CRYPTANALYSIS OF RSA USING ALGEBRAIC AND LATTICE METHODS

A DISSERTATION
SUBMITTED TO THE DEPARTMENT OF COMPUTER SCIENCE
AND THE COMMITTEE ON GRADUATE STUDIES
OF STANFORD UNIVERSITY
IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR THE DEGREE OF
DOCTOR OF PHILOSOPHY

Glenn Durfee
June 2002

Preface

We study the security of public key cryptosystems. In particular we study the RSA public key cryptosystem and several variants. We obtain our results using tools from the theory of integer lattices.

We begin with an introduction to the theory of integer lattices and describe the main results to be used throughout the rest of the work. We then review the concept of a public key cryptosystem and in particular the RSA public key cryptosystem. Next we introduce a novel algorithm for the factorization of class of integers closely related to those used by RSA and other public key cryptosystems in the literature, showing that a new class of integers can be efficiently factored. We go on to introduce new attacks on the RSA public key cryptosystem which use partial knowledge of a user's secret key, showing that leaking one quarter of the bits of the secret key is sufficient to compromise RSA. Next we describe new attacks on the RSA public key cryptosystem when a short secret exponent is used. Lastly, we describe the three Sun-Yang-Laih key generation schemes for RSA, and introduce attacks to break two of these schemes.

Acknowledgments

It is impossible to thank enough everyone who has been of help over the past four years. It has been truly an honor and a pleasure to be a part of such a vibrant community of researchers, faculty, students, and friends, and I can make only a rough attempt to express my appreciation.

I want to express my great gratitude to Dan Boneh, who has been both an advisor and a friend these many years. I would also like to thank my coauthors Yair Frankel, Nick Howgrave-Graham, and Phong Nguyen; working with them has been a great pleasure.

During my time at Stanford I had the opportunity to interact with many people. I had the pleasure of working with Cynthia Dwork, Matt Franklin, John Mitchell, Moni Naor, Victor Shoup, and Ramarathnam Venkatesan. For many enlightening conversations I would like to thank Ajay Chander, Nancy Durgin, Philippe Golle, Sudipto Guha, Snorri Gylfason, Jeremy Horwitz, Ben Hsu, Piotr Indyk, Chris Jaeger, Ilya Mironov, Nagendra Modadugu, Ben Lynn, Venessa Teague, and Jon Yard.

I would like to thank my family, whose strength and support have been with me without fail throughout the years. Thank you Mom, Dad, Pam, Nan and Grandpa S., Grandpa and Grandma D.

I would like to thank my partner, Chandra Boston, for her love and support through the joys and trials of the past few years.

Finally, I would like to express my gratitude for the late Michael Muuss, with whom I worked during many high school and college summers at the U.S. Army Research Laboratory. He brought enthusiasm, wit, and wisdom to technology and used it to change the world.

Contents

List of Figures

Chapter 1

Introduction

Cryptology is the study of secret codes. In speaking of cryptology, we discuss two main branches: *cryptography* is concerned with the writing of messages in secret code and the creation of these methods, while *cryptanalysis* is concerned with reading encrypted messages by breaking secret codes.

Ancient times saw many examples of cryptography. Over three thousand years ago, Egyptian scribes made use of hieroglyphic transformations to obscure the meaning of written messages. Mesopotamian and Babylonian scribes employed similar techniques to render cuneiform tablets unreadable to the uninitiated. The Greeks employed cryptography (and the closely related *steganography*, which is concerned with concealing the existence of communication rather than content) for military and tactical purposes. Even the *Kāma-sūtra* of ancient India touches on cryptography, listing secret writing as one of the sixty-four fundamental arts. Hundreds of other examples occur in ancient civilizations; in short, cryptography has appeared more or less spontaneously in every culture in which literacy has become widespread [45].

Cryptanalysis, on the other hand, took considerably longer to develop as a rigorous subject of study. The earliest surviving descriptions of systematic methods of code-breaking come from fifteenth century, in the Arabic encyclopedia *Ṣubḥ al-a ‘sha* [45]. It gives the first known written description of the technique of *frequency analysis*, where the frequencies of letters and letter groupings of a language are used to unravel secret codes.

Cryptology has played a role in political and military matters from medieval times through the 20th century. Perhaps most famous is the cryptologic effort of Great Britain and the United States during World War II. The efforts of thirty thousand employees at Britain's Bletchley Park in cryptanalyzing Germany's Enigma transmissions is said to have shortened the war by several years [45], and it led to the development of the first digital computer, Colossus. Similar efforts in the United States to break Japanese codes aided American intelligence in the war, convincing American authorities of the importance of cryptologic expertise and eventually leading to the establishment of the National Security Agency in 1952.

With the rise of digital computer networks in the 1970s, securing communication became an increasingly important task in the private sector as well. The needs of the banking industry, in particular, for secure digital communication became especially urgent. Developments at IBM led to the design of Lucifer, one of the first public ciphers for computer communications. The National Security Agency offered several improvements to Lucifer, and in 1975 the National Bureau of Standards presented the modified version as the Data Encryption Standard, or D.E.S. Thus the first public standard for encryption began to gain widespread use.

A method of securing communication is called a *cryptosystem.* The sender *encrypts* (or *enciphers*) a message using an encryption algorithm together with a secret *key.* This produces a *ciphertext* which is sent to the recipient. The recipient, who also possesses a key, receives the ciphertext and *decrypts* (or *deciphers*) using the key to recover the original message, called the *plaintext.*

In the history of cryptology up to 1975, all cryptosystems required the sender and the receiver to agree beforehand on the *same* key, a key that had to be rigorously protected from exposure to an adversary. This is known as *symmetric* or *secret key* cryptography. Arranging to share a secret key between two parties is often a difficult problem, and does not scale well to scenarios in which many individuals or computers might communicate with each other.

In 1976, Martin Hellman, a professor at Stanford University, and Whitfield Diffie, a graduate student, introduced the concept of *asymmetric* or *public key* cryptography in their seminal paper *New Directions in Cryptography.* They speculated that a method of encryption might exist in where the encryption key differed from the decryption key. In such a scheme, a user's encryption key could be announced to the public; any outsider could obtain this public encryption key and use it to send encrypted messages to the user. Since only the user would posses the decryption key, only she could obtain the decryption of the message. Public key cryptography also opened doors for many other applications, such as digital signatures and electronic cash.

The concept of public key cryptography circulated in the research community for some time before the first practical proposal for such a scheme was made.[1] In August 1977 the RSA public key cryptosystem, named after inventors Ron Rivest, Adi Shamir, and Len Adleman, was introduced in Martin Gardner's column on Mathematical Games in *Scientific American* [32]. The RSA cryptosystem, outlined in Section 3.1, has survived over twenty years of study by cryptanalysts in the public sector, and it is the most widely used public key cryptosystem in the world. It is used, among other places, in the SET protocol [84] for secure credit card transactions and the SSL protocol [30] for secure communication on the Internet.

Public discussion and research in cryptography in business and academia started in the last quarter of the 20th century, and continues at a furious rate. New methods for encryption are publicly announced; researchers then study these methods for weaknesses by applying the tools of cryptanalysis.[2] Only after intense public scrutiny does a new cryptosystem gain a sense of legitimacy. Studying and using methods for breaking cryptosystems is an essential step in the development of new designs for more secure cryptosystems. By learning how things break, we learn how to make them stronger.

It is in this spirit that this thesis is written. This work uses mathematical tools to study the RSA public key cryptosystem and several variants. We use tools from

[1]In the public literature. It is claimed that Great Britain's GCHQ invented public key cryptography several years before the public community. [83]

[2]It might be argued that disclosing a method of encryption unnecessarily opens it to attack; that designs kept secret will be somehow inherently safer. This, however, runs counter to the historic lessons of cryptology. Cryptanalysts are notoriously famous for their aptitude in reverse engineering undisclosed cryptographic algorithms, and the leaks of designs occur all too easily. The popular wisdom is perhaps best summarized in Kerckhoffs' principle, which states that "the security of a cryptosystem should rest not in the secrecy of the algorithm, but in the secrecy of the key" [46].

numerical algebra and the geometry of numbers to get our results. Algebraic cryptanalysis has proven to be one of the most effective methods in the study of public key cryptosystems.

1.1 Overview of This Work

This thesis applies tools from the geometry of numbers to solve several problems in cryptanalysis. We use algebraic techniques to cryptanalyze several public key cryptosystems. We focus on RSA and RSA-like schemes, and use tools from the theory of integer lattices to get our results. We believe that this field is still underexplored, and that much more work can be done utilizing connections between lattices and cryptography.

Chapter 2 provides an introduction to the theory of lattices and describes several important results needed later. In particular, Sections 2.3 and 2.4 set the foundation for the rest of the work by describing a method for finding roots to polynomial congruences developed by Håstad, Coppersmith, and others.

We then provide basic facts about public key cryptography in Chapter 3, and in particular the RSA public key cryptosystem in Section 3.1. Here the basic equations are established that will be carefully analyzed in later chapters.

In Chapter 4 we use these techniques to introduce a novel algorithm for the factorization of integers. We are particularly interested in integers with repeated factors, that is, integers of the form $N = p^r q$. Such integers were previously proposed for various cryptographic applications. For sufficiently large r our algorithm runs in polynomial time (in $\log N$). Hence, we obtain a new class of integers that can be efficiently factored. When $r \approx \sqrt{\log p}$ the algorithm is asymptotically faster than the Elliptic Curve Method. These results suggest that integers of the form $N = p^r q$ should be used with care. This is especially true when r is large, namely r greater than $\sqrt{\log p}$. Our method also provides an efficient mechanism for factoring integers of the form $N = pq$ when half of the bits of one of the factors is known; our method is substantially more efficient than previous methods.

Chapter 5 studies the security of the RSA public key cryptosystem under *partial key exposure*. We show that for short public exponent RSA, given a quarter of the bits of the private key an adversary can recover the entire private key. Similar results (though not as strong) are obtained for larger values of the public exponent e. Our results point out the danger of partial key exposure in the RSA public key cryptosystem.

In Chapter 6 we show that if the secret exponent d used in the RSA public key cryptosystem is less than $N^{0.292}$, then the system is insecure. This is the first improvement over an old result of Wiener showing that when d is less than $N^{0.25}$ the RSA system is insecure.

Finally, in Chapter 7 we address the security of a variant of RSA proposed in Asiacrypt '99 by Sun, Yang, and Laih. The Sun-Yang-Laih variants of RSA resisted all known attacks, including the short secret exponent attacks covered in Chapter 6. We show that two of the three schemes they propose can be broken by very efficient attacks.

1.2 Notation

Rings and Fields

We use standard notation and write $\mathbb{Z}$ to denote the ring of integers, $\mathbb{Q}$ for the field of rationals, and $\mathbb{R}$ for the field of real numbers. When $a \in \mathbb{R}$ we denote by $a\mathbb{Z}$ the ring with base set $\{r \in \mathbb{R} \mid r = an \text{ for some } n \in \mathbb{Z}\}$.

We will also work with the group $\mathbb{Z}_N$ of integers modulo N and the group $\mathbb{Z}_N^*$ of units modulo N.

We denote by $\mathbb{Z}^n$ (resp. $\mathbb{Q}^n$ and $\mathbb{R}^n$) the vector space with elements that are n-tuples of elements of $\mathbb{Z}$ (resp. $\mathbb{Q}$ and $\mathbb{R}$). We endow these vector spaces with the Euclidean norm: if $a \in \mathbb{R}^n$, then $\|a\|^2 = \sum_i a_i^2$.

We also work with the rings of univariate polynomials $\mathbb{Z}[x]$, $\mathbb{Q}[x]$, and $\mathbb{R}[x]$, as well as multivariate polynomials in $\mathbb{Z}[x,y]$, $\mathbb{Q}[x,y]$, and $\mathbb{R}[x,y]$.

Norms of polynomials

Given a polynomial $g(x) = \sum_i a_i x^i$ we define the *norm of* $g(x)$ by

$$\|g(x)\|^2 = \sum_i a_i^2.$$

Given nonnegative $X \in \mathbb{R}$ we define the *weighted norm of* $g(x)$ by

$$\|g(xX)\|^2 = \sum_i (a_i X^i)^2.$$

Similarly, given a polynomial $h(x,y) = \sum_{i,j} b_{i,j} x^i y^j$ we define the *norm of* $h(x,y)$ by

$$\|h(x,y)\|^2 = \sum_{i,j} b_{i,j}^2.$$

Given nonnegative $X, Y \in \mathbb{R}$ we define the *weighted norm of* $h(x,y)$ by

$$\|h(xX,yY)\|^2 = \sum_{i,j} (b_{i,j} X^i Y^j)^2.$$

A univariate polynomial is said to be *monic* when the coefficient of the leading term is equal to 1.

Cardinality

When S is a finite set we use $|S|$ to denote the number of elements in S.

Exponentiation

The symbol "e" *always* refers to the public exponent used in the RSA public key cryptosystem (see Section 3.1) and *never* to the constant $2.78\cdots$. This is especially important to keep in mind when reading the many instances where e is exponentiated as e^α for various values of α.

Chapter 2

Lattice Theory

This chapter introduces several important concepts from the geometry of numbers. Of primary importance in this work is the notion of a *lattice*, defined in the next section. In the past twenty years, lattices have played an extremely important role in cryptology (see the recent survey [62] for a discussion). In this chapter we state only the results about lattices that we will need later in this work, and refer the reader elsewhere [55] for a comprehensive introduction.

2.1 Lattices

Let $u_1, \ldots, u_w$ be linearly independent vectors in an normed n-dimensional vector space. The *lattice L spanned by $\langle u_1, \ldots, u_w \rangle$* is the set of all integer linear combinations of $u_1, \ldots, u_w$. We call w the *dimension* or *rank* of the lattice L, and that the lattice is *full rank* when $w = n$. We alternatively say that L is *generated by $\langle u_1, \ldots, u_w \rangle$* and that $\langle u_1, \ldots, u_w \rangle$ *forms a basis of L*.

We work mostly with lattices in two vector spaces:

- The set $\mathbb{R}^n$ endowed with the Euclidean norm; and,
- The set of polynomials in $\mathbb{R}[x]$ of degree at most $n-1$, endowed with the norm given by $\| \sum_{i=0}^{n-1} a_i x^i \|^2 = \sum_{i=0}^{n-1} a_i^2$.

There is a natural isomorphism between these two vector spaces given by identifying a polynomial with its coefficients vector, and throughout this work switch freely between representations as convenient.

We denote by $\langle u_1^*, \ldots, u_w^* \rangle$ the vectors obtained by applying the Gram-Schmidt orthogonalization process to the vectors $u_1, \ldots, u_w$. We define the *determinant* of the lattice L as

$$\det(L) := \prod_{i=1}^{w} \|u_i^*\|.$$

For example, if L is full rank lattice with basis in $\mathbb{R}^n$, then the determinant of L is equal to the determinant of the $w \times w$ matrix whose rows are the basis vectors $u_1, \ldots, u_w$.

We note that every nontrivial lattice in $\mathbb{R}^n$ has infinitely many bases. This gives rise to the notion of the *quality* of a basis for a lattice. The notion of quality applied to a lattice depends on the application, but usually includes some measure of the lengths or orthogonality of the vectors in a basis. There are many striking examples of how good bases are useful in cryptographic applications [62], and we shall see several in subsequent chapters.

The goal of *lattice reduction* is to find good lattice bases. The theory of lattice reduction goes back to the work of Lagrange [49], Gauss [33], Hermite [38], and Korkine and Zolotarev [48], who were interested in the reduction of quadratic forms. Lattice theory was given its geometric foundation in Minkowski's famous work *Geometrie der Zahlen* (The Geometry of Numbers) [59] in the late nineteenth century.

Minkowski proved that there is always a basis $u_1, \ldots u_w$ for a lattice L satisfying $\prod \|u_i\| \leq \gamma_w \cdot \det(L)$ for some γ_w that depends only on w (and not on the entries of u_i). However, his proof is nonconstructive, and it would be almost a century before the first polynomial-time algorithm to compute reduced bases of lattices of arbitrary dimension was discovered. This is the celebrated Lenstra-Lenstra-Lovász lattice basis reduction algorithm. This algorithm is of primary importance to the results in this work, and is summarized in Section 2.2.

Representations of Lattices

In order to discuss the running times of algorithms operating on lattices we must describe the representation of lattices given as input to these algorithms. Suppose $u_1, \ldots, u_w$ are vectors in $\mathbb{Z}^n$. The running time of an algorithm with input $\langle u_1, \ldots, u_w \rangle$ is parameterized by w, n, and by the largest element of the u_i, defined by

$$u_{largest} := \max_{i,j} \; u_{ij}.$$

For lattices in $\mathbb{Q}^n$, the situation is slightly more complex. Suppose we are given the vectors $u_1, \ldots, u_w$ in $\mathbb{Q}^n$. There is some least integer D such that $Du_1, \ldots, Du_w$ are all in $\mathbb{Z}^n$. Then we define the largest element by

$$u_{largest} := \max_{i,j} \; (Du_{ij}).$$

We note that all lattices in $\mathbb{R}^n$ used in this work are in fact lattices in $\mathbb{Q}^n$ since they are represented using rational approximations.

2.2 The LLL Algorithm

Fact 2.2.1 (LLL) *Let L be a lattice spanned by $\langle u_1, \ldots, u_w \rangle$. The LLL algorithm, given $\langle u_1, \ldots, u_w \rangle$, produces a new basis $\langle b_1, \ldots, b_w \rangle$ of L satisfying:*

(1) $\|b_i^*\|^2 \;\le\; 2\|b_{i+1}^*\|^2$ *for all* $1 \le i < w$.

(2) *For all i, if* $b_i = b_i^* + \sum_{j=1}^{i-1} \mu_j b_j^*$ *then* $|\mu_j| \le \frac{1}{2}$ *for all j.*

The algorithm performs $O(w^4 \ell)$ arithmetic operations, where $\ell = \log u_{largest}$.

We note that an LLL-reduced basis satisfies some stronger properties, but those are not relevant to our discussion.

We denote by $T_{LLL}(w, n)$ the running time of the LLL algorithm on a a basis $\langle u_1, \ldots, u_w \rangle$ satisfying $\log_2 u_{largest} \le n$. When $\langle b_1, \ldots, b_w \rangle$ is the output of the LLL algorithm on a basis for a lattice L, we say that it is an *LLL-reduced basis.* We will make heavy use of the following fundamental fact about the first vector in an LLL-reduced basis.

Fact 2.2.2 *Let L be a lattice and $b_1, \ldots, b_w$ be an LLL-reduced basis of L. Then*

$$\|b_1\| \ \le\ 2^{(w-1)/4}\det(L)^{1/w}.$$

Proof Since $b_1 = b_1^*$ the bound immediately follows from:

$$\det(L) = \prod_i \|b_i^*\| \ \ge\ \|b_1\|^w \cdot 2^{-w(w-1)/4}. \qquad \blacksquare$$

In subsequent chapters we may also need to bound the length of the second vector in an LLL-reduced basis. This is provided in the following fact.

Fact 2.2.3 *Let L be a lattice and $b_1, \ldots, b_w$ be an LLL-reduced basis of L. Then*

$$\|b_2\| \ \le\ 2^{w/4}\left[\frac{\det(L)}{\|b_1\|}\right]^{1/(w-1)}.$$

Proof Observe

$$\det(L) = \prod_i \|b_i^*\| \ \ge\ \|b_1\| \cdot \|b_2^*\|^{w-1} \cdot 2^{-(w-1)(w-2)/4},$$

giving us

$$\|b_2^*\| \le 2^{(w-2)/4}\left[\frac{\det(L)}{\|b_1\|}\right]^{1/(w-1)}.$$

Then the bound follows from

$$\|b_2\| \le \|b_2^*\| + \frac{1}{2}\|b_1\| \le (1 + \frac{1}{2\sqrt{2}})\|b_2^*\| \le \sqrt{2}\|b_2^*\|. \qquad \blacksquare$$

2.3 Finding Small Solutions to Univariate Polynomial Congruences

Much of the work described in this thesis was inspired by the seminal work of Coppersmith for finding small solutions to polynomial congruences [17]. We use this very effective technique as a starting point for many of our results. In subsequent chapters we will apply and extend this technique to solve a number of cryptanalytic problems, and discuss subtleties in its implementation and use. In this section we will introduce the general Coppersmith approach and provide a few simple examples. We use a simplified version due to Howgrave-Graham [39, 40, 41].

Suppose we are given a polynomial $f(x)$ and a real number M, and we wish to find a value $x_0 \in \mathbb{Z}$ for which $f(x_0) \equiv 0 \bmod M$. The main tool we use is stated in the following simple fact. This has been attributed to many authors. Its first use we are aware of is in the work of Håstad [37]; it was later used was used implicitly by Coppersmith [17], and stated in nearly the following form by Howgrave-Graham [41].

Recall that if $h(x) = \sum a_i x^i$ then $\|h(xX)\|^2 = \sum_i (X^i a_i)^2$.

Fact 2.3.1 *Let $h(x) \in \mathbb{R}[x]$ be a polynomial of degree w, and let $X \in \mathbb{R}$ be given. Suppose there is some $|x_0| < X$ such that*

(1) *$h(x_0) \in \mathbb{Z}$, and*

(2) *$\|h(xX)\| < 1/\sqrt{w}$.*

Then $h(x_0) = 0$.

Proof Observe

$$\begin{aligned} |h(x_0)| &= \left|\sum a_i x_0^i\right| = \left|\sum a_i X^i \left(\frac{x_0}{X}\right)^i\right| \leq \\ & \sum \left|a_i X^i \left(\frac{x_0}{X}\right)^i\right| \leq \sum \left|a_i X^i\right| \leq \sqrt{w}\,\|h(xX)\| < 1, \end{aligned}$$

but since $h(x_0) \in \mathbb{Z}$ we must have $h(x_0) = 0$. ■

Fact 2.3.1 suggests we should look for a polynomial $h(x)$ of small weighted norm satisfying $h(x_0) \in \mathbb{Z}$. To do this we will build a lattice of polynomials related to f

and use LLL to look for short vectors in that lattice.

Our first observation is that $f(x_0)/M \in \mathbb{Z}$ because $f(x_0) \equiv 0 \bmod M$. Define

$$g_{i,k}(x) := x^i (f(x)/M)^k.$$

Observe that $g_{i,k}(x_0) = x_0^i (f(x_0)/M)^k \in \mathbb{Z}$ for all $i, k \geq 0$. Furthermore, this is true for all integer linear combinations of the $g_{i,k}(x)$. The idea behind the Coppersmith technique is to build a lattice L from $g_{i,k}(xX)$ and use LLL to find a short vector in this lattice. The first vector b_1 returned by the LLL algorithm will be a low-norm polynomial $h(xX)$ also satisfying $h(x_0) \in \mathbb{Z}$. If its norm is small enough, Fact 2.3.1 would imply $h(x_0) = 0$. Traditional root-finding methods [66] such as Newton-Raphson would then find x_0.

To use Fact 2.3.1 we must have $\|h(xX)\| < 1$. Fortunately, Fact 2.2.2 allows us to compute a good bound on the norm of the first vector in an LLL-reduced basis. We see

$$\det(L) < 2^{-w(w-1)/4} w^{-w/2} \qquad \Rightarrow \qquad \|h(xX)\| < 1/\sqrt{w}. \tag{2.1}$$

Usually the "error term" of $2^{-w(w-1)/4} w^{-w/2}$ is insignificant compared to $\det(L)$ and this condition is simplified as

$$\det(L) \ll 1 \qquad \Rightarrow \qquad \|h(xX)\| < 1/\sqrt{w}. \tag{2.2}$$

The determinant of L depends on the choice of polynomials $g_{i,k}(xX)$ defining the lattice. In general it is a difficult problem to compute the determinant of a lattice when the basis has symbolic entries. However, a careful choice of basis polynomials $g_{i,k}(xX)$ may lead to a lattice with a determinant that can be easily computed. Ideally, we will be able to choose a basis so that the matrix whose rows are the coefficients vectors of $g_{i,k}(xX)$ is *full-rank* and *diagonal*, with an explicit formula for the entries on the diagonal.

We illustrate this technique with a few examples.

Example 2.3.2 *A numerical example.*

Suppose we wish to find a root x_0 of the polynomial

$$x^2 - 2849x + 5324 \equiv 0 \pmod{10001} \tag{2.3}$$

satisfying $|x_0| \leq 17$. Define $f(x) := (x^2 - 2849x + 5324)/10001$. We build a lattice with basis polynomials

$$\left\{\, 1,\ 17x,\ f(17x),\ 17xf(17x),\ f^2(17x)\, \right\}.$$

Writing this as a matrix gives us

$$\begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 17 & 0 & 0 & 0 \\ 5324{\cdot}10001^{-1} & -2849{\cdot}17{\cdot}10001^{-1} & 17^2{\cdot}10001^{-1} & 0 & 0 \\ 0 & 5324{\cdot}17{\cdot}10001^{-1} & -2849{\cdot}17^2{\cdot}10001^{-1} & 17^3{\cdot}10001^{-1} & 0 \\ 28344976{\cdot}10001^{-2} & -30336152{\cdot}17{\cdot}10001^{-2} & 8127449{\cdot}17^2{\cdot}10001^{-2} & -5698{\cdot}17^3{\cdot}10001^{-2} & 17^4{\cdot}10001^{-2} \end{bmatrix}$$

The determinant of this lattice is just the product of the entries on the diagonal:

$$\det(L) = 17^{10} \cdot 10001^{-4} \approx 2.0 \cdot 10^{-4}.$$

Recall we require $\det(L) < \gamma$, where

$$\gamma = 2^{-5(5-1)/4} 5^{-5/2} \approx 5.6 \cdot 10^{-4}.$$

Hence, condition (2.1) is satisfied. We find that the LLL algorithm returns the polynomial

$$h(17x) = (-417605x^4 - 7433369x^3 + 1970691x^2 - 2625174x + 7250016)/10001^2.$$

This leads to

$$h(x) = (-5x^4 - 1513x^3 + 6819x^2 - 154422x + 7250016)/10001^2.$$

The roots of $h(x)$ over the reals are $\{16, -307.413\ldots\}$. We find the only integer solution x_0 to equation (2.3) satisfying $|x_0| \leq 17$ is $x_0 = 16$.

Example 2.3.3 *Håstad's original result.*

Suppose we are given a monic polynomial $f(x)$ of degree d with integer coefficients, along with integers X and M. We wish to find an integer x_0 such that $|x_0| < X$ and $f(x_0) \equiv 0 \bmod M$. An early attempt to solve this problem was given by Håstad [37]. We reformulate its presentation slightly for consistency with the rest of the work.

We take as a basis for our lattice L the polynomials

$$\left\{\, 1,\ Xx,\ (Xx)^2,\ \ldots,\ (Xx)^{d-1},\ f(Xx)/M \,\right\}.$$

For instance, when $d = 6$ this results in a lattice spanned by the rows of the matrix in Figure 2.1.

$$\begin{array}{rc} 1 & : \\ Xx & : \\ (Xx)^2 & : \\ (Xx)^3 & : \\ (Xx)^4 & : \\ (Xx)^5 & : \\ f(Xx)/M & : \end{array} \left[\begin{array}{ccccccc} 1 & & & & & & \\ & X & & & & & \\ & & X^2 & & & & \\ & & & X^3 & & & \\ & & & & X^4 & & \\ & & & & & X^5 & \\ - & - & - & - & - & - & X^6 M^{-1} \end{array}\right]$$

Håstad's lattice for finding small solutions to a polynomial congruence. The "–" symbols denote nonzero off-diagonal entries whose values do not affect the determinant.

