we have the following question.

Problem 8.2. *Is there any algorithm with runtime polynomial in $(\log_2 N, n)$ which can improve the bound achieved by our method?*

Furthermore, consider the same situation for CRT-RSA. Suppose that $e_1, \ldots, e_k$ are k encryption exponents and $(d_{p_1}, d_{q_1}), \ldots, (d_{p_k}, d_{q_k})$ are the corresponding decryption exponents for the same CRT-RSA modulus N. We know when e is of order N, then one can factor N in polynomial time [66] if $d_p, d_q < N^{0.073}$. So, we can pose the following problem in the same direction.

Problem 8.3. *Is there any polynomial time algorithm to factor N with encryption exponents of order N if $d_{p_i} > N^{0.073}$ and $d_{q_i} > N^{0.073}$ for $1 \leq i \leq k$?*

8.3.3 Reconstruction of Primes

The terms $\{p, q, d, d_p, d_q, q^{-1} \pmod{p}\}$ are stored as a part of the secret key in PKCS #1 [99] to expedite the decryption in CRT-RSA. Now, one may note from Chapter 5 that Heninger and Shacham [50] used random known bits of $\{p, q, d, d_p, d_q\}$, and our work uses bits of $\{p, q\}$ to factorize N. None of the methods could utilize the knowledge of $q^{-1} \bmod p$ to factor N. In Chapter 7, we have proved that knowing $q^{-1} \bmod p$ completely is equivalent to factoring N. But, we do not have any results if one knows some random bits of $q^{-1} \bmod p$. In the presentation of the paper [50] at Crypto 2009, this problem was also asked. In this line, let us present the following two open questions.

Problem 8.4. *Can one use some known random bits of $q^{-1} \bmod p$ to factor N?*

Problem 8.5. *Does the knowledge of random bits of $q^{-1} \bmod p$ reduce the required number of bits to be known for other private keys in case of factoring N?*

In Chapter 5, we studied the case when random bits are known from the lower half of p and q. However consider the situation when random bits are known from the upper half of p and q. Hence, we have the following question.

Problem 8.6. *Can one factor N when random bits are known from the upper half of p and q?*

8.3.4 Implicit Factorization

In PKC 2009, May and Ritzenhofen [86] introduced the problem of implicit factorization. They presents some results when few LSBs of the secret primes are same. They also extend their results for balanced RSA moduli. That is for the case $N_1 = p_1 q_1, N_2 = p_2 q_2, \ldots, N_k = p_k q_k$, with p_i and q_i are of same bitsize and few LSBs of $p_1, \ldots, p_k$ are same. In Chapter 6 and Chapter 7 we analyze the situation when $p_1, \ldots, p_k$ share their MSBs. However, we can not extend our ideas for the balanced case. Hence, we leave the following open question.

Problem 8.7. *Can one factor k balanced RSA moduli $N_1, \ldots, N_k$ in polynomial time when $p_1, \ldots, p_k$ share their MSBs?*

Final Words

This thesis discusses a variety of cryptanalytic results on RSA and its variants, mainly by the use of lattice based root finding techniques. Analysis of certain versions of factorization problem has also been studied. Efficient algorithms to find small roots of modular and bivariate integer polynomials have been exploited a lot during the last decade in connection with RSA cryptanalysis as well as integer factorization, and this proves to be a potent line of research for years to come.

Bibliography

[1] Advanced Encryption Standard. National Institute of Standards and Technology, 2001. Available at `http://csrc.nist.gov/CryptoToolkit/aes/rijndael/`.

[2] D. Aggarwal and U. M. Maurer. Breaking RSA generically is equivalent to factoring. In *Proceedings of Eurocrypt'09*, volume 5479 of *Lecture Notes in Computer Science*, pages 36–53, 2009.

[3] M. Agrawal, N. Kayal, and N. Saxena. Primes is in P. *Annals of Mathematics*, 2:781–793, 2002.

[4] M. Ajtai and C. Dwork. A public-key cryptosystem with worst-case/average-case equivalence. *Electronic Colloquium on Computational Complexity (ECCC)*, 3(65), 1996.

