Lattice-based Cryptanalysis using Unravelled Linearization

Dissertation

zur Erlangung des Doktorgrades der Naturwissenschaften der Fakultät für Mathematik der Ruhr-Universität Bochum

vorgelegt von

Dipl.-Inform. Mathias Herrmann
Januar 2011
1 Introduction 5

2 Preliminaries 13

3 Finding Small Solutions of Modular Equations 17
   3.1 Coppersmith's Algorithm 18
   3.2 Jochemsz-May Strategy 20
   3.3 Linear Equations 23
      3.3.1 Linear Equations with Known Modulus 24
      3.3.2 Bivariate Linear Equations modulo Divisors 26
      3.3.3 Multivariate Linear Equations 31
   3.4 Coppersmith's Algorithm: Integer Variant 40
      3.4.1 Jochemsz-May Strategy for Small Integer Roots 44
      3.4.2 Embedding the Modular Case 46

4 Unravelled Linearization 49
   4.1 General Strategy 52

5 Pseudo Random Number Generators 57
   5.1 Blum-Blum-Shub Generator 58
   5.2 Unravelling Pseudo Random Number Generators 59
   5.3 Using More Equations 62
      5.3.1 Unravelled Linearization 63
      5.3.2 Arbitrary Number of Iterations 65
   5.4 Experiments 70
   5.5 RSA-based Generator 70

6 Attacks on RSA 77
   6.1 The RSA Cryptosystem 78
   6.2 Attacking RSA with Small Secret Exponent \( d \) 79
      6.2.1 The Boneh-Durfee Attack 79
      6.2.2 Unravelling the Boneh-Durfee attack 83
   6.3 Attack on RSA-CRT 86
      6.3.1 Jochemsz-May Attack 87
      6.3.2 Unravelling Jochemsz-May attack 87
Acknowledgements

First of all, I would like to thank Prof. Dr. Alexander May for giving me the opportunity to write a thesis under his supervision. He taught me how to handle research problems and his support in finding solutions, writing research papers and presenting them at conferences was outstanding. With his enthusiastic presentation of new problems and ideas of possible solutions, he always managed to provide me with the necessary motivation. Thank you so much - I cannot imagine a better boss, supervisor and mentor.

Another very important role was played by Maike Ritzenhofen, who shared an office with me during my time at the university. We had plenty of wonderful discussions and she always listened to my problems and ideas. She not only was a coworker but also became a very good friend. Thank you, Maike.

I would also like to thank all the friends I made in Bochum, most of all Tibor Jager, for all the coffee he invested in me, for all the time he spent listening to my problems and for all the fun that we had playing tennis, squash or just going out. Furthermore, thanks to Sven Schäge, Juraj Somorovsky, Gregor Leander, Saqib Kakvi, Alex Meurer, Enrico Thomae, Florian Kohlar and everyone else that I forgot to mention.

For proofreading the thesis I am very grateful to Gottfried Herold for suggesting an elegant way to simplify the notation in Section 5.3. I also thank Maike Ritzenhofen, Enrico Thomae and Alex Meurer for spending plenty of time proofreading this thesis.

For all the help and support in bureaucratic and organizational issues, I would like to thank our secretary Marion Reinhardt-Kalender.

I further owe a debt of gratitude to the German Research Foundation (DFG) for financially supporting my thesis for three years and also the Günther und Wilhelm Esser Foundation for providing me a stipend to finalize my thesis.

Finally, I am very thankful to my girlfriend Julia Bührig-Neupert, who gave me amazing support during the preparation of the final examination.
Chapter 1

Introduction

Cryptographic methods have been used by mankind since the times of the ancient Romans and Greek. The famous cipher of Caesar or the scytale used by the Spartans show that keeping information secret has been a concern for thousands of years. While enciphering messages was almost exclusively used in military contexts, this has tremendously changed over the last decades. In modern society, cryptography is ubiquitous. Nowadays everybody gets in contact with cryptographic methods in his daily life. For example, each time an automatic teller machine is used or a call is made using a cell phone or an item is bought in an online shop, cryptographic algorithms are employed to guarantee desired properties like confidentiality, authenticity or privacy.