Figure 2.1: Example Håstad Lattice

The dimension of this lattice is $w = d + 1$ and its determinant is

$$\det(L) = X^{w(w-1)/2} M^{-1}.$$

To satisfy condition (2.1) we require $\det(L) < 2^{-w(w-1)/4} w^{-w/2}$. This leads to

$$X < \gamma_d \cdot M^{2/d(d+1)}, \tag{2.4}$$

where $\frac{1}{2\sqrt{2}} \leq \gamma_d < \frac{1}{\sqrt{2}}$ for all d. Hence, when X satisfies bound (2.4), the LLL algorithm will find a short vector $h(xX)$ satisfying $h(x_0) = 0$ and $\|h(xX)\| < 1/\sqrt{d}$.

Standard root-finding techniques will recover x_0 from h.

The running time of this method is dominated by the time to run LLL on a lattice of dimension $d+1$ with entries of size at most $O(\log M)$.

Example 2.3.4 *Coppersmith's generic result.*

The first major improvement over Håstad's result came from Coppersmith [17]. Coppersmith suggested including powers and shifts of the original polynomial in the lattice; as we shall see, this is the reason for improved results. We use a presentation by Howgrave-Graham [40]. Again, suppose we are given a monic polynomial $f(x)$ of degree d with integer coefficients, along with integers X and M. We wish to find an integer x_0 such that $|x_0| < X$ and $f(x_0) \equiv 0 \bmod M$.

Let $m > 1$ be an integer to be determined later. Define

$$g_{i,k}(x) := x^i (f(x)/M)^k.$$

We use as a basis for our lattice L the polynomials $g_{i,k}(xX)$ for $i = 0, \ldots, d-1$ and $k = 0, \ldots, m-1$. For instance, when $d = 3$ and $m = 3$ this results in the lattice spanned by the rows of the matrix in Figure 2.2.

$$\begin{array}{ll} g_{0,0}(xX) & : \\ g_{1,0}(xX) & : \\ g_{2,0}(xX) & : \\ g_{0,1}(xX) & : \\ g_{1,1}(xX) & : \\ g_{2,1}(xX) & : \\ g_{0,2}(xX) & : \\ g_{1,2}(xX) & : \\ g_{2,2}(xX) & : \end{array} \left[\begin{array}{ccccccccc} 1 & & & & & & & & \\ & X & & & & & & & \\ & & X^2 & & & & & & \\ - & - & - & X^3M^{-1} & & & & & \\ & - & - & - & X^4M^{-1} & & & & \\ & & - & - & - & X^5M^{-1} & & & \\ - & - & - & - & - & - & X^6M^{-2} & & \\ & - & - & - & - & - & - & X^7M^{-2} & \\ & & - & - & - & - & - & - & X^8M^{-2} \end{array}\right]$$

Coppersmith's lattice for finding small solutions to a polynomial congruence. The "–" symbols denote nonzero off-diagonal entries whose values do not affect the determinant.

Figure 2.2: Example Coppersmith Lattice

The dimension of this lattice is $w = md$ and its determinant is

$$\det(L) = X^{w(w-1)/2} M^{-w(m-1)/2}.$$

We require $\det(L) < 2^{-w(w-1)/4}w^{-w/2}$; this leads to

$$X < M^{(m-1)/(w-1)}2^{-1/2}w^{-1/(w-1)}$$

which can be simplified to

$$X \;<\; \gamma_w \cdot M^{\frac{1}{d}-\epsilon},$$

where $\epsilon = \frac{d-1}{d(w-1)}$ and $\frac{1}{2\sqrt{2}} \leq \gamma_w < \frac{1}{\sqrt{2}}$ for all w. As we take $m \to \infty$ we have $w \to \infty$ and therefore $\epsilon \to 0$. In particular, if we wish to solve for up to $X < M^{\frac{1}{d}-\epsilon_0}$ for arbitrary ϵ_0, it is sufficient to take $m = O(k/d)$ where $k = \min\{1/\epsilon_0, \log M\}$.

We summarize this in the following theorem.

Theorem 2.3.5 (Univariate Coppersmith) *Let a monic polynomial $f(x)$ of degree d with integer coefficients and integers X, M be given. Suppose $X < M^{\frac{1}{d}-\epsilon}$ for some $\epsilon > 0$. There is an algorithm to find all $x_0 \in \mathbb{Z}$ satisfying $|x_0| < X$ and $f(x_0) \equiv 0$ mod M. This algorithm runs in time $O(T_{LLL}(md, m\log M))$ where $m = O(k/d)$ for $k = \min\{1/\epsilon, \log M\}$.*

We note that the generic result is in some sense "blind" to the actual polynomial being used (it takes into account only the degree, but not the coefficients), and that there may be a more optimal choice of polynomials $g_{i,k}$ to include in the lattice to solve a particular problem. In Chapter 4 we will see an example that improves over this generic result by taking into account an optimized set of polynomials.

2.4 Finding Small Solutions to Bivariate Polynomial Congruences

In this section we generalize the results of the previous section to finding solutions of bivariate polynomial congruences. We note that this is a different application of these techniques than the solution of bivariate polynomial equations (over the integers) [17]. We note that, in contrast to the previous approach, the method here is only a heuristic (for reasons that will be discussed shortly.)

In order to analyze bivariate polynomial congruences we must introduce a few observations. The first is that there is a simple generalization of Fact 2.3.1 to multivariate polynomials.

Fact 2.4.1 *Let $h(x,y) \in \mathbb{R}[x,y]$ be a polynomial which is a sum of at most w monomials, and let $X, Y \in \mathbb{R}$ be given. Suppose that*

(1) *$h(x_0,y_0) \in \mathbb{Z}$ for some $|x_0| < X$ and $|y_0| < Y$, and*

(2) *$\|h(xX,yY)\| < 1/\sqrt{w}$.*

Then $h(x_0,y_0) = 0$.

Suppose we are given a polynomial $f(x,y)$ and a real number M, and we wish to find a pair $(x_0,y_0) \in \mathbb{Z} \times \mathbb{Z}$ for which $f(x_0,y_0) \equiv 0 \bmod M$. The idea is a straightforward generalization of the approach in Section 2.3. We define

$$g_{i,j,k}(x,y) := x^i y^j (f(x,y)/M)^k,$$

and observe $g_{i,j,k}(x_0,y_0) \in \mathbb{Z}$ for all $i,j,k \geq 0$. We build a lattice from $g_{i,j,k}(xX,yY)$ by selecting certain indices (i,j,k) so that the determinant of the resulting lattice is "small enough", and compute an LLL-reduced basis for this lattice. Fact 2.2.2 bounds the norm of the first vector $h_1(xX,yY)$ of this LLL-reduced basis, allowing us to use Fact 2.4.1 to show that $h_1(x_0,y_0) = 0$.

However, a single bivariate equation may be insufficient to recover the desired root. To obtain another relation, we use the second vector $h_2(xX,yY)$ of the LLL-reduced basis. Fact 2.2.3 tells us

$$\|h_2(xX,yY)\| \leq 2^{w/4} \left[\frac{\det(L)}{\|h_1(xX,yY)\|} \right]^{1/(w-1)}. \tag{2.5}$$

So we must also provide a *lower* bound on the norm of h_1.

Suppose the indices of the $g_{i,j,k}$ are chosen so that $k \leq m$ for some m. Then all coefficients of $g_{i,j,k}(xX,yY)$ are integer multiples of M^{-m}. Thus, since $h_1(xX,yY) \neq 0$, we know it has at least one coefficient greater than or equal to M^{-m} in absolute value. So $\|h_1(xX, yY)\| \geq M^{-m}$. Equation (2.5) becomes

$$\|h_2(xX, yY)\| \leq 2^{w/4} \left(M^m \det(L)\right)^{1/(w-1)}.$$

This gives us

$$\det(L) < M^{-m} 2^{-w(w-1)/4} w^{-(w-1)/2} \qquad \Rightarrow \qquad \|h_2(xX, yY)\| < 1/\sqrt{w}. \tag{2.6}$$

In practice this condition is usually simplified as

$$\det(L) \ll M^{-m} \qquad \Rightarrow \qquad \|h_2(xX, yY)\| < 1/\sqrt{w}. \tag{2.7}$$

Hence, we obtain another polynomial $h_2(x, y) \in \mathbb{R}[x, y]$ such that $h_2(x_0, y_0) = 0$. It follows that $h_1(x, y)$ and $h_2(x, y)$ are *linearly* independent. If we make the assumption that $h_1(x, y)$ and $h_2(x, y)$ are also *algebraically* independent, we can solve for y_0 by computing the resultant $h(y) = \mathrm{Res}_x(h_1, h_2)$. Then y_0 must be a root of $h(y)$, and these roots are easily determined. From this we may find x_0 as a root of $h_1(x, y_0)$.

It is not clear why linear independence of h_1 and h_2 should imply algebraic independence, and in fact it is easy to construct (artificial) examples where this is not the case. for instance, the polynomial $x - y \equiv 0 \bmod M$ has too many solutions (even in $\mathbb{Z}^2$) thus the method must fail at this step (since all other steps are provable.) So at the moment this step of the method is only a heuristic. However, growing experimental evidence [44, 3, 9, 27] shows that it is a very good heuristic for polynomials of interest in cryptology.

Example 2.4.2 *Generic result.*

Suppose we are given a polynomial $f(x, y)$ of total degree d with at least one monic monomial $x^a y^{d-a}$ of maximum total degree. Also suppose integers X, Y, and M are given. We wish to find an integer pair (x_0, y_0) such that $|x_0| < X$, $|y_0| < Y$, and $f(x_0, y_0) \equiv 0 \bmod M$.

We will follow an approach suggested by Coppersmith [17], worked out in detail by Jutla [44]. Let $m > 1$ be an integer to be determined later. Define

$$g_{i,j,k}(x) := x^i y^j (f(x)/M)^k.$$

We use as a basis for our lattice L the polynomials $g_{i,j,k}(xX, yY)$ where the indices (i, j, k) come from the following set:

$$S := \left\{(i,j,k) \in \mathbb{Z}^3 \mid i + j + kd \leq md \text{ and } i,j,k \geq 0 \text{ and } (i < a \text{ or } j < d - a)\right\}.$$

Denote by S_m the set of polynomials $g_{i,j,k}(xX, yY)$ such that $(i, j, k) \in S$. Every $g \in S_m$ has total degree less than md. Indeed, the set S_m is in one-to-one correspondence with the set of monomials $\left\{x^\alpha y^\beta \mid \alpha + \beta \leq md\right\}$ given by $g_{i,j,k}(xX, yY) \leftrightarrow x^{i+ka} y^{j+k(d-a)}$. We may write these polynomials as the rows of a matrix in a way that puts the coefficient of the corresponding monomial on the diagonal. The resulting matrix is lower diagonal, and the contribution of $g_{i,j,k}(xX, yY) \in S_m$ to the diagonal is $M^{-k} X^{i+ka} Y^{j+k(d-a)}$. A straightforward but tedious calculation shows that the resulting lattice has determinant

$$\det(L) = \left(\prod_{\alpha=0}^{md} \prod_{\beta=0}^{md-\alpha} X^\alpha Y^\beta\right) \left(M^{-m} \prod_{k=0}^{m-1} (M^{-k})^{\frac{d}{2}((2m-2k-1)d+3)}\right).$$

For simplicity we carry out the computation using low-order terms. We find

$$\det(L) = (XY)^{\frac{d^3}{6}m^3 + o(m^3)} M^{-\frac{d^2}{6}m^3 + o(m^3)}.$$

To use condition (2.7) we require[1] $\det(L) \ll M^{-m}$. This leads to

$$XY < M^{(1/d)-\epsilon}$$

where $\epsilon \to 0$ as $m \to \infty$.

[1]We note that to use the more precise condition (2.6) requires $\det(L) < \gamma_m \cdot M^{-m}$ where $\gamma_m = 2^{-w^2/2} w^{-w/2}$ for $w = O(m^2 d^2)$. So $-\log_M \gamma_m = O(m^4 d^4)/\log M$, implying the method will not work if m is too large and M is too small. In most applications, however, we find $\log_M \gamma_m \approx 0$ and this term may be safely ignored.

We note that the shape or coefficients of a particular polynomial may allow for a better selection of basis polynomials $g_{i,j,k}$. For instance, when $f(x, y)$ has degree d in each variable separately and is monic in the monomial $x^d y^d$, a different choice of basis leads to the improved bound $XY < N^{2/3d}$. In Chapter 6 we will see an example where the shape of the polynomial allows for a substantial improvement over the generic result.

Chapter 3

Public Key Cryptography

In this chapter we present the notion of a public key cryptosystem, and in particular, the RSA public key cryptosystem. There are many good formal definitions for public key cryptosystems [34, 23, 57, 75], and we do not try to cover all of them here. Instead we try to develop the intuition that will be useful later.

A *public key* (or *asymmetric*) cryptosystem is a method for securing communication between parties who have never met before. More precisely, a public key cryptosystem is described by the following:

- a set $\mathcal{M}$ of *plaintexts* (or *messages*), and a set $\mathcal{C}$ of *ciphertexts*;
- a set $\mathcal{K}_p$ of *public keys*, and a set $\mathcal{K}_s$ of *secret keys*;
- a *key generation* algorithm $\textsf{key-gen} : \mathbb{Z} \to \mathcal{K}_p \times \mathcal{K}_s$;
- an *encryption* algorithm $E : \mathcal{K}_p \times \mathcal{M} \to \mathcal{C}$; and,
- a *decryption* algorithm $D : \mathcal{K}_s \times \mathcal{C} \to \mathcal{M}$.

The key generation, encryption, and decryption algorithms can be randomized and should run in expected time polynomial in the length of their inputs. For all $\langle K_p, K_s\rangle$ output by key-gen and all messages $M \in \mathcal{M}$ we must have that $D(K_s, E(K_p, M)) = M$.

The input to the key generation algorithm is called the *security parameter*. The hope is that as the security parameter increases, the resources required to break the cryptosystem using the resulting keys should increase more rapidly than the resources required to use it. Ideally, the running time of a break should be a (sub-)exponential function of n, while the running time of $\mathsf{key\text{-}gen}$, E, and D should be some (small) polynomial in n.

Suppose Alice is a user of a public key cryptosystem. To initialize she chooses a security parameter n and computes $\langle K_p, K_s \rangle := \mathsf{key\text{-}gen}(n)$. When another user Bob wishes to send a message to Alice securely, he obtains[1] Alice's public key K_p and computes the ciphertext $C := E(K_p, M)$. He sends ciphertext C is sent to Alice, who upon obtaining it computes the original message $M = D(K_s, C)$.

The security requirements of a cryptosystem can be defined in many ways. In general, when defining a security goal it is important to state *what resources* are available to an attacker and *what success criteria* the attacker must fulfill. A very basic requirement is that it should not be possible to derive the secret key from the public key efficiently; indeed, it is considered the most devastating cryptanalytic break to compute K_s from K_p in (say) time polynomial in the security parameter. We will see examples of this in Chapters 4, 6, and 7. We might consider security against *partial key exposure*, where information about K_s (say perhaps a subset of bits of K_s) allows an attacker to compute all of K_s; examples of this are considered in Chapter 5. There are many issues that arise in determining good notions of security, and we do not try to address them all here. There are many good surveys on the subject [57, 23].

Since the publication of *New Directions*, there have been countlessly many proposals for public key cryptosystems. Our primary focus in this work, however, will be on the RSA public key cryptosystem and simple variants [69, 77, 76]. We present the basic RSA scheme in the next section, and postpone discussion of RSA variants until the relevant chapters.

[1]Say, from a directory of public keys. Care must be taken to ensure Bob is able to obtain Alice's true public key, and not that of an impostor. There are a variety of proposals to achieve this [82].

3.1 The RSA Public Key Cryptosystem

In this section we outline the basic RSA public key cryptosystem [69].

Let n be a security parameter. The key generation algorithm for RSA computes primes p and q approximately $n/2$ bits in length, so that $N := pq$ is an integer n bits in length. More precisely, p and q are random primes subject to the constraint that $N = pq$ is a n-bit number and

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

We denote the set of all such N as $\mathbb{Z}_{(2)}$. Typically $n = 1024$, so that N is 1024 bits in length; p and q are primes typically chosen to be approximately 512 bits each.

The key generation algorithm selects integers e and d such that $ed \equiv 1$ modulo $\phi(N)$, where $\phi(N) = N - p - q + 1$ (also called the Euler totient function). We call e the *public exponent* and d the *secret exponent*. The value N is called the *public modulus*. An RSA public key is the pair of integers $\langle N, e\rangle$. The corresponding secret key is the pair $\langle N, d\rangle$. Thus $\mathcal{K}_p = \mathcal{K}_s = \mathbb{Z}_{(2)} \times \mathbb{Z}$.

How e and d are chosen depends on the application. Typically, e is chosen to satisfy certain constraints (say e is small, like $e = 3$), then d is picked from $\{1, \ldots, \phi(N)\}$ to satisfy $ed \equiv 1 \bmod \phi(N)$. However, this process may be done in reverse, and in many applications d is chosen first (say to make d short, as in Section 3.2.4 and Chapter 6).

Messages and ciphertexts are represented as elements of $\mathcal{M} = \mathcal{C} = \mathbb{Z}_N^*$. Suppose Bob wishes to send a message $M \in \mathbb{Z}_N^*$ to Alice. He obtains Alice's public key $\langle N, e\rangle$ and computes $C \equiv M^e \bmod N$ which he sends to Alice. Upon receiving C Alice may compute

$$C^d \equiv M^{ed} \equiv M \pmod{N},$$

where the last equivalence follows from Euler's Theorem.

For digital signing, the roles of these operations are reversed. If Alice intends to sign the message M she computes $S \equiv M^d \bmod N$ and sends $\langle M, S\rangle$ to Bob. Bob checks $M \overset{?}{\equiv} S^e \bmod N$.

This presentation simplifies RSA encryption and signing; in practice, randomized padding of the messages [1, 5] is required before exponentiation to prevent several security flaws [2, 21, 20]. We will not go into the details here, since all attacks in subsequent chapters succeed regardless of the padding scheme that is being used.

3.2 Previous Attacks on RSA

In this section we summarize several previously-known attacks on the RSA public key cryptosystem relevant to this work. We follow the presentation of the recent survey of attacks on RSA [6] and refer to it for a comprehensive listing of attacks on RSA.

3.2.1 Factoring

The most straightforward attack on RSA is factorization of the modulus $N = pq$. Once a factor p is discovered, the factor $q = N/p$ may be computed, so $\phi(N) = N - p - q + 1$ is revealed. This is enough to compute $d \equiv e^{-1} \bmod \phi(N)$.

The current fastest method for factoring is the General Number Field Sieve [35]. It has a running time of $\exp\big((c + o(1)) \cdot (\log N)^{1/3}(\log\log N)^{2/3}\big)$ for some $1 < c < 2$. The size of N is chosen to foil this attack. The largest integer that has been successfully factored using this method was the 512-bit RSA challenge modulus RSA-155, factored in 1999 using a massive distributed implementation of GNFS on the Internet [14]. Even though the speed of computer hardware continues to accelerate, it seems unlikely that the best factoring algorithms will be able to factor say 1024-bit RSA moduli in the next twenty years.

3.2.2 Håstad's Attack on Broadcasted Messages

In order to speed up RSA encryption (and signature verification) it is useful to use small value for the public exponent e, say $e = 3$. However, this opens up RSA to the following attack, discovered by Håstad [37].

Let us start with a simpler version. Suppose Bob wishes to send the same message M to k recipients, all of whom are using public exponent equal to 3. He obtains the public keys $\langle N_i, e_i\rangle$ for $i = 1, \ldots, k$, where $e_i = 3$ for all i. Naively, Bob computes the ciphertext $C_i = M^3 \bmod N_i$ for all i and sends C_i to the ith recipient.

A simple argument shows that as soon as $k \geq 3$, the message M is no longer secure. Suppose Eve intercepts C_1, C_2, and C_3, where $C_i = M^3 \bmod N_i$. We may assume $\gcd(N_i, N_j) = 1$ for all $i \neq j$ (otherwise, it is possible to compute a factor of one of the N_i's.) By the Chinese Remainder Theorem, she may compute $C \in \mathbb{Z}^*_{N_1N_2N_3}$ such that $C \equiv C_i \bmod N_i$. Then $C \equiv M^3 \bmod N_1N_2N_3$; however, since $M < N_i$ for all i, we have $M^3 < N_1N_2N_3$. Thus $C = M^3$ holds over the integers, and Eve can compute the cube root of C to obtain M.

Håstad proves a much stronger result. To understand it, consider the following naive defense against the above attack. Suppose Bob applies a pad to the message M prior to encrypting it so that the recipients receive slightly different messages. For instance, if M is m bits long, Bob might encrypt $i \cdot 2^m + M$ and send this to the ith recipient. Håstad proved that this linear padding scheme is not secure. In fact he showed that any fixed polynomial applied to the message will result in an insecure scheme.

Theorem 3.2.1 (Håstad) *Suppose $N_1, \ldots, N_k$ are relatively prime integers and set $N_{min} = \min_i(N_i)$. Let $g_i(x) \in \mathbb{Z}_{N_i}[x]$ be k polynomials of maximum degree d. Suppose there exists a unique $M < N_{min}$ satisfying*

$$g_i(M) = 0 \pmod{N_i} \qquad \text{for all } i \in \{0, \ldots, k\}.$$

Furthermore suppose $k > d$. There is an efficient algorithm which, given $\langle N_i, g_i(x)\rangle$ for all i, computes M.

Proof Since the N_i are relatively prime, we may use the Chinese Remainder Theorem to compute coefficients T_i satisfying $T_i \equiv 1 \bmod N_i$ and $T_i \equiv 0 \bmod N_j$ for all $i \neq j$. Setting $g(x) := \sum_i T_i g_i(x)$ we see $g(M) \equiv 0 \bmod \prod N_i$. Since the T_i are nonzero we have that $g(x)$ is not identically zero. If the leading coefficient of $g(x)$ is not one, then we may multiply by its inverse to obtain a monic polynomial $g(x)$.

The degree of $g(x)$ is at most d. By Coppersmith's Theorem (Theorem 2.3.5) we may compute all integer roots x_0 satisfying $g(x_0) \equiv 0 \bmod \prod N_i$ and $|x_0| < (\prod N_i)^{1/d}$. But we know $M < N_{min} < (\prod N_i)^{1/k} < (\prod N_i)^{1/d}$, so M is such a root. ■

This can be applied to the problem of broadcast RSA as follows. Suppose the ith plaintext is padded with a polynomial $f_i(x)$, so that $C_i \equiv (f_i(M))^{e_i} \bmod N_i$. Then the polynomials $g_i(x) := (f_i(x))^{e_i} - C_i$ satisfy the above relation. The attack succeeds once $k > \max_i(e_i \cdot \deg f_i)$.

We note that Håstad's original result was significantly weaker, requiring $k = O(d^2)$ messages where $d = \max_i(e_i \cdot \deg f_i)$. This is because the original result used the Håstad method for solving polynomial congruences (see Example 2.3.3) instead of the full Coppersmith method.

This attack suggests that *randomized* padding should be used in RSA encryption.

3.2.3 Coppersmith Attack on Short Random Pads

Like the previous attack, this attack exploits a weakness of RSA with public exponent $e = 3$. Coppersmith showed that if randomized padding is used improperly then RSA encryption is not secure [17]. Coppersmith addressed the following question: if randomized padding is used with RSA, how many bits of randomness are needed?

To motivate this question, consider the following attack. Suppose Bob sends a message M to Alice using a small random pad before encrypting. Eve obtains this and disrupts the transmission, prompting Bob to resend the message with a new random pad. The following attack shows that even though Eve does not know the random pads being used, she can still recover the message M if the random pads are too short.

For simplicity, we will assume the padding is placed in the least significant bits, so that $C_i = (2^m M + r_i)^e \bmod N$ for some small m and random $r < 2^m$. Eve now knows

$$C_1 = (2^m M + r_1)^e \pmod N \qquad \text{and} \qquad C_2 = (2^m M + r_2)^e \pmod N$$

for some unknown M, r_1, and r_2. Define $f(x, y) := x^e - C_1$ and $g(x, y) := (x+y)^e - C_2$. We see that when $x = 2^m M + r_1$, both of these polynomials have $y = r_2 - r_1$ as a root modulo N. We may compute the resultant $h(y) := \mathrm{Res}_x(f, g)$ which will be of degree at most e^2 Then $y = r_2 - r_1$ is a root of $h(y)$ modulo N. If $|r_i| < (1/2)N^{1/e^2}$ for $i = 1, 2$ then we have that $|r_2 - r_1| < N^{1/e^2}$. By Coppersmith's Theorem (Theorem 2.3.5) we may compute all of the roots $h(y)$, which will include $r_2 - r_1$. Once $r_2 - r_1$ is discovered, we may use a result of Franklin and Reiter [19] to extract M (see [6] for details).

3.2.4 Wiener's Attack on Short Secret Exponent

To speed up RSA decryption and signing, it is tempting to use a small secret exponent d rather a random $d \leq \phi(N)$. Since modular exponentiation takes time linear in $\log_2 d$, using a d that is substantially shorter than N can improve performance by a factor of 10 or more. For instance, if N is 1024 bits in length and d is 80 bits long, this results in a factor of 12 improvement while keeping d large enough to resist exhaustive search.

Unfortunately, a classic attack by Wiener [80] shows that a sufficiently short d leads to an efficient attack on the system. His method uses approximations of continued fractions. This attack is stated in the following theorem.

Theorem 3.2.2 (Wiener) *Suppose $N = pq$ and $\frac{\sqrt{N}}{2} < q < p < \sqrt{N}$. Furthermore suppose $d < \frac{1}{3}N^{1/4}$. There is an algorithm which, given N and e, generates a list of length $\log N$ of candidates for d, one of which will equal d. This algorithm runs in time linear in $\log N$.*

Proof Since $ed \equiv 1 \bmod \phi(N)$, there is some k such that $ed - k\phi(N) = 1$. We may write this as

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

Hence $\frac{e}{\phi(N)}$ is an approximation to $\frac{k}{d}$. The attacker does not know $\phi(N)$, but he does know N. Since $\frac{\sqrt{N}}{2} < q < p < 2\sqrt{N}$ we have $p + q - 1 < 3\sqrt{N}$, and thus $N - \phi(N) < 3\sqrt{N}$. Now if the attacker uses $\frac{e}{N}$ as an approximation we find

$$\left|\frac{e}{N} - \frac{k}{d}\right| = \left|\frac{ed - k\phi(N) + k\phi(N) - kN}{Nd}\right|$$

$$= \left|\frac{1-k(N-\phi(N))}{Nd}\right| \le \left|\frac{3k\sqrt{N}}{Nd}\right| = \left|\frac{3k}{d\sqrt{N}}\right|.$$

Since $e < \phi(N)$, we know $k < d < \frac{1}{3}N^{1/4}$. Thus

$$\left|\frac{e}{N} - \frac{k}{d}\right| \le \frac{1}{dN^{1/4}} < \frac{1}{2d^2}.$$

This is a classic approximation relation, and there are well-known methods [36, Thm. 177] to solve it. Such methods produce a list of all integers pairs (k_i, d_i) satisfying $\gcd(k_i, d_i) = 1$ and

$$\left|\frac{e}{N} - \frac{k_i}{d_i}\right| < \frac{1}{2d_i^2}.$$

This list is of length at most $\log N$. Since $ed - k\phi(N) = 1$ we know $\gcd(k, d) = 1$. Hence, $d = d_i$ for some $i \in \{1, \ldots, \log N\}$. ■

3.3 Cryptanalysis via the Defining Equation

The results of Chapter 5, 6, and 7 come from carefully studying the defining equation for RSA. Since $ed \equiv 1 \bmod \phi(N)$, this implies there exists an integer k such that

$$ed + k(N + 1 - (p + q)) = 1. \tag{3.1}$$

This equation succinctly summarizes the RSA, and we will refer to it frequently throughout this work.