[5] M. Ajtai and C. Dwork. A public-key cryptosystem with worst-case/average-case equivalence. In *Proceedings of STOC'97*, pages 284–293, 1997.

[6] A. O. L. Atkin and F. Morain. Elliptic curves and primality proving. *Mathematics of Computation*, 61:29–68, 1993.

[7] A. Bauer and A. Joux. Toward a rigorous variation of Coppersmith's algorithm on three variables. In *Proceedings of Eurocrypt'07*, volume 4515 of *Lecture Notes in Computer Science*, pages 361–378, 2007.

[8] J. Blömer and A. May. New partial key exposure attacks on RSA. In *Proceedings of Crypto'03*, volume 2729 of *Lecture Notes in Computer Science*, pages 27–43, 2003.

[9] J. Blömer and A. May. A generalized Wiener attack on RSA. In *Proceedings of PKC'04*, volume 2947 of *Lecture Notes in Computer Science*, pages 1–13, 2004.

[10] J. Blömer and A. May. A tool kit for finding small roots of bivariate polynomials over the integers. In *Proceedings of Eurocrypt'05*, volume 3494 of *Lecture Notes in Computer Science*, pages 251–267, 2005.

[11] D. Boneh. Twenty years of attacks on the RSA cryptosystem. *Notices of the AMS*, 46:203–213, 1999.

[12] D. Boneh. Finding smooth integers in short intervals using CRT decoding. In *Proceedings of STOC'00*, pages 265–272, 2000.

[13] D. Boneh. Finding smooth integers in short intervals using CRT decoding. *Journal of Computer and System Sciences*, 64(4):768–784, 2002.

[14] D. Boneh and G. Durfee. Cryptanalysis of RSA with private key less than $N^{0.292}$. In *Proceedings of Eurocrypt'99*, volume 1592 of *Lecture Notes in Computer Science*, pages 1–11, 1999.

[15] D. Boneh and G. Durfee. Cryptanalysis of RSA with private key d less than $N^{0.292}$. *IEEE Transactions on Information Theory*, 46:1339–1349, 2000.

[16] D. Boneh, G. Durfee, and Y. Frankel. An attack on RSA given a small fraction of the private key bits. In *Proceedings of Asiacrypt'98*, volume 1514 of *Lecture Notes in Computer Science*, pages 25–34, 1998.

[17] D. Boneh, G. Durfee, and N. Howgrave-Graham. Factoring $n = p^r q$ for large r. In *Proceedings of Crypto'99*, volume 1666 of *Lecture Notes in Computer Science*, pages 326–337, 1999.

[18] D. Boneh and R. Venkatesan. Breaking RSA may not be equivalent to factoring. In *Proceedings of Eurocrypt'98*, volume 1403 of *Lecture Notes in Computer Science*, pages 59–71, 1998.

[19] D. R. L. Brown. Breaking RSA may be as difficult as factoring. Cryptology ePrint Archive, Report 2005/380, 2005. Available at `http://eprint.iacr.org/`.

[20] Y.-G. Chen and Y.-C. Zhu. On the prime power factorization of $n!$. *Journal of Number Theory*, 82:1–11, 2000.

[21] H. Cohen. *A Course in Computational Algebraic Number Theory.* Springer-Verlag New York, Inc., New York, NY, USA, 1993.

[22] D. Coppersmith. Finding a small root of a bivariate integer equation; factoring with high bits known. In *Proceedings of Eurocrypt'96*, volume 1070 of *Lecture Notes in Computer Science*, pages 178–189, 1996.

[23] D. Coppersmith. Finding a small root of a univariate modular equation. In *Proceedings of Eurocrypt'96*, volume 1070 of *Lecture Notes in Computer Science*, pages 155–165, 1996.

[24] D. Coppersmith. Small solutions to polynomial equations, and low exponent RSA vulnerabilities. *Journal of Cryptology*, 10(4):233–260, 1997.

[25] D. Coppersmith. Finding small solutions to small degree polynomials. In *Proceedings of CaLC'01*, volume 2146 of *Lecture Notes in Computer Science*, pages 20–31, 2001.