Although modern cryptography has many different aspects, its primary goal is still to encrypt a message $m$ in such a way that a receiver is only able to decrypt the encrypted message if he possesses the correct key. More formally, an encryption function $E$ takes a message $m$ from a plaintext space $P$ and a key $k$ from a key space $K$, and maps to a ciphertext $c \in C$, where $C$ is a ciphertext space. The corresponding decryption function $D$ takes the ciphertext $c$ and the same key $k$, and maps them back to the original message $m$. This is the easiest concept to design an encryption scheme. It is called symmetric encryption because sender and receiver use the same key $k$ for encryption and decryption.

A major drawback in symmetric schemes is the problem of key exchange. Note that in order to securely communicate, the sender and receiver have to exchange the key $k$ first. The need for a key exchange has been eliminated in a ground breaking work by Diffie and Hellman [DH76], who introduced the concept of asymmetric encryption schemes. In an asymmetric encryption scheme there exist two keys, a public key $pk$ and a private key $sk$, which are used for different purposes. The public key can only be used to encrypt messages while for the decryption of a ciphertext the secret key is required. Thus, we no longer have the need to securely exchange a key, because $pk$ can be made public since it cannot be used to decrypt messages. The secret key, as the name suggests, has to be kept secret. For example, suppose Alice wishes to send an encrypted message to Bob. Then she uses Bob’s public key to encrypt the message $m$ and sends the resulting ciphertext $c$ to Bob. Bob, on the other hand, uses his secret key to decrypt $c$. An eavesdropper Eve, who might intercept the message in transit, cannot use the public key $pk$ to learn any information about the message $m$ from the ciphertext $c$. 
Introduction

A famous asymmetric encryption scheme has been presented by Rivest, Shamir and Adleman [RSA78]. Thirty years after its invention, the RSA public key encryption scheme is still one of the most commonly used encryption schemes. It basically works by selecting two primes \( p, q \) of equal bit size and computing their product \( N = pq \). A pair consisting of a public exponent \( e \) and a private exponent \( d \) is chosen, such that the property \( ed = 1 \mod \varphi(N) \) holds, where \( \varphi(N) = (p-1)(q-1) \) is Euler’s totient function. The public key of the scheme is \( pk = (N, e) \) and the private key is \( sk = (N, d) \). Now encryption takes a public key \( pk = (N, e) \) and a message \( m \in \mathbb{Z}_N \) and computes the ciphertext \( c \in \mathbb{Z}_N \) by raising \( m \) to the \( e \)-th power modulo \( N \).

\[
c = m^e \mod N
\]

The decryption function similarly takes the secret key \( sk = (N, d) \) and a ciphertext \( c \) and computes

\[
c^d = (m^e)^d = m^{ed} = m \mod N.
\]

The security of the RSA encryption scheme is based on the difficulty of computing \( m \) from \( c = m^e \mod N \) without knowing the secret exponent \( d \). I.e. it is a difficult problem to compute \( e \)-th roots modulo an integer \( N \) of unknown factorization. Note that if the factorization of \( N \) is known, then it is easy to compute \( e \)-th roots, since we are able to compute the value of \( d \) as the inverse of \( e \) modulo \( \varphi(N) \). Thus, an efficient factorization algorithm implies an efficient ‘\( e \)-th root modulo \( N \)’ algorithm. The inverse direction of this implication is not known to be true. The assumption that in general finding an \( e \)-th root modulo \( N \) is hard, is widely known as RSA assumption.

For certain inputs, however, the RSA problem, i.e. the problem of computing an \( e \)-th root modulo \( N \), is not difficult. Consider for example the ciphertext \( c = m^e \mod N \), where we know that \( m < N^{\frac{1}{2}} \) is rather small. Then \( m^e \) is smaller than \( N \) and it holds \( c = m^e \) over the integers. This problem is easy to solve using for example the Newton method to find a solution of the equation \( c - m^e = 0 \) efficiently. It is a natural question to ask under which circumstances we are able to find small solutions to an arbitrary polynomial equation \( f(x) = 0 \mod N \). To this end, Coppersmith [Cop96b] presented an algorithm that efficiently finds all solutions of \( f(x) = 0 \mod N \) which are in absolute value smaller than \( N^{\frac{1}{2}} \), where \( \delta \) is the degree of \( f \).