As discussed earlier, a break of the RSA public key cryptosystem can be defined in several ways. Most obviously the scheme is broken if an attacker is able to recover the secret exponent d. Since factorization of the modulus $N = pq$ leads to recovery of the private key d, this is also a total break. All of the attacks presented in subsequent chapters are of this type, and involve either a direct computation of the private key d or one of the factors p of the public modulus N, given the public key information $\langle N, e\rangle$ alone.

In later chapters we shall see several examples where the value $s = p + q$ is computed from the public information. We note that this immediately allows the recovery of the factorization of N; indeed, when $s = p + q$, then p and q are the two roots of the equation $x^2 - sx + N = 0$.

We emphasize that our results come from the basic RSA equations; our attacks do not use plaintext/ciphertext pairs or signatures, so they hold regardless of any padding schemes used. It is an interesting open question to determine if the attacks presented in this work can be improved if a particular padding is in use, or if the adversary is given access to known or chosen plaintext/ciphertext pairs or chosen signatures.

Chapter 4

The Lattice Factoring Method

In recent years, integers of the form $N = p^r q$ have found applications in cryptography. For example, Fujioke et al. [31] use a modulus $N = p^2 q$ in an electronic cash scheme. Okamoto and Uchiyama [64] use $N = p^2 q$ for an elegant public key system. Recently Takagi [77] observed that RSA decryption can be performed significantly faster by using a modulus of the form $N = p^r q$. In all of these applications, the factors p and q are primes of approximately the same size. The security of the system relies on the difficulty of factoring N.

We show that moduli of the form $N = p^r q$ should be used with care. In particular, let p and q be integers (not necessarily prime) of a certain length, say 512 bits each. We show that factoring $N = p^r q$ becomes easier as r gets bigger. For example, when r is on the order of $\log p$, our algorithm factors N in *polynomial time*. This is a new class of integers that can be factored efficiently. This is discussed in Section 4.2.2. When $N = p^r q$ with r on the order of $\sqrt{\log p}$, our algorithm factors N *faster than the best previously-known method* — the elliptic curve method (ECM) [54]. Hence, if p and q are 512-bit primes, then $N = p^r q$ with $r \approx 23$ can be factored by our algorithm faster than with ECM. These results suggest that integers of the form $N = p^r q$ with large r are inappropriate for cryptographic purposes. In particular, Takagi's proposal [77] should not be used with a large r.

Here is a rough idea of how the algorithm's efficiency depends on the parameter r. Suppose p and q are k-bit integers and $N = p^r q$. When $r = k^\epsilon$, the our method runs (asymptotically) in time $T(k) = 2^{(k^{1-\epsilon})+O(\log k)}$. Hence, when $\epsilon = 1$, the modulus N is roughly k^2 bits long and the algorithm will factor N in polynomial time in k. When $\epsilon = \frac{1}{2}$, the algorithm asymptotically outperforms ECM. The algorithm's efficiency and its comparison with previously-known factoring methods is discussed in Section 4.4.

We ran experiments to compare our method to ECM factoring. It is most interesting to compare the algorithms when $\epsilon \approx 1/2$, namely $r \approx \sqrt{\log p}$. Unfortunately, since $N = p^r q$ rapidly becomes too large to handle, we could only experiment with small values of p. Our largest experiment involves 96-bit primes p and q and $r = 9$. In this case, N is 960 bits long. Our results suggest that although our algorithm is asymptotically superior, for such small prime factors the ECM method is better. Our experimental results are described in Section 4.3.

An additional feature of our algorithm is that it is able to make use of *partial information* about a factor. This is sometimes called *factoring with a hint.* In particular, our method gives an algorithm for factoring $N = pq$ when half of the bits of the factor p are known. This gives an elegant restatement of a theorem originally due to Coppersmith [17] for factoring with a hint using bivariate polynomial equations. In the case that $r = 1$, our presentation also coincides with a extension of Coppersmith's theorem developed by Howgrave-Graham [40]. Our version has several practical advantages, and will be an important tool used in partial key exposure attacks discussed in the next chapter. This is discussed in Section 4.2.1.

4.1 The Lattice Factoring Method

Our goal in this section is to develop an algorithm to factor integers of the form $N = p^r q$. The main theorem of this section is given below. Recall that $\exp(n) = 2^n$ and logarithms should be interpreted as logarithms to the base 2.

Theorem 4.1.1 *Let $N = p^r q$ where $q < p^c$ for some c. The factor p can be recovered from N, r, and c by an algorithm with a running time of:*

$$\exp\left(\frac{c+1}{r+c} \cdot \log p\right) \cdot O(\gamma),$$

where $\gamma = T_{LLL}(r^2, (r+c)\log N)$. The algorithm is deterministic, and runs in polynomial space.

Note that γ is polynomial in $\log N$. It is worthwhile to consider a few examples using this theorem. For simplicity, we assume $c = 1$, so that both p and q are roughly the same size. Taking c as any small constant gives similar results.

- When $c \approx 1$ we have that $\frac{c+1}{r+c} = O(\frac{1}{r})$. Hence, the larger r is, the easier the factoring problem becomes. When $r = \epsilon \log p$ for a fixed ϵ, the algorithm is polynomial time.
- When $r \approx \log^{1/2} p$, then the running time is approximately $\exp(\log^{1/2} p)$. Thus, the running time is (asymptotically) slightly better than the Elliptic Curve Method (ECM) [54].
- For small r, the algorithm runs in exponential time.
- When c is large (e.g. on the order of r) the algorithm becomes exponential time. Hence, the algorithm is most effective when p and q are approximately the same size. All cryptographic applications of $N = p^r q$ we are aware of use p and q of approximately the same size.

We prove Theorem 4.1.1 by extending the approach for finding solutions to univariate congruences developed in Section 2.3. The main tool we will need is the following slight variant of Fact 2.3.1.

Fact 4.1.2 *Let $h(x) \in \mathbb{R}[x]$ be a polynomial of degree w, and let $m \in \mathbb{Z}$ and $X \in \mathbb{R}$ be given. Suppose there is some $|x_0| < X$ such that*

(1) $h(x_0) \in q^{-m}\mathbb{Z}$, *and*

(2) $\|h(xX)\| < q^{-m}/\sqrt{w}$.

Then $h(x_0) = 0$.

Proof Apply Fact 2.3.1 to the polynomial $q^m h(x)$. ■

Note that for simplicity we assume r and c are given to the algorithm of Theorem 4.1.1. Clearly this is not essential since one can try all possible values for r and c until the correct values are found.

4.1.1 Lattice-based factoring

We are given $N = p^r q$. Suppose that in addition, we are also given an integer P that matches p on a few of p's most significant bits. In other words, $|P - p| < X$ for some large X. For now, our objective is to find p given N, r, and P. This is clearly equivalent finding the point $x_0 := P - p$.

Define the polynomial $f(x) := (P + x)^r / N$ and observe $f(x_0) = 1/q$. Let $m > 0$ be an integer to be determined later. For $k = 0, \ldots, m$ and any $i \geq 0$ define:

$$g_{i,k}(x) := x^i f^k(x).$$

Observe that $g_{i,k}(x_0) = x_0^i q^{-k} \in q^{-m}\mathbb{Z}$ for all $i \geq 0$ and all $k = 0, \ldots, m$.

Fact 4.1.2 suggests that we should look for a low-norm integer linear combination of the $g_{i,k}$ of weighted norm less than $q^{-m}/\sqrt{w}$. Let L be the lattice spanned by:

(1) $g_{i,k}(xX)$ for $k = 0, \ldots, m-1$ and $i = 0, \ldots, r-1$, and

(2) $g_{j,m}(xX)$ for $j = 0, \ldots, w - mr - 1$.

The values of m and w will be determined later. To use Fact 2.2.2, we must bound the determinant of the resulting lattice. Let M be a matrix whose rows are the coefficients vectors for the basis of L (see Figure 4.1). Notice that M is a triangular matrix, so the determinant of L is just the product of the diagonal entries of M. This is given by

$$\det(M) = \left(\prod_{k=0}^{m-1} \prod_{i=0}^{r-1} X^{rk+i} N^{-k} \right) \left(\prod_{j=mr}^{w-1} X^j N^{-m} \right) g \leq X^{w^2/2} N^{mr(m+1)/2 - wm}.$$

Fact 2.2.2 guarantees that the LLL algorithm will find a short vector $h(xX)$ in L satisfying

$$\|h(xX)\|^w \leq 2^{w^2/4} \det(L) \leq 2^{w^2/4} X^{w^2/2} N^{mr(m+1)/2 - wm}. \tag{4.1}$$

Furthermore, since $h(xX)$ is an integer linear combination of the $g_{i,k}(xX)$, the corresponding $h(x)$ as an integer linear combination of the $g_{i,k}(x)$. Therefore $h(x_0) \in q^{-m}\mathbb{Z}$.

$$\begin{array}{c|ccccccccc}
 & 1 & x & x^2 & x^3 & x^4 & x^5 & x^6 & x^7 & x^8 \\
\hline
g_{0,0}(xX) & 1 & & & & & & & & \\
g_{1,0}(xX) & & X & & & & & & & \\
g_{0,1}(xX) & - & - & X^2N^{-1} & & & & & & \\
g_{1,1}(xX) & & - & - & X^3N^{-1} & & & & & \\
g_{0,2}(xX) & - & - & - & - & X^4N^{-2} & & & & \\
g_{1,2}(xX) & & - & - & - & - & X^5N^{-2} & & & \\
g_{0,3}(xX) & - & - & - & - & - & - & X^6N^{-3} & & \\
g_{1,3}(xX) & & - & - & - & - & - & - & X^7N^{-3} & \\
g_{2,3}(xX) & & & - & - & - & - & - & - & X^8N^{-3}
\end{array}$$

Example LFM lattice for $N = p^2q$ when $m = 3$ and $d = 9$. The entries marked with "$-$" represent non-zero off-diagonal entries we may ignore.

Figure 4.1: Example LFM Lattice

To apply Fact 4.1.2 we also require that

$$\|h(xX)\| < q^{-m}/\sqrt{w}.$$

Plugging in the bound on $\|h(xX)\|$ from equation (4.1) and reordering terms, we see this condition is satisfied when:

$$(\sqrt{2}X)^{w^2/2} < q^{-wm}N^{wm}N^{-mr(m+1)/2}/(\sqrt{w})^w. \tag{4.2}$$

We may substitute $q^{-1}N = p^r$. Because $q < p^c$ for some c, we know $N < p^{c+r}$. So inequality (4.2) is satisfied when the following holds:

$$(\sqrt{2}X)^{w^2/2} < p^{wm-mr(c+r)(m+1)/2}/(\sqrt{w})^w.$$

We note that $(\sqrt{w})^{(2/w)} \leq \sqrt{2}$ for all $w \geq 4$, so this leads to

$$X < (1/2)p^{2m/w-mr(c+r)(m+1)/w^2}.$$

Larger values of X allow us to use weaker approximations P, so we wish to find the largest X satisfying the bound. The optimal value of m is attained at $m_0 = \left\lfloor \frac{w}{r+c} - \frac{1}{2} \right\rfloor$, and we may choose w_0 so that $\frac{w_0}{r+c} - \frac{1}{2}$ is within $\frac{1}{2r+c}$ of an integer. Plugging in $m = m_0$ and $w = \max\{w_0, 4\}$ and working through tedious arithmetic results in the bound:

$$X < (1/2)p^{1-\frac{c}{r+c}-\frac{r}{w}(1+\delta)} \qquad \text{where} \qquad \delta = \frac{1}{r+c} - \frac{r+c}{4w}.$$

Since $\delta < 1$ we obtain the slightly weaker, but more appealing bound:

$$X < (1/2)p^{1-\frac{c}{r+c}-2\frac{r}{w}}. \tag{4.3}$$

So when X satisfies inequality (4.3), the LLL algorithm will find a vector $h(xX)$ in L satisfying $\|h(xX)\| < q^{-m}/\sqrt{w}$. The polynomial $h(x)$ is an integer linear combination of the $g_{i,k}(x)$ and thus satisfies $h(x_0) \in q^{-m}\mathbb{Z}$. But since $\|h(xX)\|$ is bounded, we have by Fact 2.3.1 that $h(x_0) = 0$. Traditional root-finding methods [66] such as Newton-Raphson can extract the roots of $h(x)$. Given a candidate for x_0, it is easy to check if $P + x_0$ divides N. Since h is of degree $w-1$, there are at most w roots to check before x_0 is found and the factorization of N is exposed.

We summarize this result in the following lemma.

Lemma 4.1.3 *Let $N = p^r q$ be given, and assume $q < p^c$ for some c. Furthermore assume that P is an integer satisfying*

$$|P - p| < (1/2)p^{1-\frac{c}{r+c}-2\frac{r}{w}}$$

for some w. Then the factor p may be computed from N, r, c, and P by an algorithm whose running time is dominated by $T_{LLL}(w, \lfloor 2w/r \rfloor \cdot \log N)$.

Note that as w tends to infinity, the bound on P becomes $|P - p| < (1/2)p^{1-\frac{c}{r+c}}$. When $c = 1$, taking $w = r^2$ provides a similar bound and is sufficient for practical purposes. We can now complete the proof of the main theorem.

Proof of Theorem 4.1.1 Suppose $N = p^r q$ with $q < p^c$ for some c. Let $w = 2r(r+c)$. Then, by Lemma 4.1.3 we know that given an integer P satisfying

$$|P - p| < (1/2)p^{1-\frac{c+1}{r+c}}$$

the factorization of N can be found in time $T_{LLL}(w, \lfloor 2w/r \rfloor \cdot \log N)$. Let $\epsilon := \lceil \frac{c+1}{r+c} \rceil$. We proceed as follows:

(a) For all $k = 1, \ldots, \lceil (\log N)/r \rceil$ do:

(b) For all $j = 0, \ldots, 2^{\epsilon k+1}$ do: n

(c) Set $P = 2^k + j \cdot 2^{(1-\epsilon)k-1}$.

(d) Run the algorithm of Lemma 4.1.3 using the approximation P.

The outermost loop to determine the length k of p is not necessary if the size of p is known. If p is k bits long then one of the candidate values P generated in step (c) will satisfy $|P - p| < 2^{(1-\epsilon)k-1}$ and hence $|P - p| < (1/2)p^{1-\epsilon}$ as required. Hence, the algorithm will factor N in the required time. ■

4.2 Applications

4.2.1 Factoring $N = pq$ with a Hint

One immediate application of Lemma 4.1.3 is the factorization of integers of the form $N = pq$ when partial information about one of the factors is known. Suppose $(n/4) + \delta$ bits of one of factors of an n-bit value $N = pq$ is known for some small $\delta > 0$ Coppersmith showed [17] how to apply a bivariate version of his technique to solve the integer factorization problem using this partial information. The lattice factoring method described here is the first application of the univariate Coppersmith method to the problem of integer factorization with a hint. This method generalizes to integers of the form $N = p^r q$, while the bivariate method does not. Therefore it appears this technique appears to be superior to the original bivariate Coppersmith approach.

Our method has significant performance advantages over the original bivariate method of Coppersmith. Given $(n/4) + \delta$ bits of one of the factors of $N = pq$, the bivariate method of Coppersmith builds a lattice $n^2/(9\delta^2)$ with entries of size at most $n^2/(2\delta)$. Our method creates lattices of dimension n/δ with entries of size at most $2n^2/\delta$. This results in a substantial performance improvement.

Factoring $N = pq$ knowing MSBs of p

We first describe the result for $N = pq$ when most significant bits of p are known.

Corollary 4.2.1 (MSBFact) *Let $N = pq$ of binary length n be given and assume $q < p$. Furthermore assume $P > 0$ and $t > n/4$ are integers satisfying*

$$|P - p| < 2^{(n/2)-t}.$$

Then there is an algorithm that given N, P, and t computes the factor p in time $T_{LLL}(w, 2nw)$, where $w = \lceil \frac{n}{t-(n/4)} \rceil$. We denote the running time of this algorithm by $T_{\mathsf{MSBFact}}(n, t)$.

Proof In order to use Lemma 4.1.3 we must choose w such that $|P - p| < (1/2)p^{1-\frac{1}{2}-\frac{2}{w}}$. This is achieved once $|P - p| \leq 2^{\frac{n}{2}\left(\frac{1}{2}-\frac{2}{w}\right)}$, i.e., $w \geq \frac{n}{t-(n/4)}$. ■

Some comments:

- When P and p are $n/2$-bit integers such that P matches p on the t most significant bits, we have $|P - p| < 2^{(n/2)-t}$. Informally, we say that MSBFact is given the t most significant bits of p.

- Note that as t increases, $T_{\mathsf{MSBFact}}(n, t)$ decreases. While this follows from the fact that w is inversely proportional to t, it is also makes intuitive sense since the factoring problem naturally becomes easier as more bits of p are given to MSBFact.

- This algorithm can be extended to $t \leq n/4$ by running MSBFact sequentially with approximations

$$P_j := P - 2^{(n/2)-t} + (2j - 1)2^{(n/4)-1}$$

  for $j = \{1, \ldots, 2^{(n/4)-t+1}\}$. One of these P_j will satisfy $|P_j - p| < 2^{(n/4)-1}$. Hence the total running time is $2^{(n/4)-t+1} \cdot T_{\mathsf{MSBFact}}(n, (n/4) + 1)$.

- For $t = (n/4) + 1$ we have $w = n$. In practice this may result in a lattice too large to handle (e.g. factoring $n = 1024$ bit RSA moduli). The previous trick can be applied to get a running time of $2^{(n/4)-t+C} \cdot T_{\mathsf{MSBFact}}(n, (n/4) + C)$ for any $C > 0$ to get $w = \lceil n/C \rceil$.
- In most cases w can be taken to be much smaller, but the method is no longer provable.

Factoring $N = pq$ knowing LSBs of p

We now describe a slight variant of the lattice factoring method that can be used to factor $N = pq$ when least significant bits of a factor p are known.

Corollary 4.2.2 (LSBFact) *Let $N = pq$ of binary length n be given and assume $q < p$. Furthermore assume $P > 0$, $R > 0$, and $n/2 \geq t > n/4$ are integers satisfying*

$$P \equiv p \pmod{R}, \qquad R \geq 2^t.$$

Then there is an algorithm that given N, P, R, and t computes the factor p in time $T_{LLL}(w, 6nw)$, where $w = \lceil \frac{n}{t-(n/4)} \rceil$. We denote the running time of this algorithm by $T_{\mathsf{LSBFact}}(n, t)$.

Proof In this problem we are seeking to discover the value $x_0 := (P - p)/R$, where $|x_0| < 2^{n/2-t}$. We cannot apply Lemma 4.1.3 directly, so we derive the following variant. We have $\gcd(R, N) = 1$ (otherwise we know the factorization of N immediately), so we can compute a and b satisfying $aR + bN = 1$. Define the polynomial

$$f(x) := (aP + x)^r/N,$$

and observe

$$f(aRx_0) = (aP + aRx_0)^r/N = (ap)^r/N = a^r/q \in q^{-1}\mathbb{Z}.$$

But $f(aRx_0) = f(x_0 - bNx_0) = f(x_0) + C$ for some $C \in \mathbb{Z}$, so $f(x_0) \in q^{-1}\mathbb{Z}$.

This encodes the factorization problem as a univariate root-finding problem, and we use exactly the same techniques used in Section 4.1.1 to solve it. Namely, let $m > 0$ be an integer to be determined later. For $k = 0, \ldots, m$ and any $i \geq 0$ define:

$$g_{i,k} := x^i f^k(x).$$

Then $g_{i,k}(x_0) \in q^{-m}\mathbb{Z}$ for all $i \geq 0$ and all $k \leq m$. We build a lattice from these polynomials and use LLL to find a short vector. The proof follows as in Lemma 4.1.3, and we derive the same bound $|x_0| < (1/2)p^{1-\frac{c}{r+c}-2\frac{r}{w}}$. In this case $r = c = 1$, so the bound is achieved once $|x_0| \leq 2^{\frac{n}{2}\left(\frac{1}{2}-\frac{2}{w}\right)}$, i.e., $w \geq \frac{n}{t-(n/4)}$. ■

Some comments:

- When P and p are $n/2$-bit integers such that P matches p on the t least significant bits, we have $P \equiv p \bmod 2^t$. Informally, we say that $\mathsf{LSBFact}$ is given the t least significant bits of p.

- Note that as t increases, $T_{\mathsf{LSBFact}}(n, t)$ decreases. While this follows from the fact that w is inversely proportional to t, it is also makes intuitive sense since the factoring problem naturally becomes easier as more bits of p are given to $\mathsf{LSBFact}$.

- This algorithm can be extended to $t \leq n/4$ by running $\mathsf{LSBFact}$ with $R' = 2^{(n/4)-t} \cdot R$ and approximations $P_j := P + (j-1)R$ for $j = \{1, \ldots, 2^{(n/4)-t}\}$. One of these P_j will satisfy $P_j \equiv p \bmod R'$, where $R' \geq 2^{n/4}$. Hence the total running time is $2^{(n/4)-t+1} \cdot T_{\mathsf{LSBFact}}(n, (n/4) + 1)$.

- For $t = (n/4) + 1$ we have $w = n$. In practice this may result in a lattice too large to handle (e.g. factoring $n = 1024$ bit RSA moduli). The previous trick can be applied to get a running time of $2^{(n/4)-t+C} \cdot T_{\mathsf{LSBFact}}(n, (n/4) + C)$ for any $C > 0$ to get $w = \lceil n/C \rceil$.

- In most cases w can be taken to be much smaller, but the method is no longer provable. See Figure 5.1 for examples.

4.2.2 Polynomial-Time Factoring for $N = p^r q$, $r = \Omega(\log p)$

When $r \approx \log p$ the lattice factoring method runs in time polynomial in $\log N$. This is a new class of integers that can be efficiently factored. We state this formally in the following.

Corollary 4.2.3 *Let $N = p^r q$ where $q < p^c$ for some c. Suppose $r = M \log p$. The factor p can be recovered from N, r, and c by an algorithm with a running time of $2^{(c+1)/M} \cdot T_{LLL}((1/M)\log N, (r+c)\log N)$. The algorithm is deterministic, and runs in polynomial space.*

Proof This follows from Theorem 4.1.1 with the observation that

$$\left(\frac{c+1}{r+c}\right)\log p = \frac{c+1}{M} - \frac{(c+1)c}{M\log p + c} \le \frac{c+1}{M}$$

and $r^2 = (1/M)\log N$. ■

4.3 Experiments

We implemented the lattice factoring method using Maple version 5.0 and Victor Shoup's Number Theory Library package [72]. The program operates in two phases. First, it guesses the most significant bits P of the factor p, then builds the lattice described in Section 4.1. Using NTL's implementation of LLL, it reduces the lattice from Section 4.1, looking for short vectors. Second, once a short vector is found, the corresponding polynomial is passed to Maple, which computes the roots for comparison to the factorization of N.

We tested MSBFact, LSBFact, and the algorithm of Lemma 4.1.3. The algorithm of Theorem 4.1.1 uses Lemma 4.1.3 and simple exhaustive search. Examples of running times for LSBFact are given in Section 4.3. Running times for MSBFact are similar. Here we restrict attention to the core algorithm given in Lemma 4.1.3. Example running times of this algorithm are listed in Figure 4.2.

To extend this to the full version (Theorem 4.1.1) would require exhaustive search for the "bits given". This introduces a large multiplicative factor in the running times

p (bits)	N (bits)	r	bits given	lattice dim.	running time
64	576	8	16	49	14 minutes
80	1280	15	20	72	5.2 hours
96	768	7	22	60	1.6 hours
96	960	9	22	65	3.2 hours
100	600	5	23	69	1.7 hours

Experiments performed on a 1GHz Intel Pentium III running Linux.

Figure 4.2: Running times for LFM

listed above. The resulting running times are not so impressive; for such small N, ECM performs much better. However, we expect the running time to scale polynomially with the size of the input, quickly outpacing the running times of ECM and NFS, which scale much less favorably.

Optimizations

The execution times of the algorithms presented in this chapter are dominated by the running time of the LLL algorithm. In this section we address several practical concerns that greatly improve the performance of this step.

The first observation is that in our experiments, the short vector returned by the LLL algorithm almost always corresponds to a polynomial of degree $w - 1$. This means that a linear combination which yields a short vector will include those basis vectors corresponding to the $g_{i,k}$ and $g_{j,k}$ of greatest degree. We focus attention of the LLL algorithm on these basis vectors by using the following ordering on the basis vectors:

- $g_{j,m}(xX)$ for $j = w - mr - 1, \ldots, 0$, followed by
- $g_{i,k}(xX)$ for $k = m, m-1, \ldots, 0$ and $i = 0, \ldots, r-1$.

This resulted in a speedup of over factor of two compared to the natural ordering, in which LLL spent a large amount of time reducing a portion of the basis that would ultimately be irrelevant.

The second observation is that in an LLL-reduced lattice, the worst-case result for the shortest vector will be

$$\|b_1\| \approx 2^{(w-1)/4} \det(L)^{1/w}.$$

Implementations of LLL often try to improve this by reducing the "fudge factor" of $2^{(w-1)/4}$. However, as the analysis from Section 4.1 shows, the final contribution of this term is negligible. Thus a high-quality basis reduction is unnecessary, and running times can be greatly improved by deactivating features such as Block Korkin-Zolotarev reduction.

4.4 Comparison to Other Factoring Methods

We restate the Theorem 4.1.1 so that it is easier to compare lattice factoring to existing algorithms. We first introduce some notation. Let $T_\alpha(p)$ be the function defined by:

$$T_\alpha(p) = \exp\left((\log p)^\alpha\right)$$

This function is analogous to the $L_{\alpha,\beta}(p)$ function commonly used to describe the running time of factoring algorithms [51]. Recall that

$$L_{\alpha,\beta}(p) = \exp\left(\beta(\log p)^\alpha(\log\log p)^{1-\alpha}\right)$$

One can easily see that $T_\alpha(p)$ is slightly smaller than $L_{\alpha,1}(p)$. We can now state a special case of Theorem 4.1.1.

Corollary 4.4.1 *Let $N = p^r q$ be given where p and q are both k bit integers. Suppose $r = (\log p)^\epsilon$ for some ϵ. Then given N and r, a non-trivial integer factor of N can be found in time*

$$\gamma \cdot T_{1-\epsilon}(p) = \exp\left[(\log p)^{1-\epsilon}\right] \cdot \gamma$$

where γ is polynomial in $\log N$.

Asymptotic Comparison

Let p, q be k-bit primes, and suppose we are given $N = p^r q$. We study the running time of various algorithms with respect to k and r, and analyze their behaviors as r goes to infinity. We write $r = (\log p)^\epsilon$. The standard running times [16, 50] of several algorithms are summarized in the following table, ignoring polynomial factors.