[26] D. Coppersmith, M. K. Franklin, J. Patarin, and M. K. Reiter. Low-exponent RSA with related messages. In *Proceedings of Eurocrypt'96*, volume 1070 of *Lecture Notes in Computer Science*, pages 1–9, 1996.

[27] T. H. Cormen, C. Stein, R. L. Rivest, and C. E. Leiserson. *Introduction to Algorithms.* McGraw-Hill Higher Education, 2001.

[28] J.-S. Coron. Finding small roots of bivariate integer polynomial equations revisited. In *Proceedings of Eurocrypt'04*, volume 3027 of *Lecture Notes in Computer Science*, pages 492–505, 2004.

[29] J.-S. Coron and A. May. Deterministic polynomial-time equivalence of computing the RSA secret key and factoring. *Journal of Cryptology*, 20(1):39–50, 2007.

[30] D. A. Cox, J. Little, and D. O'Shea. *Ideals, Varieties, and Algorithms: An Introduction to Computational Algebraic Geometry and Commutative Algebra.* Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2007.

[31] B. de Weger. Cryptanalysis of RSA with small prime difference. *Applicable Algebra in Engineering, Communication and Computing*, 13(1):17–28, 2002.

[32] Data Encryption Standard. National Institute of Standards and Technology, 1999. Available at `http://csrc.nist.gov/publications/fips/fips46-3/fips46-3.pdf`.

[33] W. Diffie and M. E. Hellman. New directions in cryptography. *IEEE Transactions on Information Theory*, IT-22(6):644–654, 1976. Available at `http://citeseer.ist.psu.edu/diffie76new.html`.

[34] P. Ekdahl and T. Johansson. A new version of the stream cipher SNOW. In *Proceedings of SAC'02*, volume 2595 of *Lecture Notes in Computer Science*, pages 47–61. Springer, 2003.

[35] T. El Gamal. A public key cryptosystem and a signature scheme based on discrete logarithms. In *Proceedings of Crypto'84*, volume 196 of *Lecture Notes in Computer Science*, pages 10–18, 1985.

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

[37] P. Erdös and C. Pomerance. On the largest prime factors of n and $n + 1$. *Aequationes Math*, 17:311–321, 1978.

[38] M. Ernst, E. Jochemsz, A. May, and B. de Weger. Partial key exposure attacks on RSA up to full size exponents. In *Proceedings of Eurocrypt'05*, volume 3494 of *Lecture Notes in Computer Science*, pages 371–386, 2005.

[39] eStream Project. Available at `http://www.ecrypt.eu.org/stream/`.

[40] J.-C. Faugère, R. Marinier, and G. Renault. Implicit factoring with shared most significant and middle bits. In *Proceedings of PKC'10*, volume 6056 of *Lecture Notes in Computer Science*, pages 70–87, 2010.

[41] K. Ford and G. Tenenbaum. The distribution of integers with at least two divisors in a short interval. *The Quarterly Journal of Mathematics*, 58:187–201, 2007.

[42] M. K. Franklin and M. K. Reiter. A linear protocol failure for RSA with exponent three. Crypto'95 Rump Session, 1995.

[43] Free Software Foundation, Inc. *GMP: The GNU Multiple Precision Arithmetic Library.* Available at `http://gmplib.org/`.

[44] S. Goldwasser and J. Kilian. Primality testing using elliptic curves. *Journal of the ACM*, 46(4):450–472, 1999.

[45] S. Goldwasser and S. Micali. Probabilistic encryption. *Journal of Computer and System Sciences*, 28:450–472, 1984.

[46] J. A. Halderman, S. D. Schoen, N. Heninger, W. Clarkson, W. Paul, J. A. Calandrino, A. J. Feldman, J. Appelbaum, and E. W. Felten. Lest we remember: cold-boot attacks on encryption keys. *Commun. ACM*, 52(5):91–98, 2009.

[47] G. H. Hardy and E. M. Wright. *An Introduction to the Theory of Numbers.* Oxford University Press, USA, 1960.