An extension of Coppersmith’s algorithm can be used to solve equations modulo divisors, i.e. \( f(x) = 0 \mod p \), where \( p \) is a unknown divisor of a known integer \( N \). This view on the algorithm emerged from a line of research that considers relaxations of the factorization problem. Recall that the factorization problem gets as input a composite number \( N \) and asks for a prime factor of \( N \). We already saw that the RSA problem is solvable, if the factorization problem is solvable. Thus, it is natural to investigate the complexity of the factorization problem when talking about the security of the RSA cryptosystem.

It is well known that the factorization problem is polynomial time solvable on a quantum Turing machine [Sho97]. However, on classical Turing machines the best known algorithms [LLMP90, Len87] require super polynomial running time.
Because there has been little progress in the design of better factorization algorithms, a new line of research established, which considers relaxations of the factorization problem that are solvable in polynomial time. These relaxations can be modeled by giving access to certain oracles and the goal is to make as few queries to the oracle as possible. For example, Rivest and Shamir considered in [RS85] an oracle that reveals bits of a prime factor $p$ of an RSA modulus $N = pq$. They showed that $\frac{2}{3} \log p$ queries are sufficient to factor $N$ efficiently. Their result has been improved by Coppersmith [Cop96a] to require only $\frac{1}{2} \log p$ queries. In the original work this result was obtained by finding a small solution of a bivariate equation over the integers. However, Howgrave-Graham [HG97] showed that this result can also be formulated by considering small solutions of a univariate polynomial modulo an unknown divisor.

Recently, Heninger and Shacham [HS09] and later Henecka, May and Meurer [HMM10] presented a practically motivated new oracle approach which does not reveal any bits with certainty. Instead, the oracle outputs an erroneous version of the prime factors, where every bit is correct only with a certain probability. Another oracle-based approach where the output from the oracle does not contain any explicit information at all has been presented by May and Ritzenhofen [MR09].

Besides analyzing the factorization problem with an oracle-based approach, we can similarly investigate relaxations of the RSA problem itself. Coppersmith showed in [Cop96b] that $(1 - \frac{1}{e}) \log N$ queries to an oracle revealing bits of the $e$-th root of $c = m^e \mod N$ are sufficient to efficiently compute the entire $m$. More surprisingly, Franklin and Reiter [FR95] use an oracle that on input $c = m^3 \mod N$ outputs $c' = (m + r)^3 \mod N$ for a random value $r \leq N^{\frac{1}{9}}$ to efficiently recover $m$. A generalization to arbitrary $e$ is given in [CFPR96].

A related oracle is motivated by RSA-based pseudo random number generators. The oracle contains an internal state $s_i$ which is initialized by a random seed $s_0$. On a query it updates the internal state by computing $s_i = s_{i-1}^2 \mod N$ and outputs a $(1 - \delta)$-fraction of the bits of $s_i$. The problem is to recover the internal state $s_n$ given $n$ queries to the oracle. Such an oracle has been investigated by Blackburn, Gomez, Gutierrez and Shparlinski in [BGPGS05]. However, instead of using the RSA function to update the internal state, they consider the function $s_i = s_{i-1}^2 \mod p$ for a prime $p$. Nevertheless, their algorithm similarly works for an RSA-based generator. They showed that two queries which output a $\frac{2}{3}$-fraction of the bits of consecutive internal states are sufficient to recover the entire internal state. This result has been improved by Gomez, Gutierrez and Ibeas [IGGP06] to require only an output of a $\frac{9}{14}$-fraction of the internal state bits.