Method	Asymptotic running time
Lattice Factoring Method	$\exp\left((\log p)^{1-\epsilon}\right)$
Elliptic Curve Method	$\exp\left(1.414 \cdot (\log p)^{1/2} (\log\log p)^{1/2}\right)$
Number Field Sieve	$\exp\left(1.902 \cdot (\log N)^{1/3} (\log\log N)^{2/3}\right)$

Since $N = p^r q$ and $r = k^\epsilon$, we know that

$$\log N = r \log p + \log q \geq rk = k^{1+\epsilon}.$$

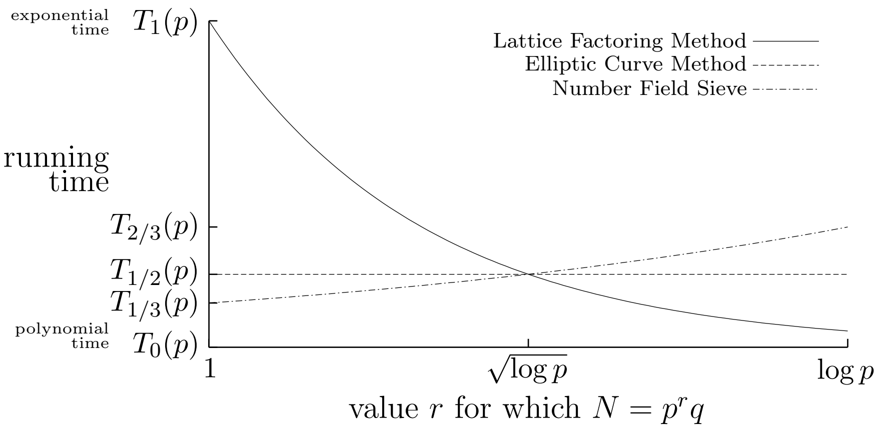
Rewriting the above running times in terms of k yields the following list of asymptotic running times.

Method	Asymptotic running time
Lattice Factoring Method	$\exp\left(k^{1-\epsilon}\right) = T_{1-\epsilon}(p)$
Elliptic Curve Method	$\exp\left(1.414 \cdot k^{1/2} (\log k)^{1/2}\right) > (T_{1/2}(p))^{1.414}$
Number Field Sieve	$\exp\left(1.902 \cdot k^{(1+\epsilon)/3} ((1+\epsilon)\log k)^{2/3}\right)$
	$> \ (T_{(1+\epsilon)/3}(p))^{1.902}$

We are particularly interested in the exponential component of the running times, which is tracked in Figure 4.3. Notice that when $\epsilon = \frac{1}{2}$, then all three algorithms run in time close to $T_{1/2}(p)$.

Practical Comparison to ECM

Of particular interest in Figure 4.3 is the point at $r = \sqrt{\log p}$ (i.e. $\epsilon = \frac{1}{2}$), where ECM, LFM, and NFS have similar asymptotic running times. We refer the reader to



Comparison of subexponential running times of current factoring methods as a function of r. Both axes are logarithmic, and polynomial time factors are suppressed.

Figure 4.3: Asymptotic comparison of LFM with ECM and NFS

Figure 4.2 for the sample running times with the lattice factoring method on similar inputs.

Since some of the larger integers that we are attempting to factor exceed 1000 bits, it is unlikely that current implementations of the Number Field Sieve will perform efficiently. This leaves only the Elliptic Curve Method for a practical comparison. Below, we reproduce a table of some example running times [81, 74] for factorizations performed by ECM.

size of p	running time with $r = 1$	predicted run time for large r	
64 bits	53 seconds	$r = 8$:	848 seconds
96 bits	2 hours	$r = 9$:	50 hours
128 bits	231 hours	$r = 10$:	7000 hours

Clearly, the Elliptic Curve Method easily beats the lattice factoring method for small integers. However, LFM scales polynomially while ECM scales exponentially. Based on the two tables above we conjecture that the point at which our method will be faster than ECM in practice is for $N = p^r q$ where p and q are somewhere around 400 bits and $r \approx 20$.

4.5 Conclusions

We showed that for cryptographic applications, integers of the form $N = p^r q$ should be used with care. In particular, we showed that the problem of factoring such N becomes easier as r get bigger. For example, when $r = \epsilon \log p$ for a fixed constant $\epsilon > 0$ the integer N can be factored in polynomial time. Hence, if p and q are k bit primes, the integer $N = p^k q$ can be factored by a polynomial time algorithm. Even when $r \approx \sqrt{\log p}$ such integers can be factored in time that is asymptotically faster than the best current methods. For much smaller r, our algorithm provides an efficient method for factoring N provided a "hint" of appropriate quality is available. This is of important practical significance in the next chapter.

Our experiments show that when the factors p and q are small (e.g. under 100 bits) the algorithm is impractical and cannot compete with the ECM. However, the algorithm scales better; we conjecture that as soon as p and q exceed 400 bits each, it performs better than ECM when r is sufficiently large.

Surprisingly, our results do not seem to follow directly from Coppersmith's results on finding small roots of bivariate polynomials over the integers. Instead, we extend the univariate root-finding technique. It is instructive to compare our results to the case of unbalanced RSA where $N = pq$ is the product of two primes of different size, say p is much larger than q. Suppose p is a prime on the order of q^s. Then, the larger s is, the *more* bits of q are needed to efficiently factor N. In contrast, we showed that when $N = p^r q$, the larger r is, the *fewer* bits of p are needed.

One drawback of the lattice factoring method is that for each guess of the most significant bits of p, the LLL algorithm has to be used to reduce the resulting lattice. It is an interesting open problem to devise a method that will enable us to run LLL once and test multiple guesses for the MSBs of p. This will significantly improve the algorithm's running time. A solution will be analogous to techniques that enable one to try multiple elliptic curves at once in the ECM. Another question is to generalize the LFM to integers of the form $N = p^r q^s$ where r and s are approximately the same size.

Chapter 5

Partial Key Exposure Attacks

Let $N = pq$ be an RSA modulus and let e, d be encryption/decryption exponents, i.e., $ed \equiv 1 \bmod \phi(N)$. In this chapter we study the following question: how many bits of d does an adversary require in order to reconstruct all of d? Surprisingly, we show that for short public exponent RSA, given only a quarter of the least significant bits of d, an adversary can efficiently recover all of d. We obtain similar results, summarized below, for larger values of e as well. Our results show that the RSA public key cryptosystem, and particularly low public exponent RSA, are vulnerable to *partial key exposure*. We refer to this class of attacks as *partial key exposure attacks*.

To motivate this problem consider a computer system which has an RSA private key stored on it. An adversary may attempt to attack the system in a variety of ways in order to obtain the private key. Some attacks (e.g. a timing attack [47]) are able to reveal some bits of the key, but may fail to reveal the entire key [24]. Our results show that attacks, such as the timing attack on RSA, need only be carried out until a quarter of the least significant bits of d are exposed. Once these bits are revealed the adversary can efficiently compute all of d. Another scenario where partial key exposure comes up is in the presence of covert channels. Such channels are often slow or have a bounded capacity. Our results show that as long as a fraction of the private exponent bits can be leaked, the remaining bits can be reconstructed.

Let $N = pq$ be an n-bit RSA modulus. The reader is reminded that we view the private exponent d as an n-bit string, so that when referring to the t most significant bits of d we refer to the t leftmost bits of d when viewed canonically as an n-bit string. For instance, it is possible that the t most significant bits of d are all zero, for some t. Similarly, a quarter of the bits of d always refers to $n/4$ bits.

5.1 Summary of Partial Key Exposure Attacks on RSA

We summarize our results in the following two theorems. The proofs are given in the subsequent sections. The first theorem applies to short public exponent RSA. The second applies to larger values of e. Throughout this chapter we assume $N = pq$ is an RSA modulus with $\sqrt{N}/2 < q < p < 2\sqrt{N}$.

Theorem 5.1.1 *Let $N = pq$ be an n-bit RSA modulus with $N = 3 \bmod 4$. Let $1 \le e, d \le \phi(N)$ satisfy $ed \equiv 1 \bmod \phi(N)$ and $e < 2^{(n/4)-3}$. There is an algorithm that given N, e, and the $n/4$ least significant bits of d computes all of d in time polynomial in n and e.*

As we shall see, the running time of the attack algorithm in the above theorem is in fact *linear* in $e \log_2 e$. Consequently, as long as e is not "too large" the attack can be efficiently mounted. For a very small value of e such as $e = 3$ we will show in Section 5.4 that the attack runs in a reasonable amount of time. For larger values, such as $e = 65537$, the attack is still feasible, though clearly takes much longer.

Theorem 5.1.2 *Let $N = pq$ be an n-bit RSA modulus with $N = 3 \bmod 4$. Let $1 \le e, d \le \phi(N)$ satisfy $ed \equiv 1 \bmod \phi(N)$.*

1. *Suppose e is a prime in the range $\{2^t, \ldots, 2^{t+1}\}$ with $n/4 \le t \le n/2$. Then there is an algorithm that given N, e, and the t most significant bits of d computes all of d in time polynomial in n.*

2. *More generally, suppose $e \in \{2^t, \ldots, 2^{t+1}\}$ is the product of at most r distinct primes $\langle e_1, \ldots, e_r \rangle$ with $n/4 \le t \le n/2$. Then there is an algorithm that given N, $\langle e_1, \ldots, e_r \rangle$, and the t most significant bits of d computes all of d in time polynomial in n and 2^r.*

3. *When the factorization of e is unknown, we obtain a weaker result. Suppose e is in the range $\{2^t, \ldots, 2^{t+1}\}$ with $t \in 0 \ldots n/2$. Further, suppose $d > \epsilon N$ for some $\epsilon > 0$. Then there is an algorithm that given N, e, ϵ, and the $n - t$ most significant bits of d computes all of d in time polynomial in n and $1/\epsilon$.*

Theorem 5.1.2 applies to public exponents e in the range $2^{n/4} \leq e \leq 2^{n/2}$. Unlike the previous theorem, Theorem 5.1.2 makes use of the *most* significant bits of d. When e is prime, at most half the bits of d are required to mount the attack. Fewer bits are needed when e is smaller. Indeed, if e is close to $N^{1/4}$ only a quarter of the MSB bits of d are required. The same result holds when e is not prime, as long as we are given the factorization of e and e does not have too many distinct prime factors. The last part of the theorem applies to $e < N^{1/2}$ when the factorization of e is not known. To mount the attack, at least half the MSB bits of d are required. More bits are necessary, the smaller e is. The attack algorithm works for most e, but may fail if d is significantly smaller than N.

One may refine Theorem 5.1.2 in many ways. It is possible to obtain other results along these lines for public exponents $e < N^{1/2}$. For instance, consider the case when the factorization of e is unknown. If the adversary is given half the most significant bits of d and a quarter of the least significant bits then we show the adversary can recover all of d. When $e < N^{1/4}$ this is better than the results of Theorem 5.1.2 part (3). However, we view attacks that require non-consecutive bits of d as artificial. We briefly sketch these variants in Section 5.3.3.

There has been recent work on protecting against partial key exposure using *exposure resilient cryptography.* Initial progress was made several years ago by Chor et al. [15], who develop the notion of *t-resilient functions* and discuss an application to protection against partially leaked cryptographic keys. More recent work has focused on *exposure resilient functions*, introduced by Canetti et al. [13], which can be used to protect against partial key exposure for a variety of cryptographic primitives. These ideas were later refined by Dodis [25] and Dodis, Sahai, and Smith [26]. Steinfeld and Zheng [73] have shown that by carefully choosing p and q (instead of selecting them at random as is typical in practice) the short public exponent partial key exposure attacks presented in this chapter can be prevented.

5.1.1 Notation

Throughout this chapter we let $N = pq$ denote an n-bit RSA modulus. We assume the primes p and q are distinct and close to $\sqrt{N}$. More precisely, we assume

$$4 < \sqrt{N}/2 < q < p < 2\sqrt{N} \tag{5.1}$$

Notice that equation (5.1) implies $p + q < 3\sqrt{N}$. Furthermore, it follows by equation (5.1) that

$$N/2 < N - 4\sqrt{N} < \phi(N) < N, \tag{5.2}$$

where $\phi(N) = N - p - q + 1$ is the Euler totient function.

Let $1 \leq e, d \leq \phi(N)$ be RSA encryption/decryption exponents. Recall the defining equation for RSA:

$$ed + k(N + 1 - (p + q)) = ed + k\phi(N) = 1. \tag{3.1}$$

For convenience, throughout this chapter we set $s := p + q$, and write equation (3.1) as

$$ed + k(N + 1 - s) = 1. \tag{5.4}$$

Under the assumption $p > q$ this implies:

$$p = \frac{1}{2}(s + \sqrt{s^2 - 4N}). \tag{5.5}$$

Since $\phi(N) > d$ we know that $k < e$.

5.2 Partial Key Exposure Attacks on Short Public Exponent RSA

In this section we consider attacks on the RSA public key cryptosystem with a "short" public exponent e. For our purposes, "short" implies that exhaustive search on all values less than e is feasible. In particular, since $k \leq e$ holds, our attack algorithm can exhaustively search all possible candidates k' for k.

Suppose we are given the least-significant m-bit block of d for some $m \geq n/4$. That is, we know some d_0 such that $d_0 \equiv d \bmod 2^m$. Reducing equation (3.1) modulo 2^m and substituting $q = N/p$, we see that $p \bmod 2^m$ is a solution for x in the equation

$$ed_0 \equiv 1 + k(N - x - N/x + 1) \pmod{2^m}.$$

This leads to the following quadratic equation in x:

$$kx^2 + (ed_0 - k(N+1) - 1)x + kN \equiv 0 \pmod{2^m}. \tag{5.6}$$

Initially, an attacker might have only a guess k' for k, so it must obtain solutions to the equation

$$k'x^2 + (ed_0 - k'(N+1) - 1)x + k'N \equiv 0 \pmod{2^m}. \tag{5.7}$$

The attack will focus on finding solutions to this equation. The main tool we use is stated in the following lemma.

Lemma 5.2.1 *With the notation as in Section 5.1.1, suppose $N \equiv 3 \bmod 4$ and $e \leq 2^{(n/4)-3}$, and k' and m are integers. There is an algorithm that given N, e, k', and m computes a list of solutions $\langle x_1, \ldots, x_\ell \rangle$ to equation (5.7) satisfying:*

(1) *If $k' = k$, then all solutions to equation (5.7) are included in the list, and*

(2) *Let $t_{k'}$ be the largest integer such that $2^{t_{k'}}$ divides k'. Then the length of the list satisfies $\ell \leq 2^{2+t_{k'}}$ and the algorithm runs in time $O(n^3 2^{t_{k'}})$.*

Proof Suppose $k' = k$. Notice that k divides the coefficients of the quadratic term and the constant term of equation (5.6); to see that k divides the coefficient of the linear term, observe $ed_0 - k(N+1) - 1 = -k(p+q)$.

Suppose $k = 2^{t_k}w$ for some integer t_k and odd integer w. Then every solution x to equation (5.6) satisfies

$$wx^2 - w(p+q)x + wN \equiv 0 \pmod{2^{m-t_k}}.$$

Furthermore, w is odd so $w^{-1} \bmod 2^{m-t_k}$ is well-defined, thus every such solution x satisfies

$$x^2 - (p+q)x + N \equiv 0 \pmod{2^{m-t_k}}. \tag{5.8}$$

We may complete the square to obtain[1]

$$(x - (p+q)/2)^2 \equiv ((p-q)/2)^2 \pmod{2^{m-t_k}}. \tag{5.9}$$

D. Redmond [68] describes an algorithm to find a solutions to equations of the form $y^2 \equiv c \bmod 2^u$, when $c \equiv 1 \bmod 8$ and $u \geq 3$. Since $N \equiv 3 \bmod 4$, we have $p \not\equiv q \bmod 4$, and it follows that $((p-q)/2)^2 \equiv 1 \bmod 8$. Furthermore, $t_k \leq \log_2 e \leq (n/4) - 3$ and $m \geq n/4$, so $m - t_k \geq 3$. Thus equation (5.9) is in precisely the required form. For completeness, we summarize this algorithm in Lemma A.1.1 in Appendix A.

The above produces one solution to equation (5.8). Lemma A.1.3 in Appendix A shows that there are exactly four solutions to this equation and that given one of these solutions the other three are easy to obtain. Call these four solutions $\{x_1, x_2, x_3, x_4\}$. It follows that the solutions to equation (5.6) are $x_i + j \cdot 2^{(n/4)+\delta-t_k}$ for $i \in \{1,2,3,4\}$ and $j \in \{0, \ldots, 2^{t_k} - 1\}$.

In summary, when $k' = k$, we expect equation (5.7) to have at most 2^{2+t_k} solutions. The above process will find them all, so condition **(1)** is satisfied. If more than 2^{2+t_k} solutions are discovered, we know $k' \neq k$, so we may safely terminate after outputting the first 2^{2+t_k} solutions without violating condition **(1)**. Hence, condition **(2)** is satisfied. ■

We are now ready prove Theorem 5.1.1, restated more precisely as follows. Recall that LSBFact is the algorithm of Corollary 4.2.2.

Theorem 5.2.2 *With the notation as in Section 5.1.1, suppose* $N \equiv 3 \bmod 4$ *and* $e \leq 2^{(n/4)-3}$. *There is an algorithm that given* N, e, *and the* $m \geq n/4$ *least significant bits of* d *factors* N *in time* $O(T_{\mathsf{LSBFact}}(n,m) \cdot e \log_2 e)$.

Proof The attack works as follows: for every candidate value k' for k in the range $\{1, \ldots, e\}$ do:

[1] We note that the fractions in this expression represent division over the integers, which is well-defined since $p \equiv q \bmod 2$.

1. Use the algorithm of Lemma 5.2.1 to compute all solutions for $x \bmod 2^m$ to equation (5.7).
2. For each solution $x_0 \bmod 2^m$ so obtained use $\mathsf{LSBFact}$ to attempt to factor N. Since one of these solutions will satisfy $x_0 \equiv p \bmod 2^m$ the attack will succeed in factoring N.

From Lemma 5.2.1 we may conclude that when k' is odd, there are at most four solutions to equation (5.7) to check; if $k' \equiv 2 \bmod 4$, then there are at most eight solutions to check, and so on. Since k' is exhaustively searched in the range $\{0, \ldots, e\}$, this implies that at most $4e\lceil \log_2 e \rceil$ solutions to equation (5.7) will be tested before the correct value of $x \equiv p \bmod 2^m$ is obtained, exposing the factorization of N. Each solution is tested using $\mathsf{LSBFact}$; since $T_{\mathsf{LSBFact}}(n, m)$ dominates the running time of the other arithmetic operations (including root-finding and use of Lemma A.1.1), this leads to a total running time that is $O(T_{\mathsf{LSBFact}}(n, m) \cdot e \log_2 e)$. ■

Remark 1. The case $N \equiv 1 \bmod 4$ requires careful analysis, and R. Steinfeld and Y. Zheng show that a modification of the above attack has an expected running time of $O(T_{\mathsf{LSBFact}}(n, m) \cdot n \cdot e \log_2 e)$ over random choice of $n/2$-bit odd integers p and q [73]. They go on to describe how to resist this partial key exposure attack by forcing $p \equiv q \bmod 2^u$ for large values of u.

One may wonder whether a similar partial key exposure attack is possible using the most significant bits of d. The answer is no. The reason is that low public exponent RSA leaks half the most significant bits of d. In other words, the adversary may obtain *half* the most significant bits of d from e and N alone. Consequently, revealing the most significant bits of d does not help the adversary in exposing the rest of d. This is stated more precisely in the following fact.

Fact 5.2.3 *With the notation as in Section 5.1.1, suppose there exists an algorithm $\mathcal{A}$ that given the $n/2$ most significant bits of d discovers all of d in time $t(n)$. Then there exists an algorithm $\mathcal{B}$ that breaks RSA in time $et(n)$.*

Proof Observe that by equation (5.4), we have $d = (1 + k(N + 1 - p - q))/e$. Let

$\tilde{d}$ be

$$\tilde{d} = \left\lfloor \frac{1 + k(N+1)}{e} \right\rfloor$$

Then

$$0 \leq \tilde{d} - d \leq k(p+q)/e \leq 3k\sqrt{N}/e < 3\sqrt{N}$$

It follows that $\tilde{d}$ matches d on the $n/2$ most significant bits of d. Hence, once k is known, the half most significant bits of d are exposed. With this observation, algorithm $\mathcal{B}$ can work as follows: try all possible values k' for k in the range $\{1, \ldots, e\}$. For each candidate k', compute the corresponding value $\tilde{d}$ using k' as a guess for k. Run algorithm $\mathcal{A}$ giving it half the most significant bits of $\tilde{d}$. Once the $k' = k$, the entire private key is exposed. ■

Remark 2. Fact 5.2.3 demonstrates that computing the exponentiation associated with the half high-order bits of d can be performed by an untrusted server. This may be of use in server-aided RSA systems where techniques using the Chinese Remainder Theorem cannot be employed. For instance, in threshold RSA [29], the factorization of the modulus is not known to any party, so Chinese Remainder techniques are of no help in accelerating computation. Fact 3.2 suggests that the half high-order bits of d can be revealed to the server with no additional compromise in security, allowing accelerated computation by off-loading that portion of the exponentiation operation.

Fact 5.2.3 explains why for low exponent RSA one cannot mount a partial key recovery attack given the most significant bits of d. It is natural to ask whether one can expose all of d given a quarter of the low order bits of d that are not necessarily the least significant ones. For instance, the following attack uses $n/4$ bits of d in bit positions $n/4$ to $n/2$, outlined below as Theorem 5.2.4. Is it possible to generalize this result to *every* subsequence of $n/4$ bits from the $n/2$ low-order bits of d? At the moment this is an open question.

Theorem 5.2.4 *With the notation as in Section 5.1.1, suppose* $|p-q| \geq 2^{(n/2)-2}$ *and* $e \leq 2^{(n/4)-3}$*, There is an algorithm that given* N*,* e*, and the* $m \geq n/4$ *bits of* d *in positions* $(n/2) - m$ *to* $n/2$ *factors* N *in time* $O(T_{\mathsf{LSBFact}}(n, m) \cdot e^2)$*.*

Proof Using the approximation $\tilde{d}$ developed in Fact 5.2.3 and small exhaustive search, we may obtain bits $n/2$ to n of d. We now know bits $(n/2) - m$ to n of d; we denote this by d_1, so that $|d - d_1| < 2^{(n/2)-m}$. The attack tries every candidate value k' for k in the range $\{1, \dots, e\}$. Define

$$s_1 := N + 1 - \frac{ed_1 - 1}{k'}.$$

Let us consider the case $k' = k$. We have

$$|s_1 - s| = \left|\frac{e}{k}(d - d_1)\right| < 2^{(n/2)-m+\log_2 e}.$$

Equation (5.5) suggests we take

$$p_1 := \frac{1}{2}(s_1 + \sqrt{s_1^2 - 4N}),$$

as an approximation to p. It turns out that this is a very good approximation, so that

$$|p_1 - p| \leq 2^{(n/2)-m+5+\log_2 e}.$$

This is proved as Lemma B.1.1 in Appendix B. So in at most $32e$ steps, exhaustive search on the high-order bits of p_1 will discover p_1^* such that $|p_1^* - p| \leq 2^{(n/2)-m}$. Each candidate for p_1^* is given to $\mathsf{LSBFact}$ to attempt to factor N.

In summary, once $k' = k$, the attack algorithm will discover the factorization of N. The testing of each candidate k' runs in time $32e \cdot T_{\mathsf{LSBFact}}(n, m)$; there are at most e candidates to test, so the total running time is $O(T_{\mathsf{LSBFact}}(n, m) \cdot e^2)$. ■

5.3 Partial Key Exposure Attacks on Medium Exponent RSA

We describe several attacks on the RSA system that can be employed when the public key e is in the range $2^{n/4}$ to $2^{n/2}$. Unlike the previous section, these attacks require the *most* significant bits of d to be given. We mount the attack by carefully studying equation (5.4):

$$ed + k(N + 1 - s) = 1$$

Recall that $s = p + q$.

The key to mounting these attacks is in finding k. Searching for k by brute force is infeasible, since k is an arbitrary element in the range $\{1, \dots, e\}$. Fortunately, given sufficiently many MSB's of d, we may compute k directly, eliminating it as an unknown from equation (5.4). Once k is revealed, we are left with two unknowns, d and s which we recover using various methods. The main tool for discovering k is presented in the following theorem. It shows that as long as $e < \sqrt{N}$ we can find k given only $\log_2 e$ MSB bits of d. The theorem produces a small constant size interval containing k. As always, we try all possible values of k in the interval until our attack algorithm succeeds.

Theorem 5.3.1 *With the notation as in Section 5.1.1, let t be an integer in the range $\{0, \dots, n/2\}$. Suppose $2^t < e < 2^{t+1}$. Then there is an algorithm that given N, e, and the t most significant bits of d efficiently computes the compute the unique k satisfying equation (5.4) up to a constant additive error.*

The proof of Theorem 5.3.1 relies on the following lemma, which provides general conditions under which k can be deduced by rounding.

Lemma 5.3.2 *Suppose d_0 is given such that the following two conditions hold:*

- **(i)** $|e(d - d_0)| < c_1 N$, *and*
- **(ii)** $ed_0 < c_2 N^{3/2}$.

Then the unique k satisfying $ed + k\phi(N) = 1$ is an integer in the range $[\tilde{k} - \Delta, \tilde{k} + \Delta]$ where $\tilde{k} = (ed_0 - 1)/N$ and $\Delta = 8c_2 + 2c_1$.

Proof Let $\tilde{k} = (ed_0 - 1)/N$. Then

$$\left|\tilde{k} - k\right| = \left|(ed_0 - 1)\left(\frac{1}{\phi(N)} - \frac{1}{N}\right) + \frac{e(d - d_0)}{\phi(N)}\right| < c_2 N^{3/2}\left(\frac{N - \phi(N)}{\phi(N) N}\right) + c_1 \frac{N}{\phi(N)}$$

Since $N - \phi(N) < 4\sqrt{N}$ and $\phi(N) > N/2$ it follows that

$$\left|\tilde{k} - k\right| < 8c_2 + 2c_1.$$

Consequently, k is an integer in the range $\{\tilde{k} - \Delta, \ldots, \tilde{k} + \Delta\}$ as required. ■

We are now prepared to prove Theorem 5.3.1.

Proof of Theorem 5.3.1

The t most significant bits of d enable us to construct an integer d_0 satisfying $|d - d_0| < 2^{n-t}$. We use Lemma 5.3.2 to compute k. By the restriction on e, condition **(i)** is satisfied with $c_1 = 2$. Since $d_0 < N$, condition **(ii)** holds with $c_2 = 2$. Hence k is an integer in a known interval of width 40. ■

5.3.1 Prime Public Exponent

We are now ready to prove part (1) of Theorem 5.1.2. Theorem 5.3.1 enables us to find k. Once k is found we reduce equation (5.4) modulo e. This removes d from the equation. We can then solve for $s \bmod e$. Given $s \bmod e$ we are able to factor the modulus.

Theorem 5.3.3 *With the notation of Section 5.1.1, let t be an integer in the range $n/4 \leq t \leq n/2$. Suppose e is a prime in the range $\{2^t, \ldots, 2^{t+1}\}$. Then there is an algorithm that given N, e, and the t most significant bits of d factors N in time $O(T_{\mathsf{LSBFact}}(n,t))$.*

Proof The assumptions of the theorem satisfy the conditions of Theorem 5.3.1. Consequently, k is known to be an integer in a constant size range. We try all candidate values k' for k. For each one we do the following:

1. Compute $s' \equiv k'^{-1} - N - 1 \pmod{e}$. This is well-defined since $\gcd(e, k') = 1$.

2. Compute a root $p' \bmod e$ for x in the quadratic

$$x^2 - s'x + N \equiv 0 \pmod{e} \tag{5.10}$$

This can be done efficiently (in probabilistic polynomial time) since e is prime. Indeed, once $s' \equiv s \equiv p + q \bmod e$, then $p' \equiv p \bmod e$.