[48] J. Håstad. On using RSA with low exponent in a public key network. In *Proceedings of Crypto'85*, volume 218 of *Lecture Notes in Computer Science*, pages 403–408, 1986.

[49] J. Håstad. Solving simultaneous modular equations of low degree. *SIAM Journal on Computing*, 17(2):336–341, 1988.

[50] N. Heninger and H. Shacham. Reconstructing RSA private keys from random key bits. In *Proceedings of Crypto'09*, volume 5677 of *Lecture Notes in Computer Science*, pages 1–17, 2009.

[51] M. Herrmann and A. May. Solving linear equations modulo divisors: On factoring given any bits. In *Proceedings of Asiacrypt'08*, volume 5350 of *Lecture Notes in Computer Science*, pages 406–424, 2008.

[52] M. J. Hinek. Another look at small RSA exponents. In *Proceedings of CT-RSA'06*, volume 3860 of *Lecture Notes in Computer Science*, pages 82–98, 2006.

[53] M. J. Hinek. *On the security of some variants of RSA*. PhD thesis, University of Waterloo, Canada, 2007. Available at `http://uwspace.uwaterloo.ca/handle/10012/2988`.

[54] M. J. Hinek. *Cryptanalysis of RSA and Its Variants.* Chapman & Hall/CRC, 2009.

[55] M. J. Hinek and C. C. Y. Lam. Common modulus attacks on small private exponent RSA and some fast variants (in practice). *Journal of Mathematical Cryptology*, 4(1):58–93, 2010.

[56] M. J. Hinek and D. R. Stinson. An inequality about factors of multivariate polynomials. Technical report, Centre of Applied Cryptographic Research (CACR), University of Waterloo, 2006. Available at `http://www.cacr.math.uwaterloo.ca/techreports/2006/cacr2006-15.pdf`.

[57] K. Hoffman and R. Kunze. *Linear Algebra.* Prentice-Hall, 1971.

[58] J. Hoffstein, J. Pipher, and J. H. Silverman. NTRU: A ring-based public key cryptosystem. In *Proceedings of ANTS'98*, volume 1423 of *Lecture Notes in Computer Science*, pages 267–288, 1998.

[59] N. Howgrave-Graham. Finding small roots of univariate modular equations revisited. In *Proceedings of IMA International Conference on Cryptography and Coding*, volume 1355 of *Lecture Notes in Computer Science*, pages 131–142, 1997.

[60] N. Howgrave-Graham. *Computational Mathematics Inspired by RSA.* PhD thesis, University of Bath, 1998. Available at `http://www.nickhg.com/cgi-bin/index.html`.

[61] N. Howgrave-Graham. Approximate integer common divisors. In *Proceedings of CaLC'01*, volume 2146 of *Lecture Notes in Computer Science*, pages 51–66, 2001.

[62] N. Howgrave-Graham and J.-P. Seifert. Extending Wiener's attack in the presence of many decrypting exponents. In *Proceedings of CQRE'99*, volume 1740 of *Lecture Notes in Computer Science*, pages 153–166, 1999.

[63] K. Ireland and M. Rosen. *A Classical Introduction to Modern Number Theory.* Springer, 1990.

[64] E. Jochemsz. *Cryptanalysis of RSA variants using small roots of polynomials.* PhD thesis, Technische Universiteit Eindhoven, Netherlands, 2007. Available at `http://www.win.tue.nl/~bdeweger/studenten.html`.

[65] E. Jochemsz and A. May. A strategy for finding roots of multivariate polynomials with new applications in attacking RSA variants. In *Proceedings of Asiacrypt'06*, volume 4284 of *Lecture Notes in Computer Science*, pages 267–282, 2006.

[66] E. Jochemsz and A. May. A polynomial time attack on RSA with private CRT-exponents smaller than $N^{0.073}$. In *Proceedings of Crypto'07*, volume 4622 of *Lecture Notes in Computer Science*, pages 395–411, 2007.