Besides investigating the oracle complexity of the factorization problem or the RSA problem, it is important to analyze the security of a cryptographic system when instantiated with specific parameters. For various reasons, keys are in practice often not chosen randomly but in such a way that they have a certain property. For example, in the RSA cryptosystem it is a major concern to perform the encryption and decryption operations as efficient as possible. In particular, suppose the secret key $sk = (N, d)$ is securely stored on a resource constrained smart card and the decryption of a ciphertext is completely handled by the small smart card processor. In order to keep the workload low, the secret exponent $d$ could be chosen small. However, in 1990 it was shown by Wiener [Wie90] that it is possible to efficiently
reconstruct $d$ from the public key $(N, e)$ if $d$ is chosen smaller than $N^{\frac{1}{4}}$. Using the algorithm of Coppersmith to find small solutions of modular equations, this result has been further improved by Boneh and Durfee [BD00] to secret keys $d$ smaller than $N^{0.284}$. In the same publication, Boneh and Durfee presented a sophisticated analysis to further improve the exponent to 0.292.

A different possibility to speed up the decryption is based on the Chinese Remainder Theorem (CRT) and dates back to Quisquater and Couvreur [QC82]. The observation is that the party performing the decryption usually has knowledge of the factorization of the modulus $N$. Given the prime factors $p$ and $q$ of $N$, the decryption can be performed by computing $m_p = c^{d_p} \mod p$ and $m_q = c^{d_q} \mod q$, where $d_p = d \mod p - 1$ and $d_q = d \mod q - 1$. The final result of the decryption modulo $N$ is derived from the partial decryptions $m_p, m_q$ using the Chinese Remainder Theorem. Efficiency is gained since the exponentiations are performed with much smaller numbers. Wiener [Wie90] proposed instead of using a small value of $d$ one should use small values $d_p, d_q$ to further increase the efficiency. In 2006, Bleichenbacher and May [BM06] showed that using small CRT exponents $d_p, d_q < N^{\delta}$ admits a polynomial time attack for a certain $\delta$. However, the value of $\delta$ depends on the size of the involved public exponent $e$ and their attack does not work if $e$ is full size, i.e. of the size of $N$. Their result has been improved by Jochemsz and May [JM07] who proposed a polynomial time attack which works for full sized $e$ as long as $\delta < 0.073$. The latter result is based on the algorithm of Coppersmith to find small solutions of multivariate equations over the integers.

The contribution in this thesis is twofold. On one hand, we analyze the problem of finding solutions to modular equations. In particular, we are interested in the problem of finding small solutions modulo divisors. Motivated by the oracle-based factorization approach, we consider a weaker oracle which no longer allows queries for specific bits but instead outputs bits at random (known) locations. On the other hand, we pursue the optimization of certain attacks on pseudo random number generators and the RSA cryptosystem which rely on the algorithm of Coppersmith. We introduce a new technique called unravelled linearization which allows to develop new attacks, as well as, simplify the proofs of previous ones.

In more detail, the thesis is organized as follows.

Chapter 2
Since most of the methods that occur in this thesis are lattice-based, we will introduce the necessary basic definitions. Furthermore, an integral part of Coppersmith’s method is a famous lattice reduction algorithm due to Lenstra, Lenstra and Lovász [LLL82]. Therefore, we will state certain results regarding the LLL algorithm which will be required in later chapters.

Chapter 3
In this chapter we explain the underlying ideas of Coppersmith’s algorithm for finding small solutions of modular equations $f(x_1, \ldots, x_n) = 0 \mod N$. It is well known that one can efficiently find a small solution $x_1^{(0)}, \ldots, x_n^{(0)}$ of a linear modular equation $g(x_1, \ldots, x_n) = 0 \mod N$. Let $X_1, \ldots, X_n$ be upper bounds on the solution, i.e. $|x_i^{(0)}| \leq X_i$, then we can heuristically find the solution if the product of the bounds is smaller than the modulus. Basically, Coppersmith’s method extends this
condition to apply to a system of polynomial equations derived from the initial

\[ f(x_1, \ldots, x_n) = 0 \mod N, \] also known as shift polynomials. Then, as a rule

of thumb, we have the condition that the product of the bounds for all occurring

monomials has to be smaller than the product of all moduli. Thus, in order to find a

modular root of \( f(x_1, \ldots, x_n) \mod N \), the crucial point is to derive an appropriate

system of polynomial equations. We will show in Section 3.2 a general strategy

from Jochemsz and May [JM06] on how to construct such a system of polynomial

equations.