3. Use $\mathsf{LSBFact}$ to find p given $p \bmod e$. This is possible since $t \geq 2^{n/4}$.

Once the correct candidate $k' = k$ is found (after a constant number of attempts) the factorization of N is exposed. Each attempt runs $\mathsf{LSBFact}$ with the hint $p \bmod e$ (equal to approximately t bits of information about p). This leads to a total running time that is $O(T_{\mathsf{LSBFact}}(n,t))$. ∎

A surprising consequence of this theorem is that, when e is prime and is roughly $\cong 2^{n/4}$, only the first $n/4$ MSB's of d are needed to mount the attack. This attack is as strong as the one on low public exponent RSA. In any case, for prime $e \in \{2^{n/4}, \ldots, 2^{n/2}\}$, the first $n/2$ most significant bits of d always suffice.

The proof shows that it is not necessary for e to be prime. As long as we can solve the quadratic in step (2) the proof can be made to work. In order to solve the quadratic we must be given the factorization of e. Unfortunately, modulo a composite, the quadratic may have many roots. We must try them all. If e has r distinct prime factors, there are at most 2^r solutions to consider. As a result, we must also bound the number of prime factors of e. We obtain part (2) of Theorem 5.1.2.

Corollary 5.3.4 *As in Theorem 5.3.3 suppose e is an integer in* $\{2^t, \ldots, 2^{t+1}\}$. *Suppose* $e = \prod_{i=1}^{r} e_i$ *where* $e_1, \ldots, e_r$ *are distinct primes. Then there is an algorithm that given* N, $\langle e_1, \ldots, e_r \rangle$, *and the* t *most significant bits of* d *factors* N *in time* $O(T_{\mathsf{LSBFact}}(n,t) \cdot 2^r)$.

We point out that when e is close to $2^{n/2}$ the same attack can be mounted even if the factorization of e is unknown. In other words, for all e sufficiently close to $2^{n/2}$, half the MSB's of d are sufficient to reconstruct all of d. Indeed, the range $\{1, \ldots, \lceil 2^{n/2+2}/e \rceil\}$ can be searched exhaustively to find $\lceil s/e \rceil$. Given the value of $\lceil s/e \rceil$ we can obtain s (since $s \bmod e$ is already known.) Since s is now known in the integers, we can directly find p using equation (5.5).

5.3.2 Public Exponent with Unknown Factorization

We now turn to proving part (3) of Theorem 5.1.2. We consider the case when e is in the range $\{2^t, \ldots, 2^{t+1}\}$ with $0 \leq t \leq n/2$. The factorization of e is unknown. The following result establishes that we can still find all of d, given some of its MSB's. Our attack works as long as k is not significantly smaller than e. At the end of the section we note that the attack heuristically works for almost all e in the range $\{2^t, \ldots, 2^{t+1}\}$.

Theorem 5.3.5 *With the notation as in Section 5.1.1, let t be an integer in the range $\{0, \ldots, n/2\}$. Suppose e is in the range $\{2^t, \ldots, 2^{t+1}\}$. Further suppose $k > \epsilon \cdot e$ for some $\epsilon > 0$. Then there is an algorithm that given N, e, and the $n-t$ most significant bits of d finds all of d. The algorithm runs in time $O(n^3/\epsilon)$.*

Proof Given the $n-t$ most significant bits of d we can construct a d_0 such that $0 \leq d - d_0 < 2^t$. Since $e < 2^{n/2}$ we can use d_0 and Theorem 5.3.1 to limit k to a constant size interval. For each candidate k' for k we do the following:

1. Compute $d_1 \equiv e^{-1} \bmod k'$. This is possible since e and k are relatively prime, so only candidates k' relatively prime to e are worth considering. Since $ed - k\phi(n) = 1$ we know that $d_1 \equiv d \bmod k'$ when $k' = k$.

2. By assumption $k' > \epsilon 2^t$. Note that at this point we know $d \bmod k'$ as well as the $n-t$ MSB's of d. We determine the rest of the bits by an exhaustive search. More precisely, write
$$d = k'd_2 + d_1$$
Then $d_2 = d_0/k' + (d - d_0)/k' - d_1/k'$. The only unknown term in this sum is $v = (d - d_0)/k'$. Since $k' > \epsilon 2^t$ we know that $v = (d - d_0)/k' < 1/\epsilon$. To find v, we try all possible candidates v' in the range $\{0, \ldots, 1/\epsilon\}$. For each pair (k', v') of candidates we compute the corresponding value of d and test it.

3. Once the correct values $v' = v$ and $k' = k$ are found, d is exposed. Testing each pair (k', v') of candidates takes $O(n^3)$ time, and there are $O(1/\epsilon)$ candidate pairs to test. ∎

Theorem 5.3.5 works without having to know the factorization of e. Unfortunately, the results are not as strong as in the previous section. When e is close to $N^{1/4}$, Theorem 5.3.5 implies that 3/4 of the bits of d are needed to reconstruct d. This is much worse than the corresponding bound achieved in the previous section, where only 1/4 the bits were required. When e is close to $N^{1/2}$ the theorem produces results similar to the previous section.

Theorem 5.3.5 can only be applied when $k > \epsilon \cdot e$. Intuitively, k behaves roughly as a random integer in the range $\{1, \ldots, e\}$. As such, we should have $k > e/10$ for about 90% of the $e \in \{2^t, \ldots, 2^{t+1}\}$. Hence, heuristically the attack works efficiently for almost all e.

5.3.3 Further Results

What if the factorization of e is unknown and e was not randomly chosen? Although it may be computationally infeasible, it is possible for e, d to be specifically chosen as factors of $1 + k\phi(N)$ for very small k, violating the conditions of Theorem 5.3.5. We stress that this is particularly unlikely, as not only would the rather large value of $1+k\phi(N)$ would need to be factored into ed, but a factor e in the range $\{2^{n/4}, \ldots, 2^{n/2}\}$ would need to be obtained, and one that itself cannot easily be factored (making it vulnerable to Corollary 5.3.4). However, under these circumstances, the above attacks would not apply. We conclude with the following general result which holds for all $e < 2^{n/2}$. Unfortunately, the result requires non-consecutive bits of d.

Theorem 5.3.6 *With the notation as in Section 5.1.1, let t be an integer in $\{1, \ldots, n/2\}$ and e in $\{2^t, \ldots, 2^{t+1}\}$. There is an algorithm that given N, e, the t most significant bits of d, and the $m \geq n/4$ least significant bits of d factors N in time $O(T_{\mathsf{LSBFact}}(n, m))$.*

Proof Sketch Using Theorem 5.3.1 we may compute a constant size interval I containing k. Observe that the proof of Theorem 5.2.2 applies for all e, as long as k and the m least significant bits of d are known. To recover d, run the algorithm of Theorem 5.2.2 on all candidate values of k in I. ■

length of N	bits of d exposed	LFM lattice dim.	$T_{\mathsf{LSBFact}}(n,m)$	max runtime
768 bits	208 bits	9	0.5 seconds	3.5 seconds
768 bits	198 bits	23	69 seconds	8 minutes
1024 bits	285 bits	9	0.9 seconds	6.5 seconds
1024 bits	264 bits	35	40 minutes	4.7 hours

Experiments performed on a 1GHz Intel Pentium III running Linux.

Figure 5.1: Running times of LSBFact used in partial key exposure attacks

5.4 Experiments

To test the effectiveness of our attacks, we implemented them using the C++ programming language with the Number Theory Library package by V. Shoup [72]. The running time of LSBFact dominates the running time of our attacks (all other steps are efficient arithmetic operations), so its running time is the primary object of discussion in this section.

We experimented with partial key exposure attacks with $e = 3$. We note that when $e = 3$, we must have $k = 2$. Since k is known, we may eliminate the outer loop in the algorithm of Theorem 5.2.2, so at most eight solutions to equation (5.7) are computed. Only seven of these need to be checked, since at least one of them will expose either p or q. Hence, the running time of the attack when $e = 3$ is at most $7 \cdot T_{\mathsf{LSBFact}}(n,m)$.

Some experiments with $e = 3$ are summarized in Figure 5.1.

For medium public exponent e, where e is prime, the attack proceeds similarly: with the right guess for k, the value $p \bmod e$ can be determined. Thus, the running time of an attack on medium, prime public exponent e will be equal to $T_{\mathsf{LSBFact}}(n, \log_2 e)$ times the number of candidates for k that must be checked, which is always less than or equal to 40. With e of known factorization and r factors, there is a multiplicative increase of 2^r in running time to compute and check the distinct solutions to equation (5.10).

When the factors of e are unknown and the $(n - \log_2 e)$ bits of d are given, the attack is immediate – the algorithm suggested in Theorem 5.3.5 involves only a few simple arithmetic operations and a very small search.

5.5 Conclusions

We study the RSA public key cryptosystem's vulnerability to partial key exposure. We showed that for low exponent RSA, a quarter of the least significant bits of d are sufficient for efficiently reconstructing all of d. We obtain similar results for larger values of e as long as $e < \sqrt{N}$. For instance, when e is close to $\sqrt{N}$, half the most significant bits of d suffice.

The results presented in this chapter demonstrate the danger of leaking a fraction of the bits of d. We note that discrete log schemes (e.g. DSS, ElGamal) do not seem vulnerable to partial key exposure. A fraction of the bits of the private key in a discrete log based system does not seem to enable the adversary to efficiently break the system.

There are a number of related open problems:

- Does there exist a polynomial time algorithm which enables one to break the RSA system for values of e substantially larger than $\sqrt{N}$, given only a subset of the bits of d?

- Our strongest result with respect to the fewest bits of d required to break the system is for an e in the range $\{N^{1/4}, \ldots, N^{1/2}\}$ with known factorization. For an e with unknown factorization and in the same range, does there exist a polynomial time algorithm which provides as strong a result?

Chapter 6

Attacks on Short Secret Exponent RSA

In this chapter, we show that if the secret exponent d used in the RSA public key cryptosystem is less than $N^{0.292}$, then the system is insecure. This is the first improvement over an old result of Wiener showing that when d is less than $N^{0.25}$ the RSA system is insecure.

To motivate this problem, consider the problem of speeding up RSA signature generation. The most computationally expensive step in signature generation is the operation $m \mapsto m^d \bmod N$. Thus one is tempted to use a short secret exponent d. Unfortunately, Wiener [80] showed over ten years ago that if one uses $d < N^{0.25}$ then the RSA system can be broken. (See Section 3.2.4.) Since then there have been no improvements to this bound. Verheul and Tilborg [79] showed that as long as $d < N^{0.5}$ it is possible to expose d in less time than an exhaustive search; however, their algorithm requires exponential time as soon as $d > N^{0.25}$.

In this chapter we describe the first substantial improvement to Wiener's result. We show that as long as $d < N^{0.292}$ one can efficiently break the system. In particular, when $d < N^{0.292}$ an attacker can recover the short secret key given the public key.

Wiener describes a number of clever techniques for avoiding his attack while still providing fast RSA signature generation. One such suggestion is to use a large value of e. Indeed, Wiener's attack provides no information as soon as $e > N^{1.5}$. In contrast, our approach is effective as long as $e < N^{1.875}$. Consequently, larger values of e must be used to defeat the attack. We discuss this variant in Section 6.4.

6.1 Overview: The Small Inverse Problem

Our main results will come from carefully examining the defining equation for RSA:

$$ed + k(N + 1 - (p + q)) = 1. \tag{3.1}$$

Writing $s := -(p + q)$ and $A := N + 1$, equation (3.1) can be simplified to:

$$k(A + s) \equiv 1 \pmod{e}.$$

Throughout the rest of the chapter we write $e = N^{\alpha}$ for some α. Typically, e is of the same order of magnitude as N (e.g. $e > N/10$) and therefore α is very close to 1. As we shall see, when α is much smaller than 1, our results become even stronger.

Suppose the secret exponent d satisfies $d < N^{\delta}$. Our goal is to find the largest possible δ for which the value of d can be recovered efficiently from e and N. By equation (3.1) we know that

$$|k| < \frac{de}{\phi(N)} \leq 3de/(2N) < (3/2)e^{1+\frac{\delta-1}{\alpha}}.$$

Similarly, since both p and q are less than $2\sqrt{N}$ we know that

$$|s| < 3N^{0.5} = 3e^{1/(2\alpha)}.$$

To summarize, taking $\alpha \approx 1$ (which is the common case) and ignoring small constants, we end up with the following problem: find integers k and s satisfying

$$k(A + s) \equiv 1 \pmod{e} \quad \text{where} \quad |s| < e^{0.5} \text{ and } |k| < e^{\delta}. \tag{6.2}$$

The problem can be viewed as follows: given an integer A, find an element "close" to

A whose inverse modulo e is "small". We refer to this as the *small inverse problem.* Clearly, if for a given value of δ one can efficiently list all the solutions to the small inverse problem, then RSA with secret exponent smaller than N^δ is insecure: simply observe that s modulo e reveals all of s since $s < e$; once $s = -(p+q)$ is revealed, it is straightforward to recover p and q since $N = pq$ is also known. We will show how to solve the small inverse problem whenever $\delta < 1 - \frac{1}{2}\sqrt{2} \approx 0.292$.

A first pass at a solution to the small inverse problem when α is close to 1 is given in Section 6.2. In Section 6.3, we improve this approach and prove the main result of the chapter. Section 6.4 provides a solution for arbitrary α. Finally, Section 6.5 describes experimental results with the attack algorithm.

6.2 Solving the Small Inverse Problem

In this section we focus on the case when e is of the same order of magnitude as N, i.e. if $e = N^\alpha$ then α is close to 1. To simplify the exposition, in this section we simply take $\alpha = 1$. In Section 6.4 we give the general solution for arbitrary α. When $\alpha = 1$ the small inverse problem is the following: given a polynomial $f(x,y) = (x(A+y)-1)/e$, find (x_0, y_0) satisfying

$$f(x_0, y_0) \in \mathbb{Z} \quad \text{where} \quad |x_0| < e^\delta \ \text{ and } \ |y_0| < e^{0.5}.$$

We show that the problem can be solved whenever $\delta < 1 - \frac{1}{2}\sqrt{2} \approx 0.292$. We begin by giving an algorithm that works when $\delta < \frac{7}{6} - \frac{1}{3}\sqrt{7} \approx 0.284$. Our approach follows that outlined in Section 2.4, but we make several improvements specific to the small inverse problem. For simplicity, let $X = e^\delta$ and $Y = e^{0.5}$.

The main tool we use is Fact 2.4.1, which suggests that we look for a polynomial $h(x,y)$ of small norm satisfying $h(x_0, y_0) \in \mathbb{Z}$. For integers $i, j, k \geq 0$ we define:

$$g_{i,k}(x,y) := x^i f^k(x,y) \quad \text{and} \quad h_{j,k}(x,y) := y^j f^k(x,y).$$

We refer to the $g_{i,k}$ polynomials as x-shifts and the $h_{j,k}$ polynomials as y-shifts. Observe that $g_{i,k}(x_0, y_0) \in \mathbb{Z}$ and $h_{j,k}(x_0, y_0) \in \mathbb{Z}$ for all $i, j, k \geq 0$. We are interested in finding a low-norm integer linear combination of the polynomials $g_{i,k}(xX, yY)$ and $h_{j,k}(xX, yY)$. To do so we form a lattice spanned by these polynomials. Our goal is to build a lattice that has sufficiently small vectors and then use LLL to find them. By Fact 2.2.2 we must show that the lattice spanned by the polynomials has a sufficiently small determinant.

Given an integer m, we build a lattice L spanned by the polynomials for $k = 0, \ldots, m$. For each k we use $g_{i,k}(xX, yY)$ for $i = 0, \ldots, m-k$ and use $h_{j,k}(xX, yY)$ for $j = 0, \ldots, t$ for some parameter t that will be determined later. For example, when $m = 3$ and $t = 1$ the lattice is spanned by the rows of the matrix in Figure 6.1. Since

$$
\begin{array}{r|c|cc|ccc|cccc||cccc|}
 & 1 & x & xy & x^2 & x^2y & x^2y^2 & x^3 & x^3y & x^3y^2 & x^3y^3 & y & xy^2 & x^2y^3 & x^3y^4 \\
\hline
1 & 1 & & & & & & & & & & & & & \\
\hline
x & & X & & & & & & & & & & & & \\
f & - & - & e^{-1}XY & & & & & & & & & & & \\
\hline
x^2 & & & & X^2 & & & & & & & & & & \\
xf & & - & & - & e^{-1}X^2Y & & & & & & & & & \\
f^2 & - & - & - & - & - & e^{-2}X^2Y^2 & & & & & & & & \\
\hline
x^3 & & & & & & & X^3 & & & & & & & \\
x^2f & & & & - & & & - & e^{-1}X^3Y & & & & & & \\
xf^2 & & - & & - & - & & - & - & e^{-2}X^3Y^2 & & & & & \\
f^3 & - & - & - & - & - & - & - & - & - & e^{-3}X^3Y^3 & & & & \\
\hline\hline
y & & & & & & & & & & & Y & & & \\
yf & & & - & & & & & & & & - & e^{-1}XY^2 & & \\
yf^2 & & & - & & - & - & & & & & - & - & e^{-2}X^2Y^3 & \\
yf^3 & & & - & & & - & & - & - & - & - & - & - & e^{-3}X^3Y^4 \\
\hline
\end{array}
$$

The matrix spanned by $g_{i,k}$ and $h_{j,k}$ for $k = 0, \ldots, 3$, $i = 0, \ldots, 3-k$, and $j = 0, 1$. The "–" symbols denote non-zero entries whose value we do not care about.

Figure 6.1: Example lattice used in attacks on short secret exponent RSA

the lattice is spanned by a lower triangular matrix, its determinant is only affected by entries on the diagonal, which we give explicitly. Each "block" of rows corresponds to a certain power of x. The last block is the result of the y-shifts. In the example in Figure 6.1, $t = 1$, so only linear shifts of y are given. As we shall see, the y-shifts are the main reason for our improved results.

We now turn to calculating the determinant of the lattice L. A routine calculation shows that the determinant of the submatrix corresponding to all x shifts (i.e. ignoring the y-shifts by taking $t = 0$) is

$$\det_x = e^{-m(m+1)(m+2)/6} \cdot X^{m(m+1)(m+2)/3} \cdot Y^{m(m+1)(m+2)/6}.$$

For example, when $m = 3$ the determinant of the submatrix excluding the bottom block is $e^{-10}X^{20}Y^{10}$. Plugging in $X = e^{\delta}$ and $Y = e^{0.5}$ we obtain

$$\det_x = e^{m(m+1)(m+2)(4\delta-1)/12} = e^{\frac{4\delta-1}{12}m^3+o(m^3)}.$$

For us to be able to use Fact 2.3.1, we must have (roughly) $\det_x < 1$, implying $(4\delta - 1) < 0$. We obtain $\delta < 0.25$. This is exactly Wiener's result. It turns out that any lattice formed by taking only the x-shifts cannot be used to improve on Wiener's result.

To improve on Wiener's result we include the y-shifts into the calculation. For a given value of m and t, the product of the elements on the diagonal of the submatrix corresponding to the y-shifts is:

$$\det_y = e^{-tm(m+1)/2} \cdot X^{tm(m+1)/2} \cdot Y^{t(m+1)(m+t+1)/2}.$$

Plugging in the values of X and Y, we obtain:

$$\det_y = e^{tm(m+1)(\delta-1)/2+t(m+1)(m+t+1)/4} = e^{\frac{2\delta-1}{4}tm^2+\frac{1}{4}mt^2+o(tm^2)}.$$

The determinant of the entire matrix is $\det(L) = \det_x \cdot \det_y$.

We intend to apply Fact 2.3.1 to the shortest vectors in the LLL-reduced basis of L. To do so, we must ensure that the norm of b_1 is less than $1/\sqrt{w}$. Combining this with Fact 2.2.2, we must solve for the largest value of δ satisfying

$$\det(L) < 1/\gamma,$$

where $\gamma = 2^{w^2/4} w^{w/2}$. Since the dimension w is only a function of δ (but not of the public exponent e), γ is a fixed constant. Manipulating the expressions for the determinant and the dimension to solve for δ requires tedious arithmetic. We provide the exact solution in Appendix C. Here, we carry out the computation ignoring low order terms. That is, we write

$$\det(L) = e^{\frac{4\delta-1}{12}m^3 + \frac{2\delta-1}{4}tm^2 + \frac{1}{4}mt^2 + o(m^3)}, \qquad \log_e(\gamma) \approx 0.$$

To satisfy $\det(L) < 1/\gamma$ we must have

$$\frac{4\delta-1}{12}m^3 + \frac{2\delta-1}{4}tm^2 + \frac{1}{4}mt^2 < 0.$$

For every m the left hand side is minimized at $t = \frac{m(1-2\delta)}{2}$. Plugging this value in leads to:

$$m^2\left[-1 + 4\delta - \frac{3}{2}(1-2\delta)^2 + \frac{3}{4}(1-2\delta)^2\right] < 0,$$

implying $-7 + 28\delta - 12\delta^2 < 0$. Hence,

$$\delta < \frac{7}{6} - \frac{1}{3}\sqrt{7} \approx 0.284.$$

Hence, for large enough m, whenever $d < N^{0.284-\epsilon}$ for any fixed $\epsilon > 0$ we can find a polynomial $H_1(x,y) \in \mathbb{R}[x,y]$ such that $H_1(x_0,y_0) = 0$.

One relation is not enough, so to obtain another we use Fact 2.2.3 to bound the norm of the second vector of the LLL-reduced basis. Because every coefficient of $g_{i,k}(xX,yY)$ and $h_{j,k}(xX,yY)$ is a multiple of e^{-m}, we know $\|H_1(xX,yY)\| \geq e^{-m}$. Combining this with Fact 2.2.3 gives us the bound

$$\det(L) < e^{-m}/\gamma' \tag{6.3}$$

where $\gamma' = 2^{w(w-1)/4} w^{(w-1)/2}$. For large enough m, this inequality is guaranteed to hold, since the modifications only effect low order terms. Hence, we obtain another polynomial $H_2(x,y) \in \mathbb{R}[x,y]$ linearly independent of $H_1(x,y)$ such that

$H_2(x_0, y_0) = 0$. If it turns out that $H_2(x, y)$ is algebraically independent, we solve for y_0 by computing the resultant $H(y) = \mathrm{Res}_x(H_1, H_2)$. Then y_0 must be a root of $H(y)$. The roots of $H(y)$ are easily determined, and one such root will expose $y_0 = -(p+q)$, allowing us to find the factorization of N.

The reader should be reminded that, just as in Section 2.4, the polynomials $H_1(x, y)$, $H_2(x, y)$ are not guaranteed to be algebraically independent; they might have a common factor. Indeed, in the general case we cannot guarantee that the resultant $H(y)$ is not identically zero. Consequently, we cannot claim this approach as a theorem. At the moment it is a heuristic. Our experiments suggest show it is a very good heuristic, as discussed in Section 6.5. We could not find a single example where the algorithm fails. The reason the algorithm works so well is that in our lattice, short vectors produced by LLL appear to behave as independent vectors.

6.3 Improved Determinant Bounds

The results of the last section show that the small inverse problem can be solved when $\delta < 0.284$. The bound is derived from the determinant of the lattice L, which gives an upper bound on the lengths of the shortest vectors of the lattice. In this section, we improve the bounds on the lengths of the shortest vectors of L, and show that these improved bounds imply the attack is effective for all $d < N^{0.292}$.

We begin with a brief discussion of how we may improve the bounds on the shortest vectors. In the last section, we compute the determinant of a lattice L generated by shifts and powers of f. Since L is full rank and its corresponding matrix triangular, the determinant is just the product of the entries on the diagonal – carefully balanced so that this product is less than (approximately) 1. Once $\delta > 0.284$ the approach no longer works, as this product exceeds 1 for every choice of m. But if some of the larger terms of this product were removed, we might be able to afford greater values of δ. Intuitively, this suggests that we should “throw away” rows of M with large contributions to the diagonal. Unfortunately, the resulting lattice is not full rank, and computing its determinant is not so easy. What we will show is that a judicious choice of rows to eliminate results in lattice for which there is an improved bound on the determinant, leading to a successful attack for all $\delta < 0.292$. Specifically, we show that as long as $\delta < 0.292$, there is a sublattice L' of L in which LLL will find vectors that are short enough. Most of this section is devoted to developing the necessary tools for bounding the determinant of non-full rank lattices. These results may be of independent interest.

We use the following approach. First, we introduce the notion of *geometrically progressive* matrices, and state the main theorem to be used to bound the determinant of a submatrix of any geometrically progressive matrix. A proof of this theorem is given in Appendix D. Second, we show that the portion of the matrix M developed in Section 6.2 corresponding to the y-shifts is geometrically progressive, yielding desirable bounds on the rectangular matrix formed from selected rows of M. Third, we review the new determinant computation and conclude that the attack outlined in Section 6.2 works for all $d < N^{0.292}$.

Geometrically Progressive Matrices

Recall the lattice L defined from the coefficients vectors of shifts and powers of the bivariate polynomial $f(x, y)$. Of particular interest is the inclusion of the y-shifts $h_{j,k}(xX, yY)$, which lead to a result improving on Wiener's bound. We begin by noting that there is a natural organization of these rows corresponding to y-shifts into "blocks" $h_{1,k}, \ldots, h_{t,k}$ for $k = 0, \ldots, m$, and that a similar organization is induced on the corresponding columns (that is, those columns that are zero in every row induced by an x-shift). To keep the results of this section general, we work with generic matrices in which the rows and columns have been divided into $a + 1$ blocks of size b. Specifically, let a, b be positive integers and let M be an $(a + 1)b \times (a + 1)b$ matrix. We index the columns by pairs (i, j), with $i = 0, \ldots, a$ and $j = 1, \ldots, b$, so that the pair (i, j) corresponds to the $(b \cdot i + j)$th column of M. Similarly, we use the pair (k, ℓ) to index the $(b \cdot k + \ell)$th row of M, for $k = 0, \ldots, a$ and $\ell = 1, \ldots, b$. The entry in the (i, j)th column of the (k, ℓ)th row is denoted $M(i, j, k, \ell)$. Note that the diagonal entries of M are precisely those of the form $M(k, \ell, k, \ell)$.

Definition 6.3.1 *Let $C, D, c_1, c_2, c_3, c_4, \beta$ be real numbers with C, D, $\beta \geq 1$. A* *matrix M is said to be* geometrically progressive with parameters $(C,\ D,\ c_1,\ c_2,\ c_3,\ c_4,\ \beta)$ *if the following conditions hold for all $i,\ k = 0, \ldots, a$ and $j,\ \ell = 1, \ldots, b$:*

(i) $|M(i,j,k,\ell)| \leq C \cdot D^{c_1 i + c_2 j + c_3 k + c_4 \ell}$.

(ii) $M(k,\ell,k,\ell) = D^{c_1 k + c_2 \ell + c_3 k + c_4 \ell}$.

(iii) $M(i,j,k,\ell) = 0$ *whenever $i > k$ or $j > \ell$.*

(iv) $\beta c_1 + c_3 \geq 0$ *and* $\beta c_2 + c_4 \geq 0$.

When the parameters $C, D, c_1, c_2, c_3, c_4, \beta$ are understood we say simply that M is geometrically progressive.

The main tool we use in this section is the following theorem, which provides a good bound on the determinant of a geometrically progressive matrix with deleted rows. A proof is given in Appendix D.