[67] A. Joux, D. Naccache, and E. Thomé. When e-th roots become easier than factoring. In *Proceedings of Asiacrypt'07*, volume 4833 of *Lecture Notes in Computer Science*, pages 13–28, 2007.

[68] A. Joux and J. Stern. Lattice reduction: A toolbox for the cryptanalyst. *Journal of Cryptology*, 11(3):161–185, 1998.

[69] T. Kleinjung, K. Aoki, J. Franke, A. K. Lenstra, E. Thomé, J. W. Bos, P. Gaudry, A. Kruppa, P. L. Montgomery, D. A. Osvik, H. J. J. te Riele, A. Timofeev, and P. Zimmermann. Factorization of a 768-bit RSA modulus. In *Proceedings of Crypto'10*, volume 6223 of *Lecture Notes in Computer Science*, pages 333–350, 2010.

[70] D. E. Knuth. *The Art of Computer Programming: Seminumerical Algorithms*, volume 2. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 3 edition, 1997.

[71] N. Koblitz. Elliptic curve cryptosystems. *Mathematics of Computation*, 48:203–209, 1987.

[72] P. C. Kocher. Timing attacks on implementations of Diffie-Hellman, RSA, DSS, and other systems. In *Proceedings of Crypto'96*, volume 1109 of *Lecture Notes in Computer Science*, pages 104–113, 1996.

[73] J. C. Lagarias. The computational complexity of simultaneous Diophantine approximation problems. In *Proceedings of FOCS'82*, pages 32–39, 1982.

[74] J. C. Lagarias. The computational complexity of simultaneous Diophantine approximation problems. *SIAM Journal on Computing*, 14(1):196–209, 1985.

[75] A. K. Lenstra. Integer factoring. In *Encyclopedia of Cryptography and Security*, pages 290–297. Springer, 2005.

[76] A. K. Lenstra and H. W. Lenstra, Jr., editors. *The Development of the Number Field Sieve*, volume 1554 of *Lecture Notes in Mathematics*. Springer-Verlag, Berlin, 1993.

[77] A. K. Lenstra, H. W. Lenstra, Jr., and L. Lovász. Factoring polynomials with rational coefficients. *Mathematische Annalen*, 261:515–534, 1982.

[78] A. K. Lenstra, H. W. Lenstra, Jr., M. S. Manasse, and J. M. Pollard. The factorization of the ninth Fermat number. *Mathematics of Computation*, 61(203):319–349, 1993.

[79] H. W. Lenstra, Jr. Factoring integers with elliptic curves. *Annals of Mathematics*, 126:649–673, 1987.

[80] F. Luca and P. Stănică. On the prime power factorization of $n!$. *Journal of Number Theory*, 102/2:298–305, 2003.

[81] S. Maitra and S. Sarkar. A new class of weak encryption exponents in RSA. In *Proceedings of Indocrypt'08*, volume 5365 of *Lecture Notes in Computer Science*, pages 337–349, 2008.

[82] S. Maitra, S. Sarkar, and S. Sen Gupta. Factoring RSA modulus using prime reconstruction from random known bits. In *Proceedings of Africacrypt'10*, volume 6055 of *Lecture Notes in Computer Science*, pages 82–99, 2010.

[83] A. May. *New RSA Vulnerabilities Using Lattice Reduction Methods*. PhD thesis, University of Paderborn, Germany, 2003. Available at `http://www.cs.uni-paderborn.de/uploads/tx_sibibtex/bp.pdf`.

[84] A. May. Using LLL-reduction for solving RSA and factorization problems. Technical report, LLL+25 Conference in honour of the 25th birthday of the LLL algorithm, Technische Universität Darmstadt, 2007. Available at `http://www.informatik.tu-darmstadt.de/KP/alex.html`.

[85] A. May and M. Ritzenhofen. Solving systems of modular equations in one variable: How many RSA-encrypted messages does Eve need to know? In *Proceedings of PKC'08*, volume 4939 of *Lecture Notes in Computer Science*, pages 37–46, 2008.

[86] A. May and M. Ritzenhofen. Implicit factoring: On polynomial time factoring given only an implicit hint. In *Proceedings of PKC'09*, volume 5443 of *Lecture Notes in Computer Science*, pages 1–14, 2009.