Theorem 6.3.1 *Let M be an $(a+1)b \times (a+1)b$ geometrically progressive matrix with parameters $(C,\ D,\ c_1,\ c_2,\ c_3,\ c_4,\ \beta)$, and let B be a real number. Define*

$$S_B := \{(k,\ell) \in \{0, \ldots, a\} \times \{1, \ldots, b\} \mid M(k,\ell,k,\ell) \leq B\},$$

and set $s_b := |S_B|$, the number of elements in S_B.

If L is the lattice defined by the rows $(k,\ell) \in S_B$ of M, then

$$\det(L) \leq ((a+1)b)^{s_b/2}(1+C)^{s_b^2} \prod_{(k,\ell)\in S_B} M(k,\ell,k,\ell).$$

The basic idea is that when we remove rows with large entries on the diagonal, the resulting submatrix yields a sublattice with a determinant close to what is expected, to within a certain multiplicative error.

A Geometrically Progressive Submatrix

Recall the procedure outlined in Section 6.2 for creating the lattice L. We define the polynomials

$$g_{i,k}(x,y) = x^i f^k(x,y) \quad \text{and} \quad h_{j,k}(x,y) = y^j f^k(x,y),$$

and form a lattice from the coefficients vectors of every $g_{i,k}(xX, yY)$ and $h_{j,k}(xX, yY)$, for $k = 0, \ldots, m$, $i = 0, \ldots, m-k$, and $j = 1, \ldots, t$.

We denote by M_y the portion of the matrix M with rows corresponding to the y-shifts $h_{j,k}(xX, yY)$ and columns corresponding to variables of the form $x^u y^v$, $v > u$. Specifically, M_y is the $(m+1)t \times (m+1)t$ lower-right-hand submatrix of the matrix M presented in Section 6.2. We make the following claim about the entries of M_y.

Lemma 6.3.2 *For all positive integers m, t, the matrix M_y is geometrically progressive with parameters (m^{2m}, e, $\frac{1}{2}+\delta$, $-\frac{1}{2}$, -1, 1, 2).*

Proof For simplicity, we take $e = N^\alpha$ with $\alpha = 1$. Let (k, ℓ) be given with $k = 0, \ldots, m$ and $\ell = 1, \ldots, t$. The row (k, ℓ) of M_y corresponds to the y-shift $h_{\ell,k}(xX, yY)$. Observe

$$h_{\ell,k}(xX, yY) = e^{-k} y^\ell Y^\ell f^k(xX, yY) = \sum_{u=0}^{k} \sum_{v=0}^{u} c_{u,v} x^u y^{v+\ell},$$

where

$$c_{u,v} = \binom{k}{u}\binom{u}{v}(-1)^{k-u} e^{-k} A^{u-v} X^u Y^{v+\ell}.$$

The column (i, j) for $i = 0, \ldots, m$ and $j = 1, \ldots, t$ corresponds to the coefficient of $x^i y^{i+j}$ in $h_{\ell,k}(xX, yY)$, which by the above is

$$M_y(i, j, k, \ell) = c_{i,i+j-\ell} = \binom{k}{i}\binom{i}{i+j}(-1)^{k-i} e^{-k} A^{\ell-j} X^i Y^{i+j}.$$

It is easy to see that the above quantity equals 0 whenever $i > k$ or $j > \ell$, satisfying condition **(iii)**. Writing $X = e^\delta$, $Y = e^{\frac{1}{2}}$ and knowing $A < e$, we see

$$|M_y(i,j,k,\ell)| \leq \left| \binom{k}{i} \binom{i}{i+j} (-1)^{k-i} e^{(\frac{1}{2}+\delta)i - \frac{1}{2}j - k + \ell} \right| \leq m^{2m} \cdot e^{(\frac{1}{2}+\delta)i - \frac{1}{2}j - k + \ell},$$

satisfying condition **(i)**. Furthermore, routine calculation confirms

$$M_y(k,\ell,k,\ell) = e^{(\frac{1}{2}+\delta)k - \frac{1}{2}\ell - k + \ell},$$

satisfying condition **(ii)**. Lastly, observe $2\cdot(\frac{1}{2}+\delta)+(-1) = 2\delta \geq 0$ and $2\cdot -\frac{1}{2}+1 \geq 0$, so condition **(iv)** is met. Hence, M_y is geometrically progressive with parameters (m^{2m}, e, $\frac{1}{2}+\delta$, $-\frac{1}{2}$, -1, 1, 2). ■

Remark 3. When $\alpha < 1$ we find that M_y is geometrically progressive with parameters (m^{2m}, e, $\frac{1}{2\alpha} + \frac{\delta}{\alpha}$, $\frac{1}{2\alpha} - 1$, -1, 1, 2α). For $\alpha > 1$, we have that M_y is geometrically progressive with parameters ($(2m)^{2m}$, e, $\frac{1}{2\alpha} + \frac{\delta}{\alpha}$, $-\frac{1}{2\alpha}$, -1, $\frac{1}{\alpha}$, 2α). The proofs of these statements follow as above, with the slight modification in the latter case where we use $A < 2e^{1/\alpha}$ instead of $A < e$.

Bounding the Determinant of the New Lattice

We now have the tools necessary to find improved bounds on the short vectors of L. Namely, we now would like to show that for all $d < N^{0.292}$, LLL finds short vectors in M that give rise to polynomials $H_1(x,y)$ and $H_2(x,y)$ such that $H_1(x_0,y_0) = H_2(x_0,y_0) = 0$.

We begin by setting the parameter $t := (1-2\delta)m$. Note that this means our lattice will include twice as many y-shifts as used in Section 6.2, which, as we shall see, is the reason for the improved results. Define M_1 as follows: Take every row $g_{i,k}$ of M corresponding to the x-shifts, and take only those rows $h_{\ell,k}$ of M whose entry on the diagonal is less than or equal to 1. That is, we throw away those rows $h_{\ell,k}$ where the leading term exceeds 1. Clearly, the lattice L_1 described by M_1 is a sublattice of L, so short vectors in L_1 will be in L.

Since all x-shifts are present in M_1, we may perform Gaussian elimination to set the first $(m+1)(m+2)/2$ off-diagonal columns of every row to zero. Specifically, there is a unitary matrix A over $\mathbb{R}$ such that $M_2 := AM_1$ is a matrix of the following form:

$$\begin{array}{r|c|c|c|c|}
 & 1\ x\ xy\ \cdots\ x^m y^m & y\ \cdots\ y^t & \cdots & x^m y^{m+1}\ \cdots\ x^m y^{m+t} \\
\hline
x\text{-shifts} & \Lambda & \multicolumn{3}{c|}{0} \\
\hline
\text{selected } y\text{-shifts} & 0 & \multicolumn{3}{c|}{M'_y} \\
\hline
\end{array}$$

where Λ is a diagonal matrix and M'_y consists of our selected rows of M_y. Furthermore, since A is unitary, the determinant of the lattice L_2 described by M_2 is equal to $\det(L_1)$.

We would like to obtain a good bound on $\det(L_2)$. Since the x-shifts and selected y-shifts portions of the lattice L_2 are orthogonal, it is sufficient to bound the determinant of each separately. Let w' be the number of rows of M'_y, and denote by L'_y the lattice induced by M'_y. The determinant of the lattice L_2 is $\det(L_2) = \det(\Lambda) \cdot \det(L'_y)$, and its dimension is $w = (m+1)(m+2)/2 + w'$. As we shall see, the dimension w is only a function of δ (but not of e), so γ is only a fixed constant.

We begin by computing w'. Let $S \subseteq \{0, \ldots, m\} \times \{1, \ldots, t\}$ be the subset of indices such that $M_y(k, \ell, k, \ell) \leq 1$ for $(k, \ell) \in S$, so S is a set with exactly w' elements. Since $(k, \ell) \in S$ only if $e^{(\delta - \frac{1}{2})k + \frac{1}{2}\ell} \leq 1$, we know $\ell \leq (1-2\delta)k$. Since we have taken $t = (1-2\delta)m$, we know every every pair (k, ℓ) satisfies $\ell \leq (1-2\delta)k \leq t$, so $\ell \leq (1-2\delta)k$ if and only if $(k, \ell) \in S$. Thus

$$w' = \sum_{k=0}^{m} \lfloor (1-2\delta)k \rfloor \geq \sum_{k=0}^{m} [(1-2\delta)k - 1] = (\frac{1}{2} - \delta)m^2 + o(m^2),$$

implying

$$w = w' + (m+1)(m+2)/2 = (1-\delta)m^2 + o(m^2).$$

Now we bound $\det(L'_y)$. Since this lattice is defined by the rows $(k, \ell) \in S$ of M_y, by Theorem 6.3.1 we have

$$\begin{aligned}\det(L'_y) &\le C_L \cdot \prod_{(k,\ell)\in S} M_y(k,\ell,k,\ell)\\ &\le C_L \cdot \prod_{k=0}^{m} \prod_{\ell=0}^{\lfloor(1-2\delta)k\rfloor} e^{(\delta-\frac{1}{2})k+\frac{1}{2}\ell}\\ &\le C_L \cdot e^{(-\frac{1}{12}+\frac{\delta}{3}-\frac{\delta^2}{3})m^3+o(m^3)},\end{aligned}$$

where $C_L := \left[(m+1)(1-2\delta)m\right]^{w'/2}(1+m^{2m})^{(w')^2}$.

Finally, recall from Section 6.2 that

$$\det(\Lambda) = \_x = e^{-m(m+1)(m+2)/6}\cdot X^{m(m+1)(m+2)/3}\cdot Y^{m(m+1)(m+2)/6} = e^{(\frac{\delta}{3}-\frac{1}{12})m^3+o(m^3)}.$$

So to satisfy bound (6.3) we require

$$\det(L_1) = \det(\Lambda)\det(L'_y) \le C_L \cdot e^{(\frac{\delta}{3}-\frac{1}{12})m^3+(-\frac{1}{12}+\frac{\delta}{3}-\frac{\delta^2}{3})m^3+o(m^3)} < e^{-m}/\gamma', \qquad (6.4)$$

where $\gamma' = 2^{(w^2-1)/4}w^{(w-1)/2}$. Note that $C_L\cdot\gamma'$ is a function of only δ (but not of e), so we may make the approximation $\log_e(C_L\cdot\gamma') \approx 0$. So bound (6.4) leads to

$$\left(-\frac{1}{6}+\frac{2\delta}{3}-\frac{\delta^2}{3}\right)m^3+o(m^3) < 0,$$

implying $2\delta^2-4\delta+1 \ge 0$. Hence, we need

$$\delta < 1-\frac{\sqrt{2}}{2} \approx 0.292.$$

Thus, when $\delta < 0.292$, for sufficiently large m we have $\det(L_1) \le e^{-m}/(\gamma'\cdot C_L)$, implying that the first and second vectors of the LLL-reduced L_1 (denoted $H_1(xX, yY)$ and $H_2(xX, yY)$, respectively) have norm less than $1/\sqrt{w}$. By Fact 2.4.1 these polynomials satisfy $H_1(x_0,y_0) = H_2(x_0,y_0) = 0$. As before, if these polynomials turn out to be algebraically independent, we may compute the resultant $H(y) := \mathrm{Res}_x(H_1(x,y), H_2(x,y))$ and solve for the roots of $H(y)$ to expose $y_0 = -(p+q)$, revealing the factorization of N.

6.4 Cryptanalysis of Arbitrary Public Exponents

In his paper, Wiener suggests using large values of e when the exponent d is small. This can be done by adding multiples of $\phi(N)$ to e before making it known as the public key. When $e > N^{1.5}$, Wiener's attack will fail even when d is small. We show that our attack applies even when $e > N^{1.5}$ is used.

As described in Section 6.1, we solve the small inverse problem:

$$k(A+s) \equiv 1 \pmod{e} \quad \text{where} \quad |k| < e^{1+\frac{\delta-1}{\alpha}} \text{ and } |s| < 3e^{1/(2\alpha)},$$

for arbitrary values of α. We build the exact same lattice used in Section 6.2. Working through the calculations one sees that the determinant of the lattice in question is

$$\begin{aligned} \det_x(L) &= e^{\frac{m^3}{6\alpha}(\alpha+2\delta-\frac{3}{2})+o(m^3)}, \\ \det_y(L) &= e^{\frac{tm^2}{2\alpha}(\delta-\frac{1}{2})+\frac{mt^2}{2}\frac{1}{2\alpha}+o(tm^2)}. \end{aligned}$$

The dimension is as before. Therefore, to apply Fact 2.3.1 we must have

$$\frac{m^3}{6\alpha}(\alpha+2\delta-\frac{3}{2}) + \frac{tm^2}{2\alpha}(\delta-\frac{1}{2}) + \frac{mt^2}{2}\frac{1}{2\alpha} < 0,$$

which leads to

$$m^2(2\alpha+4\delta-3) - 3tm(1-2\delta) + 3t^2 < 0.$$

As before, the left hand side is minimized at $\min t = \frac{1}{2}m(1-2\delta)$, which leads to

$$m^2[2\alpha+7\delta-\frac{15}{4}-3\delta^2] < 0,$$

and hence

$$\delta < \frac{7}{6} - \frac{1}{3}(1+6\alpha)^{1/2}.$$

Indeed, for $\alpha = 1$, we obtain the results of Section 6.2. The expression shows that when $\alpha < 1$ our attack becomes even stronger. For instance, if $e \approx N^{2/3}$ then RSA is insecure whenever $d < N^{\delta}$ for $\delta < \frac{7}{6} - \frac{\sqrt{5}}{3} \approx 0.422$. Note that if $e \approx N^{2/3}$ then d must satisfy $d > N^{1/3}$.

When $\alpha = \frac{15}{8}$ the bound implies that $\delta = 0$. Consequently, the attack becomes totally ineffective whenever $e > N^{1.875}$. This is an improvement over Wiener's attack, which becomes ineffective as soon as $e > N^{1.5}$.

6.5 Experiments

We ran several dozen experiments to test our results when $d > N^{0.25}$. Our experiments were carried out using the LLL implementation available in Victor Shoup's NTL package [72]. In all our experiments LLL produced two independent relations $H_1(x, y)$ and $H_2(x, y)$. In every case, the resultant $H(y) := \text{Res}_x(H_1(x, y), H_2(x, y))$ with respect to x was a polynomial of the form $H(y) = (y + p + q)H_0(y)$, with $H_0(y)$ irreducible over $\mathbb{Z}$ (similarly for x). Hence, the unique solution (x_0, y_0) was correctly determined in every trial executed. We show the parameters of some attacks executed in Figure 6.2. In each of these tests, d was chosen uniformly at random in the range

N	d	δ	m	t	lattice dim.	running time	Advantage over Wiener's attack
1024 bits	283 bits	0.277	7	3	45	2.5 hours	27 bits
2048 bits	550 bits	0.275	7	3	45	16 hours	50 bits
4096 bits	1060 bits	0.265	5	2	25	3 hours	60 bits
10000 bits	2550 bits	0.255	3	1	11	19 minutes	50 bits

Experiments performed on a 1GHz Intel Pentium III running Linux.

Figure 6.2: Running times for short secret exponent attack

$\left[\frac{3}{4}N^{\delta}, N^{\delta}\right]$ (thus guaranteeing the condition $d > N^{0.25}$). The second to last row of the table is especially interesting as it is an example in which our attack breaks RSA with a private key that is 60 bits larger than Wiener's bound.

6.6 Conclusions

Our results show that Wiener's bound on low private exponent RSA is not tight. In particular, we were able to improve the bound first from $d < N^{0.25}$ to $d < N^{0.284}$. Using an improved analysis of the determinant, we obtained $d < N^{0.292}$. Our results also improve Wiener's attack when large values of e are used. We showed that our attack becomes ineffective only once $e > N^{1.875}$. In contrast, Wiener's attack became ineffective as soon as $e > N^{1.5}$.

Unfortunately, we cannot state this attack as a theorem since we cannot prove that it always succeeds. However, experiments that we carried out demonstrate its effectiveness. We were not able to find a single example where the attack fails. This is similar to the situation with many factoring algorithms, where one cannot prove that they work; instead one gives strong heuristic arguments that explain their running time. In our case, the heuristic "assumption" we make is that the two shortest vectors in an LLL reduced basis give rise to algebraically independent polynomials. Our experiments confirm this assumption. We note that a similar assumption is used in the work of Bleichenbacher [3] and Jutla [44].

To conclude, we note that Wiener suggested a defense against the low private exponent attack based on the Chinese Remainder Theorem. When $N = pq$ the idea is to use a secret exponent d such that both $d_p = d \bmod (p-1)$ and $d_q = d \bmod (q-1)$ are small. Such d speed up RSA signature generation since RSA signatures are often generated modulo p and q separately and then combined using the CRT. Since $d_p \neq d_q$ the value of d is likely to be large, namely close to $\phi(N)$. Consequently, our low exponent attack does not apply to such d. It is an open problem whether there is an efficient attack on such private keys. The best known attack runs in time $\min(\sqrt{d_p}, \sqrt{d_q})$.

Chapter 7

Attacks on RSA Variants

At Asiacrypt '99, Sun, Yang and Laih [76] noticed that the Wiener attack [80] (Section 3.2.4) and the attack of the previous chapter require some natural assumptions on the public modulus N. For instance, the Wiener's bound $N^{0.25}$ only holds if $p + q = O(\sqrt{N})$ and e is not too large. Similar restrictions apply to the extension to Wiener's attack by Verheul and van Tilborg [79] and to the attack in Chapter 6. This led Sun, Yang and Laih to propose in [76] simple variants of RSA using a short secret exponent that, *a priori*, foiled all such attacks.

They proposed three RSA-like schemes in which the only the RSA key generation is modified. In the first scheme, one chooses p and q of greatly different size, and a small exponent d in such a way that the previous attacks cannot apply. In particular, d can smaller than $N^{0.25}$ if p and q are unbalanced enough and the scheme will still resist previous attacks. The second scheme consists of a tricky construction that selects slightly unbalanced p and q in such a way that both e and d are small, roughly around $\sqrt{N}$. The third scheme is a mix of the first two schemes, which allows a trade-off between the sizes of e and d. Sakai, Morii and Kasahara [70] earlier proposed a different key generation scheme which achieves similar results to the third scheme, but that scheme can easily been shown insecure [76].

In this chapter we show that the first and third Sun-Yang-Laih schemes are insecure. We accomplish this by introducing a powerful generalization of the attack in the previous chapter. Our attack can also break the second scheme, but only if the parameters are carelessly chosen.

Recall that the last chapter reduced the problem of recovering the factors p and q to solving the Small Inverse Problem (equation (6.2)). However, when p and q are unbalanced this equation is not enough, because it has no longer any "small" solution. Our attack extends method of the previous chapter by taking into account the equation $N = pq$. We use a system of two modular equations with three unknowns; interestingly, when p and q are unbalanced, this approach leads to an attack on systems with d even larger than the $N^{0.292}$ bound of the previous chapter. The attack is extremely efficient in practice: for typical instances of two of the schemes of [76], this approach breaks the schemes within several minutes. Also, our "trivariate" version of Coppersmith's technique we use may be of independent interest.

7.1 The Sun-Yang-Laih Key Generation Schemes

Scheme (I)

The first scheme uses unbalanced RSA moduli [71], that is, $N = pq$ for $p \ll q$. The recommended parameters are chosen to defeat the Wiener attack and the attacks of the previous chapter.

1. Fix desired bit-length n for N, bit-length ℓ_d for d, bit-length ℓ_p for p, and security parameter $\gamma > 64$, subject to the following constraints:

   - $\ell_d + \ell_p > n/3$, and
   - $\ell_d > \gamma + \ell_p/2$.

   Note that the smaller ℓ_p is, the smaller ℓ_d is allowed to be.

2. Select the prime p of bit length ℓ_p and the prime q of bit length $n - \ell_p$ uniformly at random. Note that both n and ℓ_p must be large enough so that $N = pq$ cannot be factored efficiently by ECM or NFS.

3. Pick the secret exponent d uniformly at random from ℓ_d-bit integers.

4. If the public exponent e defined by $ed \equiv 1 \bmod \phi(N)$ is not larger than $\phi(N)/2$, go to Step 3.

A choice of parameters suggested by the authors is: p is a 256-bit prime, q is a 768-bit prime, d is a 192-bit number. Note that 192 is far below the Wiener bound of 256 bits, and the bound of 299 bits achieved in the previous chapter, yet the choice of parameters foils both attacks.

Scheme (II)

The second scheme selects one of the primes in such a way that allows e and d to be simultaneously small.

1. Fix the bit-length n of N, and set $\ell_p = (n/2) - 112$ and $\ell_d = (n/2) + 56$.
2. Select a random prime p of ℓ_p and a random k of 112 bits.
3. Select a random d of ℓ_d bits such that $\gcd(d, k(p-1)) = 1$.
4. Compute u and v such that $du - k(p-1)v = 1$ with $0 < u < k(p-1)$ and $0 < v < d$.
5. If $\gcd(v+1, d) \neq 1$ then return to Step 3.
6. Select a random h of 56 bits until $q = v + hd + 1$ is prime.
7. Set $e := u + hk(p-1)$ and $N = pq$.

Notice that e and d satisfy the equation $ed = 1 + k\phi(N)$. They both have approximate bit-length $(n/2) + 56$. The primes p and q have approximate bit-length $(n/2) - 112$ and $(n/2) + 112$ respectively.

A possible choice of parameters for Scheme (II) might be: p a 400-bit prime, q a 624-bit prime, and e and d are each 568-bit integers.

Scheme (III)

The third scheme is a mix of the first two schemes, allowing a trade-off between the lengths of e and d. More precisely, the scheme is a parametrized version of scheme (II), where p, k, d and h have bit-lengths ℓ_p, ℓ_k, ℓ_d, and $n - \ell_p - \ell_d$, respectively. These parameters may be chosen freely, but to resist various attacks, the following is required:

- $\ell_p > n/2$;
- $\ell_k \gg \ell_p - \ell_d + 1$;
- $4\alpha(2\beta + \alpha - 1) \gg 3(1 - \beta - \alpha)^2$, where $\alpha = (n - \ell_p)(n + \ell_k - \ell_d)$ and $\beta = \ell_k/(n + \ell_k - \ell_d)$;
- $\ell_k + \ell_p > n/3$; and
- k must withstand an exhaustive search, i.e., $\ell_k > 64$.

A choice of parameters suggested by the authors is: p is a 256-bit prime, q is a 768-bit prime, e is an 880-bit number, and d is a 256-bit number.

7.2 The Attack Algorithm

In this section we demonstrate how to launch an attack on Schemes (I) and (III). The approach used here closely follows the low private exponent attacks described in the previous chapter, but differs in several crucial ways to allow it to work for these variants of RSA. Interestingly, our attack gets better (works for larger and larger d) the more unbalanced the factors of the modulus become.

Recall the defining equation for RSA:

$$ed + k(N + 1 - (p + q)) = ed + k\phi(N) = 1. \tag{3.1}$$

We note that the approach in the last chapter treats this as an equation with two "small" unknowns, k and $s = -(p + q)$. This approach no longer works if p and q are unbalanced, since a good bound on s can no longer be established. For this reason, the authors of the schemes from Section 7.1 hoped that these schemes would resist attack. However, we will see that a more careful analysis of the RSA equation, namely one that does not treat $p+q$ as a single unknown quantity but instead leaves p and q separately as unknowns, leads to a successful attack against two of these schemes.

Writing $A = N+1$, the RSA equation implies

$$1 + k(A - p - q) \equiv 0 \pmod{e}.$$

The critical improvement of our attack is to view this as a modular equation with *three* unknowns, k, p, q, with the special property that the product pq of two of them is the known quantity N. We may view this problem as follows: given a polynomial $f(x,y,z) = (x(A+y+z)-1)/e$, find (x_0, y_0, z_0) satisfying:

$$f(x_0, y_0, z_0) \in \mathbb{Z},$$

where

$$|x_0| < X, \quad |y_0| < Y, \quad |z_0| < Z, \quad \text{and} \quad y_0 z_0 = N.$$

Note that the bounds $X \approx ed/N$, $Y \approx p$, and $Z \approx q$ can be estimated to within a power of 2 based on the security parameters chosen for the scheme.

Following Coppersmith's method, our approach is to pick r equations of the form $x^{u_1} y^{u_2} z^{u_3} \cdot f^v(x,y,z)$ and to search for integer linear combinations of these polynomials with low weighted norm. However, the method for choosing these polynomials is not as straightforward as in previous chapters, and will take special consideration.

The basic idea is to start with a handful of equations of the form $y^{a+j} f^m(x,y,z)$ for $j = 0, \ldots, t$ for some integers a and t with $t \geq 0$. Knowing $N = pq$ allows us to replace all occurrences of the monomial yz with the constant N, reducing the number of variables in each of these equations to approximately m^2 instead of the expected $\frac{1}{3}m^3$. We will refer to these as the *primary polynomials*.

Since there are only $t+1$ of these equations, this will result in a lattice that is less than full rank; we therefore include some additional equations to bring the lattice to full rank in order to compute its determinant. We refer to these as the *helper polynomials*. We have a great deal of choice in picking the helper polynomials; naturally, some choices are better than others, and it is generally a tedious optimization problem to choose the primary and helper polynomials that are optimal. The equations we work with are the following. Fix an integer m, and let a and $t > 0$ be integers which we will optimize later. We define

- $g_{k,i,b}(x, y, z) := x^i y^a z^b f^k(x, y, z)$, for $k = 0, \dots, m-1$; $\quad i = 1, \dots, m-k$; $b = 0, 1$; and,
- $h_{k,j}(x, y, z) := y^{a+j} f^k(x, y, z)$, for $k = 0, \dots, m$, and $j = 0, \dots, t$.

The primary polynomials are $h_{m,j}(x, y, z)$ for $j = 0, \dots, t$, and the rest are the helper polynomials. Following the method technique, we form a lattice L from $g_{k,i,b}(xX, yY, zZ)$ and $h_{k,j}(xX, yY, zZ)$ and use LLL to find low-norm integer linear combinations $h_1(xX, yY, zZ)$ and $h_2(xX, yY, zZ)$. The polynomials $h_1(x, y, z)$ and $h_2(x, y, z)$ have (k, p, q) as a root; to remove z as an unknown, we use the equality $z = N/y$, obtaining $H_1(x, y)$ and $H_2(x, y)$ which have (k, p) as a solution. As long as $H_1(x, y)$ and $H_2(x, y)$ are algebraically independent, we can compute the resultant $\text{Res}_x(H_1(x, y), H_2(x, y))$ yields a polynomial $H(y)$ which has p as a root. Using standard root-finding techniques allows us to recover the factor p of N efficiently, completing the attack.

The running time of this algorithm is dominated by the time to run LLL on the lattice L, which has dimension $(m + 1)(m + t + 1)$. So it would be ideal to keep the parameters m and t as low as possible, limiting to a reasonable number the polynomials used to construct L. Surprisingly, the attack is successful even if only a handful of polynomials are used. The example given by the original authors for schemes (I) succumbs easily to this attack with $m = 3$ and $t = 1$; with these parameters, our attack generates 20 polynomials. Scheme (III) can be cryptanalyzed with parameters $m = 2$ and $t = 2$, yielding 15 polynomials. This gives lattices of dimension 20 (see Figure 7.1) and 15, respectively, which can be reduced *via* the LLL algorithm within a matter of seconds on a desktop computer. We discuss our implementation and the results of our experiments more in Section 7.3.