[87] A. J. Menezes, P. C. van Oorschot, and S. A. Vanstone. *Handbook of Applied Cryptography*. CRC Press, 2001. Available at `http://www.cacr.math.uwaterloo.ca/hac/`.

[88] R. C. Merkle and M. E. Hellman. Hiding information and signatures in trapdoor knapsacks. *IEEE Transactions on Information Theory*, 24:525–530, 1978.

[89] D. Micciancio and S. Goldwasser. *Complexity of Lattice Problems: A Cryptographic Perspective*. Springer International Series in Engineering and Computer Science, 2002.

[90] G. L. Miller. Riemann's hypothesis and tests for primality. *Journal of Computer and System Sciences*, 13(3):300–317, December 1976. (Invited publication).

[91] V. S. Miller. Use of elliptic curves in cryptography. In *Proceedings of Crypto'85*, volume 218 of *Lecture Notes in Computer Science*, pages 417–426, 1986.

[92] H. Minkowski. *Geometrie der Zahlen*. Teubner Verlag, 1896.

[93] P. Q. Nguyen and D. Stehlé. Floating-point LLL revisited. In *Proceedings of Eurocrypt'05*, volume 3494 of *Lecture Notes in Computer Science*, pages 215–233, 2005.

[94] P. Q. Nguyen and J. Stern. The two faces of lattices in cryptology. In *Proceedings of CaLC'01*, volume 2146 of *Lecture Notes in Computer Science*, pages 146–180, 2001.

[95] P. Q. Nguyen and B. Vallée. *The LLL Algorithm: Survey and Applications*, chapter 2, pages 19–69. Information Security and Cryptography. Springer, 2009.

[96] A. Nitaj. Another generalization of Wiener's attack on RSA. In *Proceedings of Africacrypt'08*, volume 5023 of *Lecture Notes in Computer Science*, pages 174–190, 2008.

[97] A. Nitaj. Cryptanalysis of RSA using the ratio of the primes. In *Proceedings of Africacrypt'09*, volume 5580 of *Lecture Notes in Computer Science*, pages 98–115, 2009.

[98] T. Okamoto. Fast publickey cryptosystem using congruent polynomial equations. *Electronic letters*, 22(11):581–582, 1986.

[99] Public-Key Cryptography Standards (PKCS) #1 v2.1: RSA Cryptography Standard. RSA Security Inc., 2002. Available at `http://www.rsa.com/rsalabs/node.asp?id=2125`.

[100] C. Pomerance. The quadratic sieve factoring algorithm. In *Proceedings of Eurocrypt'84*, volume 209 of *Lecture Notes in Computer Science*, pages 169–182, 1985.

[101] C. Pomerance and H. W. Lenstra, Jr. Primality testing with Gaussian periods. Technical report, Dartmouth College, 2005. Available at `http://www.math.dartmouth.edu/~carlp/`.

[102] G. Qiao and K.-Y. Lam. RSA signature algorithm for microcontroller implementation. In *Proceedings of CARDIS'98*, volume 1820 of *Lecture Notes in Computer Science*, pages 353–356, 1998.

[103] J. J. Quisquater and C. Couvreur. Fast decipherment algorithm for RSA public-key cryptosystem. *Electronic Letters*, 18(21):905–907, 1982.

[104] M. O. Rabin. Digitalized signatures and public-key functions as intractable as factorization. Technical report, Massachusetts Institute of Technology, Cambridge, MA, USA, 1979.

[105] M. O. Rabin. Probabilistic algorithms for testing primality. *Journal of Number Theory*, 12:128–138, 1980.

[106] O. Regev. Lattices in computer science (lecture notes). Tel Aviv University, 2004. Available at `http://www.cs.tau.ac.il/~odedr/teaching/lattices_fall_2004/index.html`.

[107] M. Ritzenhofen. *On Efficiently Calculating Small Solutions of Systems of Polynomial Equations*. PhD thesis, Ruhr-University of Bochum, Germany, 2010. Available at `http://www.cits.rub.de/personen/maike.html`.