Analysis of the Attack

In order to be sure that LLL returns vectors that are short enough to use Fact 2.4.1, we must derive sufficiently small bounds on the determinant of the lattice L formed from the polynomials $g_{k,i,b}(xX, yY, zZ)$ and $h_{k,j}(xX, yY, zZ)$. Fortunately, this choice of polynomials makes the computation of the determinant of L fairly straightforward, if somewhat tedious. We provide the details in appendix E.

Representing the Lattice as a Triangular Matrix. In order to compute the volume of the lattice L, we would like to list the polynomials $g_{k,i,b}(xX, yY, zZ)$ and $h_{k,j}(xX, yY, zZ)$ in a way that yields a triangular matrix. There is an ordering on these polynomials that leads to such a representation: we first list the $g_{k,i,b}(xX, yY, zZ)$ indexed outermost by $k = 0, \ldots, m-1$, then $i = 0, \ldots, k-1$, then innermost by $b = 0, 1$. After all of these we list $h_{k,j}(xX, yY, zZ)$ indexed outermost by $k = 0, \ldots, m$ then $j = 0, \ldots, t$. (See Figure 7.1 for the case of $m = 2$, $t = 1$, $a = 1$.) Each new polynomial introduces exactly one new monomial $x^{u_1}y^{u_2}$ or $x^{u_1}z^{u_3}$. Note that no monomial involving the product yz appears since we have applied the substitution $yz = N$, and powers of N can be easily eliminated from the diagonal (See Remark 1 below).

The determinant of this matrix is simply the product of the entries on the diagonal, which for $m = 3$, $t = 1$, $a = 1$ is

$$\det(L) = e^{-20}X^{40}Y^{34}Z^{4}. \tag{7.2}$$

We expect the LLL algorithm to return vectors short enough to use Lemma 2.4.1 when $\det(L) \ll 1$. The example given by the original authors for Scheme (I) is to use p of 256 bits, q of 768 bits, d of 192 bits, and e of 1024 bits. This gives bounds

$$X \approx ed/N \approx e^{3/16}, \quad Y \approx e^{1/4}, \text{ and } Z \approx e^{3/4};$$

we may then confirm

$$\det(L) = e^{-20}X^{40}Y^{34}Z^{4} \approx e^{-1} \ll 1,$$

$$
\begin{array}{r|cccccc|cccc|cc||cccccc|cc|}
 & xy & x & x^2y & x^2 & x^3y & x^3 & x^2y^2 & x^2z & x^3y^2 & x^3z & x^3y^3 & x^3z^2 & y & y^2 & xy^2 & xy^3 & x^2y^3 & x^2y^4 & x^3y^4 & x^3y^5 \\
\hline
xy & XY & & & & & & & & & & & & & & & & & & & \\
xyz & & X & & & & & & & & & & & & & & & & & & \\
x^2y & & & X^2Y & & & & & & & & & & & & & & & & & \\
x^2yz & & & & X^2 & & & & & & & & & & & & & & & & \\
x^3y & & & & & X^3Y & & & & & & & & & & & & & & & \\
x^3yz & & & & & & X^3 & & & & & & & & & & & & & & \\
\hline
xyf & - & & - & - & & & e^{-1}X^2Y^2 & & & & & & & & & & & & & \\
xyzf & & - & - & - & & & & e^{-1}X^2Z & & & & & & & & & & & & \\
x^2yf & & & - & & - & - & & & e^{-1}X^3Y^2 & & & & & & & & & & & \\
x^2yzf & & & & - & - & - & & & & e^{-1}X^3Z & & & & & & & & & & \\
\hline
xyf^2 & - & & - & - & - & - & - & & - & - & e^{-2}X^3Y^3 & & & & & & & & & \\
xyzf^2 & & - & - & - & - & - & & - & - & - & & e^{-2}X^3Z^2 & & & & & & & & \\
\hline\hline
y & & & & & & & & & & & & & Y & & & & & & & \\
y^2 & & & & & & & & & & & & & & Y^2 & & & & & & \\
yf & - & - & & & & & & & & & & & - & & e^{-1}XY^2 & & & & & \\
y^2f & & - & & & & & & & & & & & & - & - & e^{-1}XY^3 & & & & \\
f^2 & - & - & - & - & & & - & - & & & & & - & & - & & e^{-2}X^2Y^3 & & & \\
y^2f^2 & - & & - & - & & & - & & & & & & & - & - & - & - & e^{-2}X^2Y^4 & & \\
\hline
yf^3 & - & - & - & - & - & - & - & - & - & - & - & - & - & & - & & - & & e^{-3}X^3Y^4 & \\
y^2f^3 & - & & - & - & - & - & - & & - & - & - & & & - & - & - & - & - & - & e^{-3}X^3Y^5 \\
\hline
\end{array}
$$

Example of the lattice formed by the vectors $g_{k,i,b}(xX, yY, zZ)$ and $h_{k,j}(xX, yY, zZ)$ when $m = 2$, $t = 1$, and $a = 1$. Entries marked with "–" indicate off-diagonal quantities whose values do not affect the determinant calculation.

Figure 7.1: Example lattice used in attacks on the Sun-Yang-Laih schemes

so Lemma 2.4.1 applies.[1] Therefore, when we run the LLL algorithm on this lattice, we will get two short vectors corresponding to polynomials $h_1(x, y, z)$, $h_2(x, y, z)$; by the bound on the determinant, we know that these polynomials will have norm that is low enough to use Lemma 2.4.1. Therefore these polynomials will have (k, p, q) as a solution over the integers. To turn these into bivariate equations, we use the equality $z = N/y$ to get $H_1(x, y)$ and $H_2(x, y)$ which have (k, p) as a solution over the integers. We then take the resultant $Res_x(H_1(x, y), H_2(x, y))$ to obtain a univariate polynomial $H(y)$ that has p as a root.

[1]The reader may have noticed that we have suppressed the error term associated with the execution of the LLL algorithm. But even if the LLL "fudge factor" is taken into account, the resulting error term is almost completely insignificant. In this example, since $w = 20$ we have $2^{w^2/4}w^{w/2} < 2^{241} \ll 2^{1023} < e$.

Slightly larger parameters m and t are required to rigorously obtain the bound for norm of the second basis vector, although in practice the LLL algorithm works well enough so that the parameters chosen here are sufficient.

More generally, if we pick optimal values for t and a take m sufficiently large, our attack will be successful for even larger bounds on d. The highest possible bound on d for which our attack can work depends on the parameters chosen for the scheme. Suppose the parameter $d \approx N^{\delta}$ is used. The table below summarizes the largest possible δ for which our attack can succeed. We point out the choices of parameters that give rise to the schemes of Section 7.1.

		$\log_N(e)$						
		1.0	0.9	0.86	0.8	0.7	0.6	0.55
$\log_N(p)$	0.5	0.284	0.323	0.339	0.363	0.406	0.451	0.475
	0.4	0.296	0.334	0.350	0.374	0.415	0.460	0.483_{II}
	0.3	0.334	0.369	0.384	0.406	0.446	0.487	0.510
	0.25	0.364_{I}	0.398	0.412_{III}	0.433	0.471	0.511	0.532
	0.2	0.406	0.437	0.450	0.470	0.505	0.542	0.562
	0.1	0.539	0.563	0.573	0.588	0.615	0.644	0.659

Largest δ (where $d < N^{\delta}$) for which our attack can succeed, as a function of the system parameters.

Figure 7.2: Success region of attacks on Sun-Yang-Laih schemes

For example, with the example for Scheme (I), where $e \approx N$ and $p \approx N^{0.25}$, our attack will be successful not only for the $\delta = 0.188$ suggested, but all the way up to $\delta < 0.364$ (assuming a large enough m is used.) Similarly, our attack works in Scheme (III) up to $d < N^{0.412}$. Notice that our attack comes close to, but cannot quite reach, the $d < N^{0.55}$ required to break Scheme (II).

Remark 1. Care must be taken when performing the substitution $yz \mapsto N$ to ensure that the coefficient of the polynomial appearing on the diagonal is not multiplied by powers of N. For example, suppose we are working with the polynomial $g(x, y, z) = x^{u_1} y^{u_2} z^{u_3} f^k(x, y, z)$ with $u_2 > u_3$. We see that

$$g(x, y, z) = e^{-k} N^{u_4} x^{u_5} y^{u_6} + v(x, y, z),$$

where $u_4 = u_2 - u_3$, $u_5 = k + u_1$, $u_6 = k + u_4$, and $v(x, y, z)$ is some polynomial in which the term $x^{u_5} y^{u_6}$ does not appear. Since N and e are relatively prime, we may compute integers A and B such that $AN^{u_4} + Be^k = 1$. Instead of using $g(x, y, z)$ in the lattice we use instead

$$g^*(x, y, z) := e^{-k}x^{u_5}y^{u_6} + A \cdot v(x, y, z).$$

We know that $g(x_0, y_0, z_0) \in \mathbb{Z}$, so $A \cdot g(x_0, y_0, z_0) \in \mathbb{Z}$. But

$$g^*(x_0, y_0, z_0) - A \cdot g(x_0, y_0, z_0) = (1 - AN^u)e^{-k}x_0^{u_5}y_0^{u_6} = Bx_0^{u_5}y_0^{u_6} \in \mathbb{Z},$$

so $g^*(x_0, y_0, z_0) \in \mathbb{Z}$. Hence $g^*(x, y, z)$ is a valid substitution for $g(x, y, z)$ when building the lattice.

7.2.1 Comparison with the Bivariate Approach

Alternatively, one can consider the system of two modular equations with three unknowns as a single bivariate equation by incorporating the equation $N = pq$ into the main trivariate equation. This was independently noticed by Willi Meier [58], who also addressed the problem of breaking Schemes (I) and (III), using a bivariate approach rather than our trivariate approach. One then obtains an equation of the form $f(x, y) = (x^2y + Axy + Bx + Cy)/e$, where the unknowns are k and the smallest prime among p and q.

However, it turns out that the application of Coppersmith's technique to this particular bivariate equation yields worse bounds than with the trivariate approach previously described. For example, the bivariate approach allows one to break scheme (I) as long as $d < N^{0.135}$ (and perhaps slightly higher, if sublattices are considered as in the previous chapter), but fails for larger d. One can view the bivariate approach a special case of our trivariate approach, in which one degree of freedom for optimization has been removed. One then sees that the bivariate approach constrains the choice of primary and helper polynomials in a suboptimal way, resulting in worse bounds on d.

7.3 Experiments

We implemented this attack using Victor Shoup's Number Theory Library [72] and the Maple Analytical Computation System [56]. The attack runs very efficiently, and in all instances of Schemes (I) and (III) we tested, it produced algebraically independent polynomials $H_1(x, y)$ and $H_2(x, y)$. These yielded a resultant $H(y) = (y - p)H_0(y)$, where $H_0(y)$ is irreducible, exposing the factor p of N in every instance. This strongly suggests that this "heuristic" assumption needed to complete the multivariate modular version of Coppersmith's technique is extremely reliable, and we conjecture that it always holds for suitably bounded lattices of this form. The running times of our attacks are given in Figure 7.3.

Scheme	size of n	size of p	size of e	size of d	m	t	a	lattice rank	running time
I	1024	256	1024	192	3	1	1	20	40 seconds
III	1024	256	880	256	2	2	0	15	9 seconds

Experiments performed on a 500MHz Intel Pentium III running Solaris.

Figure 7.3: Running times of attacks on Sun-Yang-Laih schemes.

7.4 Conclusions

We showed that unbalanced RSA [71] actually improves the attacks on short secret exponent by allowing larger exponent. This enabled us to break most of the RSA schemes [76] with short secret exponent from Asiacrypt '99. The attack extends the attack of the previous chapter by using a "trivariate" version of Coppersmith's lattice-based technique for finding small roots of low-degree modular polynomial equations. This attack is provable up to the step where the bivariate heuristic assumption is required.

These results illustrate once again the fact that one should be very cautious when using RSA with short secret exponent. Again, the best method to enjoy the computational advantage of short secret exponent is the countermeasure proposed by Wiener [80] to use a secret exponent d such that both $d \bmod (p-1)$ and d mod

$(q-1)$ are small. Such a d speeds up RSA signature generation when the signature is generated modulo p and q separately and combined using the Chinese Remainder Theorem. Classical attacks do not work since d is likely to be close to $\phi(N)$. It is an open problem whether there is an efficient attack on such secret exponents. The best known attack runs in time $\min(\sqrt{d_p}, \sqrt{d_q})$.

Bibliography

[1] M. Bellare and P. Rogaway. Optimal asymmetric encryption. In proceedings *Eurocrypt '94*, Lecture Notes in Computer Science, vol. 950, Springer-Verlag, pp. 92–111, 1994.

[2] D. Bleichenbacher. Chosen ciphertext attacks against protocols based on the RSA encryption standard PKCS #1. In proceedings *Crypto '98*, Lecture Notes in Computer Science, vol. 1462, Springer-Verlag, pp. 1–12, 1998.

[3] D. Bleichenbacher. On the security of the KMOV public key cryptosystem. In proceedings *Crypto '97*, Lecture Notes in Computer Science, vol. 1294, Springer-Verlag, pp. 235-248, 1997.

[4] D. Boneh. Finding smooth integers using CRT decoding. In proceedings *STOC 2000*, pp. 265–272, Portland, Oregon, 2000.

[5] D. Boneh. Simplified OAEP for the RSA and Rabin functions. In proceedings *Crypto 2001*, Lecture Notes in Computer Science, vol. 2139, Springer-Verlag, pp. 275–291, 2001.

[6] D. Boneh. Twenty years of attacks on the RSA cryptosystem. *Notices of the AMS*, 46(2):203–213, 1999.

[7] D. Boneh, R. DeMillo, and R. Lipton. On the importance of checking cryptographic protocols for faults. In proceedings *Eurocrypt '97*, Lecture Notes in Computer Science, vol. 1233, Springer-Verlag, pp. 37–51, 1997.

[8] D. Boneh and G. Durfee. Cryptanalysis of RSA with private key d less than $N^{0.292}$. In proceedings *Eurocrypt '99*, Lecture Notes in Computer Science, vol. 1592, Springer-Verlag, pp. 1–11, 1999.

[9] D. Boneh and G. Durfee. Cryptanalysis of RSA with private key d less than $N^{0.292}$. *IEEE Transactions on Information Theory*, vol. 46, no. 4, pp. 1339–1349, July 2000.

[10] D. Boneh, G. Durfee, and Y. Frankel. An attack on RSA given a small fraction of the private key bits. In proceedings *Asiacrypt '98*, Lecture Notes in Computer Science, vol. 1514, Springer-Verlag, pp. 25–34, 1998.

[11] D. Boneh, G. Durfee, and N. Howgrave-Graham. Factoring $N = p^r q$ for large r. In proceedings *Crypto '99*, Lecture Notes in Computer Science, vol. 1666, Springer-Verlag, pp. 326–337, 1999.

[12] D. Boneh and R. Venkatesan. Breaking RSA may not be equivalent to factoring. In proceedings *Eurocrypt '98*, Lecture Notes in Computer Science, vol. 1403, Springer-Verlag, pp. 59–71, 1998.

[13] R. Canetti, Y. Dodis, S. Halevi, E. Kushilevitz, A. Sahai. Exposure-Resilient Functions and All-or-Nothing Transforms. In proceedings *Eurocrypt 2000*, Lecture Notes in Computer Science, vol. 1807, Springer-Verlag, pp. 453–469, 2000.

[14] S. Cavallar, B. Dodson, A. K. Lenstra, W. Lioen, P. L. Montgomery, B. Murphy, H. te Riele, K. Aardal, J. Gilchrist, G. Guillerm, P. Leyland, J. Marchand, F. Morain, A. Muffett, C. Putnam, C. Putnam, and P. Zimmermann. Factorization of 512-bit RSA key using the number field sieve. In proceedings *Eurocrypt 2000*, Lecture Notes in Computer Science, vol. 1807, Springer-Verlag, 2000. Factorization announced in August, 1999.

[15] B. Chor, J. Friedman, O. Goldreich, J. Høastad, S. Rudich, R. Smolensky. The bit extraction problem or t-resilient functions. In proceedings *26th*

Annual Symposium on Foundations of Computer Science (FOCS), pp. 396–407, 1985.

[16] D. Coppersmith. Modifications to the number field sieve. *Journal of Cryptology*, vol. 6, pp. 169–180, 1993.

[17] D. Coppersmith. Small solutions to polynomial equations, and low exponent RSA vulnerabilities. *Journal of Cryptology*, vol. 10, pp. 233–260, 1997.

[18] D. Coppersmith. Finding small solutions to small degree polynomials. In proceedings *Cryptography and Lattice Conference*, Lecture Notes in Computer Science, vol. 2146, Springer-Verlag, 2001.

[19] D. Coppersmith, M. Franklin, J. Patarin, and M. Reiter. Low exponent RSA with related messages. In proceedings *Eurocrypt '96*, Lecture Notes in Computer Science, vol. 1070, Springer-Verlag, pp. 1–9, 1996.

[20] D. Coppersmith, S. Halevi, and C. S. Jutla. ISO 9796 and the New Forgery Strategy. Presented at Rump Session of Crypto '99, 1999.

[21] S. Coron, D. Naccache, and J. P. Stern. On the Security of RSA Padding. In proceedings *Crypto '99*, Lecture Notes in Computer Science, vol. 1666, Springer-Verlag, pp. 1–18, 1999.

[22] C. Coupé, P. Nguyen, and J. Stern. The effectiveness of lattice attacks against low-exponent RSA. In proceedings *Public Key Cryptography '99*, Lecture Notes in Computer Science, vol. 1751, Springer-Verlag, 1999.

[23] R. Cramer and V. Shoup. A practical public key cryptosystem provably secure against adaptive chosen ciphertext attack. Journal version, to appear.

[24] J-F. Dhem, F. Koeune, P. Leroux, P. Mestré, J-J. Quisquater, and J-L. Willems. A practical implementation of the timing attack. In proceedings *CARDIS '98* – Third smart card research and advanced application conference, UCL, Louvain-la-Neuve, Belgium, Sep. 14-16, 1998.

[25] Y. Dodis. Exposure-Resilient Cryptography. Ph.D. Thesis, MIT, 2000.

[26] Y. Dodis, A. Sahai, A. Smith. On perfect and adaptive security in exposure-resilient cryptography. In proceedings *Eurocrypt 2001*, Lecture Notes in Computer Science, vol. 2045, Springer-Verlag, pp. 301–324, 2001.

[27] G. Durfee and P. Nguyen. Cryptanalysis of the RSA schemes with short secret exponent from Asiacrypt '99. In proceedings *Asiacrypt 2000*, Lecture Notes in Computer Science, vol. 1976, Springer-Verlag, pp. 14–29, 2000.

[28] T. ElGamal. A public key cryptosystem and a signature scheme based on the discrete logarithm. *IEEE Transactions on Information Theory*, vol. 31, no. 4, pp. 469–472, 1985.

[29] Y. Frankel. A practical protocol for large group oriented networks. In proceedings *Eurocrypt '89*, Lecture Notes in Computer Science, vol. 434, Springer-Verlag, pp. 56–61.

[30] A. Freier, P. Karlton, and P. Kocher. The SSL Protocol, Version 3.0. Internet Draft. See:
`http://home.netscape.com/eng/ssl3/ssl-toc.html`.

[31] A. Fujioke, T. Okamoto, and S. Miyaguchi. ESIGN: an efficient digital signature implementation for smartcards In proceedings *Eurocrypt '91*, Lecture Notes in Computer Science, vol. 547, Springer-Verlag, pp. 446–457, 1991.

[32] M. Gardner. Mathematical games: *A new kind of cipher that would take millions of years to break.* Scientific American, 237(2):120–124, August 1997.

[33] C. F. Gauss. *Disquisitiones Arithmeticæ.* Leipzig, 1801.

[34] O. Goldreich. *Foundations of Cryptography – Fragments of a Book.*

[35] D. Gordon. *Discrete Logarithms in $GF(p)$ using the Number Field Sieve*, SIAM J. Discrete Math., Vol. 6, pp. 124–138, 1993.

[36] G. Hardy and E. Wright. *An Introduction to the Theory of Numbers.* Fourth Edition. Oxford Clarendon Press, 1975.

[37] J. Håstad. Solving simultaneous modular equations of low degree. *SIAM Journal on Computing*, vol. 17, no. 2, pp. 336–341, 1988.

[38] C. Hermite. Extraits de lettres de M. Hermite à M. Jacobi sur différents objets de la théorie des nombres, deuxième lettre. *J. Reine Agnew., Math.*, 40:279-290, 1850.

[39] N. Howgrave-Graham. Computational mathematics inspired by RSA. Ph.D. Thesis, University of Bath, 1999.

[40] N. Howgrave-Graham. Extending LLL to Gaussian integers. Unpublished Manuscript, March 1998.
`http://www.bath.ac.uk/~mapnahg/pub/gauss.ps`

[41] N. Howgrave-Graham. Finding small roots of univariate modular equations revisited. In proceedings *Cryptography and Coding*, Lecture Notes in Computer Science, vol. 1355, Springer-Verlag, pp. 131–142, 1997.

[42] N. Howgrave-Graham. Private communications, 2001.

[43] A. Joux and J. Stern. Lattice reductions: a toolbox for the cryptanalyst. *Journal of Cryptology*, vol. 11, no. 3, pp. 161–185, 1998.

[44] C. Jutla. On finding small solutions of modular multivariate polynomial equations. In proceedings *Eurocrypt '98*, Lecture Notes in Computer Science, vol. 1403, Springer-Verlag, pp. 158–170, 1998.

[45] D. Kahn. *The Codebreakers.* Scribner, New York, 1996.

[46] A. Kerckhoffs. La cryptographie militaire. *Journal des sciences militaires*, vol. IX, Jan.–Feb. 1883.

[47] P. Kocher. Timing attacks on implementations of Diffie-Hellman, RSA, DSS, and other systems. In proceedings *Crypto '96*, Lecture Notes in Computer Science, vol. 1109, Springer-Verlag, pp. 104–113.

[48] A. Korkine and G. Zolotareff. Sur les formes quadratiques. *Math. Ann*, 6:336-389, 1873.

[49] L. Lagrange. Recherches d'arithmétique. *Mouv. Mém. Acad.*, 1773.

[50] A. Lenstra and H. W. Lenstra Jr. Algorithms in Number Theory. In *Handbook of Theoretical Computer Science* (Volume A: Algorithms and Complexity), ch. 12, pp. 673–715, 1990.

[51] A. Lenstra and H. W. Lenstra Jr. The development of the number field sieve. Lecture Notes in Mathematics, vol. 1554, Springer-Verlag, 1994.

[52] A. Lenstra, H.W. Lenstra Jr, and L. Lovász. Factoring polynomials with rational coefficients. *Mathematische Annalen*, vol. 261, pp. 515–534, 1982.

[53] H. W. Lenstra Jr. Divisors in Residue Classes. *Mathematics of Computation*, vol. 42, no. 165, pp. 331-340, 1984.

[54] H. W. Lenstra Jr. Factoring integers with elliptic curves. *Annuals of Mathematics*, vol. 126, pp. 649–673, 1987.

[55] L. Lovász. An algorithmic theory of numbers, graphs, and convexity. *SIAM CBMS-NSF Regional Conference Series in Applied Mathematics*, vol. 50, 1986.

[56] Waterloo Maple. The Maple computational algebra system for algebra, number theory and geometry. Information available at: `http://www.maplesoft.com/products/Maple6/maple6info.html`.

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

[58] W. Meier. Private communication. June, 2000.

[59] H. Minkowski. *Geometrie der Zahlen.* Teubner-Verlag, Leipzig, 1896.

[60] P. Nguyen. Private communications.

[61] P. Nguyen and J. Stern. Lattice reduction in cryptology: an update. In *Algorithmic Number Theory – Proceedings of ANTS IV*, Lecture Notes in Computer Science, vol. 1838, Springer-Verlag, 2000.

[62] P. Nguyen and J. Stern. The Two Faces of Lattices in Cryptology. In proceedings *Cryptography and Lattices Conference*, Lecture Notes in Computer Science, vol. 2146, Springer-Verlag, 2001.

[63] I. Niven, H. Zuckerman, and H. Montgomery. An Introduction to the Theory of Numbers. Jon Wiley & Sons, Fifth Edition, pp. 87–88, 1991.

[64] T. Okamoto and S. Uchiyama. A new public key cryptosystem as secure as factoring. In proceedings *Eurocrypt '98*, Lecture Notes in Computer Science, vol. 1403, Springer-Verlag, pp. 310–318, 1998.

[65] R. Prealta and T. Okamoto. Faster factoring of integers of special form. IEICE Transactions Fundamentals, vol. E79-A, no. 4, pp. 489–493, 1996.

[66] W. Press, S. Teukolsky, W. Vetterling, and B. Flannery. Numerical Recipes in C: The Art of Scientific Computing. Second Edition. Cambridge University Press, pp. 347–393, 1997.

[67] J.J. Quisquater and C. Couvreur. Fast decipherment algorithm for RSA public-key cryptosystem. *Electronic Letters*, vol. 18, no. 21, pp. 905–907, 1982.

[68] D. Redmond. Number Theory: An Introduction. *Monographs and Textbooks in Pure and Applied Mathematics*, no. 201, Marcel Dekker, 1996.

[69] R. Rivest, A. Shamir, and L. Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Communications of the ACM*, vol. 21, no. 2, pp. 120–126, 1978.

[70] R. Sakai, M. Morii, and M. Kasahara. New key generation algorithm for RSA cryptosystem. *IEICE Trans. Fundamentals*, E77-A(1):89–97, 1994.

[71] A. Shamir. RSA for Paranoids. *RSA Laboratories CryptoBytes*, vol. 1, no. 3, pp. 1–4, 1995.

[72] V. Shoup. Number Theory Library (NTL), `http://www.shoup.net/ntl/`.

[73] R. Steinfeld and Y. Zheng. An advantage of low-exponent RSA with modulus primes sharing least significant bits. In Proceedings *RSA Conference 2001, Cryptographer's Track*, Lecture Notes in Computer Science, vol. 2020, Springer-Verlag, pp. 52–62, 2001.

[74] R. Silverman and S. Wagstaff. A Practical analysis of the elliptic curve factoring algorithm. *Mathematics of Computation*, vol. 61, 1993.

[75] D. Stinson. *Cryptography: Theory and Practice.* CRC Press, 1994.

[76] H. Sun, W. Yang, and C. Laih. On the design of RSA with short secret exponent. In proceedings *Asiacrypt '99*, Lecture Notes in Computer Science, vol. 1716, Springer-Verlag, pp. 150–164, 1999.

[77] T. Takagi. Fast RSA-type cryptosystem modulo $p^k q$. In proceedings *Crypto '98*, Lecture Notes in Computer Science, vol. 1462, Springer-Verlag, pp. 318–326, 1998.

[78] B. Vallée, M. Girault, and P. Toffin. How to guess ℓth roots modulo n by reducing lattice bases. In proceedings *AAEEC-6*, Lecture Notes in Computer Science, vol. 357, Springer-Verlag, pp. 427–442, 1988.

[79] E. Verheul and H. van Tilborg. Cryptanalysis of less short RSA secret exponents. *Applicable Algebra in Engineering, Communication, and Computing*, vol. 8, pp. 425–435, 1997.

[80] M. Wiener. Cryptanalysis of short RSA secret exponents. *IEEE Transactions on Information Theory*, vol. 36, no. 3, pp. 553–558, 1990.

[81] P. Zimmerman. Private communications.

[82] See `http://csrc.nist.gov/pki/`.

[83] See `http://www.cesg.gov.uk/`.

[84] See `http://www.setco.org/`.