[108] R. L. Rivest. The RC4 encryption algorithm. RSA Data Security, Inc., 1992.

[109] R. L. Rivest and A. Shamir. Efficient factoring based on partial information. In *Proceedings of Eurocrypt'85*, volume 219 of *Lecture Notes in Computer Science*, pages 31–34, 1986.

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

[111] G. G. Rose and P. Hawkes. Turing: A fast stream cipher. In *Proceedings of FSE'03*, volume 2887 of *Lecture Notes in Computer Science*, pages 290–306, 2003.

[112] S. Sarkar and S. Maitra. Approximate integer common divisor problem relates to implicit factorization. To appear in IEEE Transactions on Information Theory (accepted on 12th December, 2010).

[113] S. Sarkar and S. Maitra. Further results on implicit factoring in polynomial time. *Advances in Mathematics of Communications*, 3(2):205–217, 2009.

[114] S. Sarkar and S. Maitra. Cryptanalysis of RSA with more than one decryption exponent. *Information Processing Letters*, 110(8-9):336–340, 2010.

[115] S. Sarkar and S. Maitra. Cryptanalysis of RSA with two decryption exponents. *Information Processing Letters*, 110(5):178–181, 2010.

[116] S. Sarkar and S. Maitra. Some applications of lattice based root finding techniques. *Advances in Mathematics of Communications*, 4(4):519–531, 2010.

[117] A. Shamir. A polynomial time algorithm for breaking the basic Merkle-Hellman cryptosystem. In *Proceedings of Crypto'82*, pages 279–288, 1982.

[118] A. Shamir. A polynomial time algorithm for breaking the basic Merkle-Hellman cryptosystem. In *Proceedings of FOCS'82*, pages 145–152, 1982.

[119] P. W. Shor. Algorithms for quantum computation: Discrete logarithms and factoring. In *Proceedings of FOCS'94*, pages 124–134, 1994.

[120] P. W. Shor. Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer. *SIAM Journal on Computing*, 26(5):1484–1509, 1997.

[121] C. L. Siegel. *Lectures on the Geometry of Numbers.* Springer-Verlag, 1989.

[122] S. Singh. *The Code Book: The Science of Secrecy from Ancient Egypt to Quantum Cryptography.* Anchor, 2000.

[123] R. Solovay and V. Strassen. A fast Monte-Carlo test for primality. *SIAM Journal on Computing*, 6(1):84–85, 1977.

[124] W. Stein. *Sage Mathematics Software.* Free Software Foundation, Inc., 2009. Available at `http://www.sagemath.org`. (Open source project initiated by W. Stein and contributed by many).

[125] R. Steinfeld, S. Contini, H. Wang, and J. Pieprzyk. Converse results to the Wiener attack on RSA. In *Proceedings of PKC'05*, volume 3386 of *Lecture Notes in Computer Science*, pages 184–198, 2005.

[126] D. R. Stinson. *Cryptography: Theory and Practice.* Chapman & Hall/CRC, third edition, 2005.

[127] T. Takagi. Fast RSA-type cryptosystem modulo p^kq. In *Proceedings of Crypto'98*, volume 1462 of *Lecture Notes in Computer Science*, pages 318–326, 1998.

[128] M. van Dijk, C. Gentry, S. Halevi, and V. Vaikuntanathan. Fully homomorphic encryption over the integers. In *Proceedings of Eurocrypt'10*, volume 6110 of *Lecture Notes in Computer Science*, pages 24–43, 2010.

[129] P. S. Wang and L. P. Rothschild. Factoring multivariate polynomials over the integers. *SIGSAM Bulletin*, pages 21–29, 1973.

[130] M. J. Wiener. Cryptanalysis of short RSA secret exponents. *IEEE Transactions on Information Theory*, 36(3):553–558, 1990.

[131] S. Y. Yan. *Cryptanalytic Attacks on RSA.* Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2007.

[132] P. Zimmerman. Integer factoring records. Available at `http://www.loria.fr/~zimmerma/records/factor.html`.