Appendix A

Solutions to Modular Quadratic Equations

This appendix considers the problem of finding solutions for y in equations of the form

$$y^2 \equiv c \pmod{2^u}, \tag{A.1}$$

where $u \geq 3$ and $c \equiv 1 \bmod 8$.

We begin with an algorithm to find a solution to this equation presented in the following lemma. The proof appears in [68, p. 184]. For convenience we first define the following polynomial (with coefficients in $\mathbb{Q}$):

$$P_\nu(z) := \sum_{i=0}^{\nu} (-1)^{(i-1)} \frac{1}{4^i(2i-1)} \binom{2i}{i} z^i.$$

Lemma A.1.1 *Suppose $u \geq 3$ and $c \equiv 1$ mod 8. Then $y_0 := P_{u-3}(c-1)$ is an integer which satisfies $(y_0)^2 \equiv c$ mod 2^u.*

This provides one solution to equation (A.1). The following shows how to convert any solution to equation (A.1) into any other solution. We provide a proof different from what appears in [68].

Lemma A.1.2 *Assume y_0, y_1 are in the range $\{0, \ldots, 2^u - 1\}$, and suppose y_0 is a* *solution for y in equation (A.1). Then y_1 is a solution to (A.1) if and only if*

$$y_1 \equiv \sigma y_0 + \tau 2^{u-1} \pmod{2^u}$$

for some $\sigma \in \{-1, 1\}$ and $\tau \in \{0, 1\}$.

Proof Let $\sigma \in \{-1, 1\}$ and $\tau \in \{0, 1\}$ be arbitrary, and let y_0 be any solution to equation (A.1). If we define $y_1 := \sigma y_0 + \tau 2^{u-1}$ then we see

$$(y_1)^2 \equiv \sigma^2 (y_0)^2 + \sigma\tau 2^u y_0 + \tau^2 2^{2u-2} \pmod{2^u}.$$

Since $u \geq 3$, we have $2u - 2 \geq u$, so reducing modulo 2^u yields

$$(y_1)^2 \equiv \sigma^2 (y_0)^2 + \sigma\tau 2^u y_0 + \tau^2 2^{2u-2}. \equiv \sigma^2 (y_0)^2 \equiv (y_0)^2 \equiv c \pmod{2^u},$$

so y_1 is a solution for y in equation (A.1).

Now suppose that y_0 and y_1 are arbitrary solutions to equation (A.1). Since $c \equiv 1$ mod 8, it follows that $y_0^2 \equiv y_1^2 \equiv 1$ mod 4. So $y_1 \equiv \sigma y_0$ mod 4, where $\sigma \in \{-1, 1\}$. Thus there exists some integer v such that $y_1 = \sigma y_0 + 4v$. From equation (A.1) we see

$$y_1^2 \equiv (\sigma y_0 + 4v)^2 \equiv y_0^2 + 8\sigma y_0 v + 16v^2 \equiv c \pmod{2^u},$$

implying $8\sigma y_0 v + 16v^2 \equiv 0$ mod 2^u. So 2^u divides $8\sigma y_0 v + 16v^2 = 8v(\sigma y_0 + 2v^2)$. Now, $2v^2$ is even and y_0 is odd (recall $y_0^2 \equiv 1$ mod 8), so it must be the case that 2^{u-3} divides v. Hence, $y_1 = \sigma y_0 + 4v = \sigma y_0 + 2^{u-1}\tau$. Since we are interested in solutions modulo 2^u, all choices of τ are equivalent to either $\tau = 0$ or $\tau = 1$. ■

We are now ready to prove the main result of the Appendix.

Lemma A.1.3 *There are exactly four solutions for y in equation (A.1), and these solutions can be computed in time $O(u^3)$.*

The proof of the first clause is an immediate consequence of Lemmas A.1.1 and A.1.2. The running time analysis follows from the description of Lemma A.1.1. Note that it is especially efficient to evaluate $P_{u-3}(c-1)$ when only its result modulo 2^u is desired.

Appendix B

Approximate Square Roots

This appendix considers the problem of bounding the error of the square roots of approximated values. The main tool is the following lemma, needed for Theorem 5.2.4.

Lemma B.1.1 *With the notation as in Section 5.1.1, suppose* $|p - q| \geq 2^{(n/2)-2}$ *and* $e \leq 2^{(n/4)-3}$*. Furthermore suppose the quantities* d_1, s_1, p_1 *satisfy* $|d_1 - d| < 2^{n/4}$,

$$|s_1 - s| \leq 2^{(n/4)+\log_2 e}, \qquad p_1 = \frac{1}{2}(s_1 + \sqrt{s_1^2 - 4N}).$$

Then

$$|p_1 - p| \leq 2^{(n/4)+5+\log_2 e}.$$

The proof of this lemma follows a suggestion made by P. Nguyen [60].

Proof Without loss of generality, assume $s_1 \geq s$, so that $p_1 \geq p$. Observe

$$\begin{aligned} p_1 - p &= \frac{1}{2}(s_1 - s) + \frac{1}{2}(\sqrt{s_1^2 - 4N} - \sqrt{s^2 - 4N}) \\ &= \frac{1}{2}(s_1 - s) + \frac{(s_1 + s)(s_1 - s)}{2(\sqrt{s_1^2 - 4N} + \sqrt{s^2 - 4N})}. \end{aligned}$$

Since $s_1 \geq s$ we have $s_1^2 - 4N \geq s^2 - 4N = (p - q)^2$, and we know $(p - q) \geq 2^{(n/4)-2}$. The bound on e tells us $s_1 \leq 2s$, so that

$$s_1 \leq 2s = 2(p + q) \leq 4p \leq 2^{3+(n/2)}.$$

Using these facts we may derive the following bound:

$$\begin{aligned}
p_1 - p &\leq \frac{1}{2}(s_1 - s) + \frac{(s_1 + s)(s_1 - s)}{4(p - q)} \\
&\leq 2^{(n/4)-1+\log_2 e} + \frac{(6p)(2^{(n/4)+\log_2 e})}{2^{n/2}} \\
&\leq 2^{(n/4)-1+\log_2 e} + 2^{(n/4)+4+\log_2 e} \\
&\leq 2^{(n/4)+5+\log_2 e}
\end{aligned}$$

as desired. ■

Appendix C

Determinants for Short Secret Exponent

We give the exact expressions evaluating to the determinant of the lattice described in Section 6.2. We know

$$\begin{aligned}\det_x &= e^{m(m+1)(m+2)(4\delta-1)/12} \\ \det_y &= e^{tm(m+1)(\delta-1)/2+t(m+1)(m+t+1)/4}\end{aligned}$$

The determinant of the entire lattice is $\det_x \cdot \det_y$ and its dimension is $w = (m+1)(m+2)/2 + t(m+1)$.

To satisfy $\det(L) = \det_x \cdot \det_y \ll 1$ we must have

$$m(m+1)(m+2)\frac{4\delta-1}{12} + tm(m+1)\frac{\delta-1}{2} + \frac{t(m+1)(m+t+1)}{4} < 0,$$

which leads to

$$m(m+2)(-1+4\delta) + 3tm(-1+2\delta) + 3t(t+1) < 0$$

For every m the left hand side is minimized at $t = \frac{m(1-2\delta)-1}{2}$. Plugging this value in leads to:

$$-(3+2m+7m^2) + \delta(28m^2+20m) - 12m^2\delta^2 < 0$$

implying

$$\delta < \frac{7}{6} - \frac{1}{3}\sqrt{7 + \frac{16}{m} + \frac{4}{m^2}} + \frac{5}{6m}$$

As was shown in Section 6.2, when m goes to infinity this values converges to

$$\delta < \frac{7}{6} - \frac{\sqrt{7}}{3} \approx 0.2847$$

For a particular value of $\delta < 0.2847$ we must take m to be at least

$$m > \frac{-1 + 10\delta + 2(-5 + 16\delta + 16\delta^2)^{1/2}}{7 - 28\delta + 12\delta^2}$$

For example, when $\delta = 0.27$ we must take $m \geq 10$ leading to a lattice of dimension 86.

Appendix D

Proof of Theorem 6.3.1

We use the following approach. First we introduce the notion of *diagonally dominant* matrices, and show that there is an easy bound on the determinant of any lattice formed from a subset of the rows of a diagonally dominant matrix M. We then show that for certain submatrices of geometrically progressive matrices there is a unitary transformation over $\mathbb{R}$ that puts the submatrix into a diagonally dominant form, giving the desired determinant bounds. We then verify that these bounds yield the conclusion of Theorem 6.3.1.

Let M be an $n \times n$ triangular matrix with rows $u_1, \ldots, u_n$. We say that M is *diagonally dominant to within a factor* C when $|u_{i,j}| \leq C \cdot |u_{i,i}|$ for all $i, j = \{1, \ldots, n\}$. When the factor C is understood, we say simply that M is *diagonally dominant*.

Let S be a subset of the row indices. We define M_S to be the $|S| \times n$ matrix whose rows are $u_i, i \in S$. We say that an arbitrary $w \times n$ matrix $\tilde{M}$ is diagonally dominant when there is a $S \subseteq \{1, \ldots, n\}$ and diagonally dominant matrix M such that $\tilde{M} = M_S$ and $|S| = w$. We say that a lattice L is diagonally dominant when there is a basis $u_1, \ldots, u_w$ for L such that the matrix with rows $u_1, \ldots, u_w$ is diagonally dominant. Diagonally dominant lattices have determinants that are easy to bound, as shown in the following fact.

Fact D.1.1 *Let $w \leq n$ be given and take $S \subseteq \{1, \ldots, n\}$ with $|S| = w$. If L is a* *lattice spanned by the rows u_i for $i \in S$ of a diagonally dominant matrix M, then*

$$\det(L) \leq n^{w/2} C^w \prod_{i \in S} |u_{i,i}|.$$

Proof Observe that since $\|u_i^*\| \leq \|u_i\|$ we have:

$$\det(L) = \prod_{i \in S} \|u_i^*\| \leq \prod_{i \in S} \|u_i\| \leq \prod_{i \in S} \sqrt{n} C |u_{i,i}| = n^{w/2} C^w \prod_{i \in S} |u_{i,i}|. \qquad \blacksquare$$

Now let M be an $(a+1)b \times (a+1)b$ geometrically progressive matrix. We have a bound of $C \cdot D^{c_1 i + c_2 j + c_3 k + c_4 \ell}$ for the entry at row (k, ℓ) and column (i, j); if we knew this was always less than C times the bound $D^{c_1 k + c_2 \ell + c_3 k + c_4 \ell}$ of that row's entry on the diagonal, then by conditions **(i)** and **(ii)** on geometrically progressive matrices we would have that M is diagonally dominant to within a factor C.

Unfortunately, this is not always the case. We call a column index (i, j) *bad* when the following condition holds:

$$D^{c_1 i + c_2 j + c_3 k + c_4 \ell} > D^{c_1 k + c_2 \ell + c_3 k + c_4 \ell},$$

or equivalently, $c_1(k - i) + c_2(\ell - j) < 0$. It should be noted that the "badness" of a column is a statement about the *bound* on the entry in the column, which is a function of the *parameters* of the geometrically progressive matrix, not of the entry itself. Indeed, the actual entry $M(i, j, k, \ell)$ of a bad column (i, j) could be zero, leading us to the following observation.

Remark D1. Let M be a geometrically progressive matrix and S a subset of the rows. If $M(i, j, k, \ell) = 0$ for every bad column (i, j) of every row $(k, \ell) \in S$, then M_S is diagonally dominant to within a factor C. This is because for each (i, j) that is not bad in the row (k, ℓ), we have

$$M(i, j, k, \ell) \leq C \cdot D^{c_1 i + c_2 j + c_3 k + c_4 \ell} \leq C \cdot D^{c_1 k + c_2 \ell + c_3 k + c_4 \ell} = C \cdot M(k, \ell, k, \ell).$$

Remark B1 suggests that we should be looking for a submatrix M_S whose entries are zero in bad columns. Although this is unlikely for any submatrix M_S of the matrix M developed in Section 6.2, what we shall see is that there is a unitary transformation over $\mathbb{R}$ that eliminates entries at bad columns in rows of M_S. Once the diagonal dominance of this transformed submatrix has been established, Fact D.1.1 can then be employed to bound the determinant of the corresponding lattice.

Our goal now is to show that special submatrices of geometrically progressive matrices can be put into a diagonally dominant form. Consider the following situation: suppose we take a subset S of the rows of a geometrically progressive matrix M and wish to bound the determinant of the lattice described by M_S. We wish to guarantee that there are "enough" rows included in S so that we may eliminate all nonzero entries at bad columns in rows of M_S. We prove this for certain natural subsets S in Lemma D.1.2. We then use this guarantee to show that such an elimination procedure will be successful; namely, we show that there is a unitary transformation U over $\mathbb{R}$ such that $U \cdot M_S$ is diagonally dominant. This is shown in Lemma D.1.3, leading directly to a proof of Theorem 6.3.1.

Lemma D.1.2 *Let M be an $(a+1)b \times (a+1)b$ geometrically progressive matrix with parameters (C, D, c_1, c_2, c_3, c_4, β), let $B \in \mathbb{R}$ be a constant. Define*

$$S_B := \{(k,\ell) \in \{0,\ldots,a\} \times \{1,\ldots,b\} \mid M(k,\ell,k,\ell) \le B\}.$$

For all $(k,\ell) \in S_B$ and $i \le k, j \le \ell$, if column (i,j) is bad in row (k,ℓ) then $(i,j) \in S_B$.

Proof We begin by assuming that (i,j) is bad, so $D^{c_1(k-i)+c_2(\ell-j)} < 1$ and thus

$$D^{(\beta-1)c_1(k-i)+(\beta-1)c_2(\ell-j)} = \left(D^{c_1(k-i)+c_2(\ell-j)}\right)^{(\beta-1)} \le 1. \tag{D.1}$$

Seeking contradiction, we now assume $(i,j) \notin S_B$, that is, $M(i,j,i,j) > B$. It follows that

$$D^{(c_1+c_3)i+(c_2+c_4)j} = M(i,j,i,j) > B \ge M(k,\ell,k,\ell) = D^{(c_1+c_3)k+(c_2+c_4)\ell}.$$

Hence

$$D^{(c_1+c_3)(k-i)+(c_2+c_4)(\ell-j)} < 1. \tag{D.2}$$

Combining equations (D.1) and (D.2) yields

$$D^{(\beta c_1+c_3)(k-i)+(\beta c_2+c_4)(\ell-j)} < 1. \tag{D.3}$$

Note that $i \leq k$ and $j \leq \ell$ by the hypotheses of the theorem, and we are guaranteed $\beta c_1 + c_3 \geq 0$ and $\beta c_2 + c_4 \geq 0$ since M is geometrically progressive. So $(\beta c_1 + c_3)(k - i) + (\beta c_2 + c_4)(\ell - j) \geq 0$. Furthermore, $D \geq 1$, so

$$D^{(\beta c_1+c_3)(k-i)+(\beta c_2+c_4)(\ell-j)} \geq D^0 = 1,$$

contradicting equation (D.3). Hence, $(i, j) \in S_B$ as desired. ∎

Lemma D.1.3 *Let M be an $(a+1)b \times (a+1)b$ geometrically progressive matrix with parameters (C, D, c_1, c_2, c_3, c_4, β), let $B \in \mathbb{R}$ be a constant, define*

$$S_B := \{(k,\ell) \in \{0,\ldots,a\} \times \{1,\ldots,b\} \mid M(k,\ell,k,\ell) \leq B\},$$

and set $w := |S_B|$. There is a $w \times w$ unitary matrix U over $\mathbb{R}$ such that $U \cdot M_{S_B}$ is diagonally dominant to within a factor $(1+C)^w$.

Proof We proceed by induction. There are w rows in the matrix M_{S_B}, and we build matrices U_r such that the last r rows of $U_r \cdot M_{S_B}$ are diagonally dominant[1] to within a factor $(1+C)^w$, and the first $w-r$ rows identical to those in M_{S_B}. The U we seek is U_w.

Clearly, $U_0 = I$ trivially satisfies this condition. Now suppose we have a unitary matrix U_{r-1} over $\mathbb{R}$ such that the last $r-1$ rows of $U_{r-1} \cdot M_{S_B}$ are diagonally dominant to within a factor $(1+C)^w$ and the first $w-r$ rows are identical to those of M_{S_B}. We would like to find U_r that satisfies this condition for the last r rows, and we do this by finding a unitary matrix V over $\mathbb{R}$ such that $U_r := V \cdot U_{r-1}$ satisfies this condition.

[1] To say that the last r rows of a $w \times n$ matrix $\tilde{M}$ are diagonally dominant means simply that $\tilde{M}_{\{(w-r+1),\ldots,w\}}$ is diagonally dominant.

Roughly speaking, the purpose of V is to "clean up" row $(w-r+1)$ of M_{S_B}; that is, it guarantees that $(1+C)^w$ times the last column of row $(w-r+1)$ dominates all other columns of row $(w-r+1)$ in $V \cdot U_{r-1} \cdot M_{S_B}$.

Since M_{S_B} is formed from rows of M, we may choose a pair (k,ℓ) such that row $(w-r+1)$ of M_{S_B} is the (k,ℓ)th row of M. By Lemma D.1.2, for every bad column (i,j) satisfying $i \leq k$ and $j \leq \ell$, the corresponding row (i,j) is in S_B. So there are at most $w-1$ bad columns with nonzero entries in the row (clearly, (k,ℓ) is not bad.)

We build V in stages by constructing elementary row operations $V_1,\ldots,V_{w-1}$ and letting $V := V_{w-1} \cdot V_{w-2} \cdots V_1$. Each V_s sets another bad column (i_s,j_s) in the row to 0, so that the $(w-r+1)$th row of $V_s \cdots V_1 \cdot U_{r-1} \cdot M_{S_B}$ has nonzero entries in at most $w-s-1$ bad columns. We show that each V_s increases every column of the row by at most a factor of $(1+C)$.

Define

$$v^{(s)} := (V_s \cdots V_1 \cdot U_{r-1} \cdot M_{S_B})_{\{w-r+1\}},$$

that is, $v^{(s)}$ is the $(w+r-1)$th row of $V_s \cdots V_1 \cdot U_{r-1} \cdot M_{S_B}$. We denote the entry in the (i,j)th column of $v^{(s)}$ as $v^{(s)}(i,j)$. We maintain the following three invariants for $s=1,\ldots,w-1$:

(i) $\left|v^{(s)}(i,j)\right| \leq (1+C)^s C \cdot D^{c_1 i + c_2 j + c_3 k + c_4 \ell}$ for all columns (i,j);

(ii) $i > k$ or $j > \ell$ implies $v^{(s)}(i,j) = 0$; and,

(iii) the number of bad columns with nonzero entries in $v^{(s)}$ is at most $w-s-1$.

These conditions are satisfied trivially for $s=0$, since $v^{(0)}$ is identical to row (k,ℓ) of the geometrically progressive matrix M. Now suppose that every column (i,j) of $v^{(s-1)}$ satisfies these three conditions. If there are no nonzero entries of $v^{(s-1)}$ at bad columns, we are done, and may take $V_s,\ldots,V_{w-1} := I$. Otherwise, let (i_s,j_s) be the rightmost bad column such that $v^{(s-1)}(i_s,j_s) \neq 0$. Since $v^{(s-1)}(i_s,j_s) \neq 0$, we know by the inductive hypothesis that $i_s \leq k$ and $j_s \leq \ell$. Since (i_s,j_s) is also bad, we know that $(i_s,j_s) \in S_B$. So we may pick a t such that row (i_s,j_s) of M is row t of M_{S_B}. Define V_s to be the elementary row operation that subtracts $\frac{v^{(s-1)}(i_s,j_s)}{M(i_s,j_s,i_s,j_s)}$ times row t

from row $(w-r+1)$. Observe for every column (i,j),

$$\begin{aligned}
\left|v^{(s)}(i,j)\right| &\leq \left|v^{(s-1)}(i,j)\right| + \left|\frac{v^{(s-1)}(i_s,j_s)}{M(i_s,j_s,i_s,j_s)} \cdot M(i,j,i_s,j_s)\right| \\
&\leq (1+C)^{s-1}C \cdot D^{c_1 i + c_2 j + c_3 k + c_4 \ell} \\
&\quad + \frac{(1+C)^{s-1}C \cdot D^{c_1 i_s + c_2 j_s + c_3 k + c_4 \ell}}{D^{c_1 i_s + c_2 j_s + c_3 i_s + c_4 j_s}} \cdot C \cdot D^{c_1 i + c_2 j + c_3 i_s + c_4 j_s \ell} \\
&= (1+C)^s C \cdot D^{c_1 i + c_2 j + c_3 k + c_4 \ell}.
\end{aligned}$$

So condition **(i)** is met.

Now let (i,j) be given with either $i > k$ or $j > \ell$. Since $v^{(s-1)}(i_s,j_s) \neq 0$, we know by condition **(ii)** of the inductive hypothesis that $i_s \leq k$ and $j_s \leq \ell$. So either $i > k \geq i_s$ or $j > \ell \geq j_s$, implying $M(i,j,i_s,j_s) = 0$. Thus

$$v^{(s)}(i,j) = v^{(s-1)}i,j - \frac{v^{(s-1)}(i_s,j_s)}{M(i_s,j_s,i_s,j_s)} \cdot M(i,j,i_s,j_s) = 0 - 0 = 0,$$

satisfying condition **(ii)**.

We now claim that the number of bad columns with nonzero entries in $v^{(s)}$ is at most $w-s-1$. Clearly, $v^{(s)}(i_s,j_s) = 0$, and columns to the right of (i_s,j_s) are unchanged from $v^{(s-1)}$. Since (i_s,j_s) was chosen to be the rightmost nonzero bad column of $v^{(s-1)}$, this implies that no nonzero column in $v^{(s)}$ to the right of (i_s,j_s) is bad. But since this is the sth elimination step, there are at least $s-1$ bad columns (i,j) to the right of (i_s,j_s) satisfying $i \leq k$ and $j \leq \ell$. Thus, the number of bad columns with nonzero entries in $v^{(s)}$ is at most $w-s-1$, satisfying condition **(iii)**.

Thus, $v^{(w-1)}$ has a zero in every bad column, so

$$v^{(w-1)}(i,j) \leq (1+C)^{w-1}C \cdot D^{c_1 i + c_2 j + c_3 k + c_4 \ell} \leq (1+C)^w \cdot M(k,\ell,k,\ell)$$

for all columns (i,j). Setting $V := V_{w-1} \cdots V_1$ and $U_r := V \cdot U_{r-1}$, we have that the last r rows of $U_r \cdot M_{S_B}$ are diagonally dominant to within a factor $(1+C)^w$. Finally, taking $U := U_w$ completes the result. ■

We are now ready to complete the proof of Theorem 6.3.1.

Proof of Theorem 6.3.1 By Lemma D.1.3 we have a $w \times w$ unitary matrix U

over $\mathbb{R}$ such that $U \cdot M_{S_B}$ is diagonally dominant to within a factor $(1+C)^w$. Since U is unitary over $\mathbb{R}$, the lattice L' induced by the rows of $U \cdot M_{S_B}$ has the same determinant as the lattice L induced by the rows of M_{S_B}, so by Fact D.1.1 yielding the desired bound

$$\det(L) = \det(L') \leq ((a+1)b)^{w/2}(1+C)^{w^2} \prod_{(k,\ell)\in S_B} M(k,\ell,k,\ell),$$

where $w = |S_B|$. ∎

Appendix E

Determinants for the Sun-Yang-Laih Schemes

The general formula for the determinant of the lattice we build in Section 7.2 is

$$\det(L) = e^{C_e} X^{C_x} Y^{C_y} Z^{C_z},$$

where

$$\begin{aligned}
C_e &= -\frac{1}{6}m(m+1)(2m+3t+1),\\
C_x &= \frac{1}{6}m(m+1)(4m+3t+5),\\
C_y &= \begin{cases} \frac{1}{6}(m^3+3(a+t+1)m^2+(3t^2+6at+3a^2+6a+6t+2)m \\ \qquad\qquad +(3t^2+6at+3a^2+4a+3t-a^3)) & \text{if } a \geq 0,\\ \frac{1}{6}(m^3+3(a+t+1)m^2+(3t^2+6at+3a^2+6a+6t+2)m \\ \qquad\qquad +(3t^2+6at+3a^2+3a+3t)) & \text{if } a < 0,\end{cases}\\
C_z &= \begin{cases} \frac{1}{6}(m^3-3(a-1)m^2+(3a^2-6a+2)m+(3a^2-2a-a^3)) & \text{if } a \geq 0,\\ \frac{1}{6}(m^3-3(a-1)m^2+(3a^2-6a+2)m+(3a^2-3a)) & \text{if } a < 0.\end{cases}
\end{aligned}$$

We need $\det(L) \ll 1$. In order to optimize the choice of t and a, we write $t = \tau m$ and $a = \alpha m$, and observe

$$\begin{aligned}
C_e &= -\frac{1}{6}(3\tau + 2)m^3 + o(m^3), \\
C_x &= \frac{1}{6}(3\tau + 4)m^3 + o(m^3), \\
C_y &= \begin{cases} \frac{1}{6}(3\tau^2 + 6\alpha\tau + 3\alpha^2 + 3\alpha + 3\tau + 1 - \alpha^3)m^3 + o(m^3) & \text{if } \alpha \geq 0, \\ \frac{1}{6}(3\tau^2 + 6\alpha\tau + 3\alpha^2 + 3\alpha + 3\tau + 1)m^3 + o(m^3) & \text{if } \alpha < 0, \end{cases} \\
C_z &= \begin{cases} \frac{1}{6}(3\alpha^2 - 3\alpha + 1 - \alpha^3)m^3 + o(m^3) & \text{if } \alpha \geq 0, \\ \frac{1}{6}(3\alpha^2 - 3\alpha + 1)m^3 + o(m^3) & \text{if } \alpha < 0. \end{cases}
\end{aligned}$$

Suppose we write $e = N^{\varepsilon}$, $d = N^{\delta}$, and $Y = N^{\beta}$, so $Z = N^{1-\beta}$. Then $X = N^{\varepsilon+\delta-1}$. So the requirement on $\det(L)$ now becomes

$$\varepsilon C_e + (\varepsilon + \delta - 1)C_x + \beta C_y + (1 - \beta)C_z < 0. \tag{E.1}$$

The left-hand-side of this expression achieves its minimum at

$$\begin{aligned}
\tau_0 &= (2\alpha_0\beta - \beta - \delta + 1)/(2\beta), \\
\alpha_0 &= \begin{cases} 1 - \beta - (1 - \beta - \delta + \beta^2)^{(1/2)} & \text{if } \beta < \delta, \\ (\beta - \delta)/(2\beta - 2) & \text{if } \beta \geq \delta. \end{cases}
\end{aligned}$$

Using $\tau = \tau_0$ and $\alpha = \alpha_0$ will give us the minimum value on the left-hand-side of inequality E.1, affording us the largest possible X to give an attack on the largest possible $d < N^{\delta}$. The entries in Figure 7.2 were generated by plugging in τ_0 and α_0 and solving for equality in Equation E.1.

It is interesting to note that formulation of the root-finding problem for RSA as a trivariate equation is strictly more powerful than its formulation as the small inverse problem. This is because the small inverse problem is not expected to have a unique solution once $\delta > 0.5$, while this attack works in many cases with $\delta > 0.5$. We note that when $\varepsilon = 1$ and $\beta = 0.5$ – as in standard RSA – this attack gives the same results as the simpler short secret exponent attack outlined in Section 6.2 ($d < N^{0.284}$). The additional optimizations that come from using lattices of less than full rank (Section 6.3) should also work with this approach, but the improvement would likely be minimal.