In Section 3.3 we put special interest in the analysis of linear equations modulo

divisors. A particular application of the obtained result arises in the factoring with

known bits problem. Recall that Coppersmith [Cop96a] showed that \( \frac{1}{2} \log p \) queries

to a bit oracle are sufficient for the factorization of an RSA modulus \( N = pq \). It is

important to know that for the attack of Coppersmith to work, the bits have to be

in one consecutive block. A practical instantiation of an oracle that reveals bits is

for example given by certain side channel attacks. However, it is not clear, why such

a side channel attack should reveal consecutive bits and not some bits in arbitrary

locations. By analyzing a multivariate linear equation modulo a divisor \( p \) of an

RSA modulus \( N \), we show that approximately \( 0.7 \log p \) bits are sufficient to find the

factorization of \( N \), no matter how they are distributed. In the proof we apply a new

strategy for the selection of the shift polynomials when searching for small solutions

modulo a divisor. In a nutshell, instead of fixing some modulus \( p^m \) and optimizing

the selection of the shift polynomials sharing the desired root modulo \( p^m \), we fix a

set of shift polynomials and optimize the modulus. This turns out to be a significant

advantage when considering multivariate linear equations with an arbitrary number

of variables.

Concluding this chapter, we will present a slightly different algorithm of Coppersmith [Cop96a, Cop97] that deals with the problem of finding small solutions of multivariate equations over the integers, i.e. \( f(x_1, \ldots, x_n) = 0 \). Similar to the modular case, it is a crucial step to define an appropriate collection of shift polynomials that share the common root \( x_1^{(0)}, \ldots, x_n^{(0)} \). Therefore, we recall in Section 3.4.1 a general strategy [JM06] to derive such a collection.

Chapter 4

Currently, there exist two approaches to improve small root bounds. One is given

by Coppersmith’s algorithm as it is presented in Chapter 3. In particular, this

method exploits the algebraic structure of the underlying polynomial. For example, if the polynomial contains monomials \( x \) and \( x^2 \), then Coppersmith’s algorithm

makes use of the fact that these monomials are not independent, but instead \( x^2 \) is

already determined by \( x \). A second possibility to improve small root bounds works

via linearization. This approach tries to combine multiple monomials into a single

variable. Intuitively, a benefit is obtained since the smaller monomials are hidden

in the large monomial, because the size of the new variable is approximately given by

the size of its largest monomial. The linearization technique, however, only works

if the combined monomials have a similar coefficient. Because otherwise a possible

large coefficient becomes part of the new variable and, thus, increases its size

dramatically.

Summarizing, Coppersmith’s technique exploits of the algebraic structure of the
polynomial, while the linearization makes use of the similarity of the polynomials’ coefficients. In this chapter we present a technique, called unravelled linearization, which combines the best of both worlds. Namely, we allow for a linearization but still exploit the polynomial structure. In a nutshell, the first step of unravelled linearization is to perform a linearization of the initial polynomial \( f \) to obtain a polynomial \( g \). However, in contrast to a standard linearization, we keep the relations that are induced by this linearization in mind. Moreover, when constructing a lattice basis for finding small roots of the polynomial \( g \) we make use of the relations between the variables in order to optimize the small root bounds. Let us exemplify the process for the polynomial equation \( f(x, y) = x^2 - y + ax + b = 0 \mod N \). The linearization step of \( f(x, y) \) results in a polynomial \( g(u, x) = u + ax + b \) together with the relation \( u = x^2 - y \). For applying the algorithm of Coppersmith we construct a collection of shift polynomials containing multiples and powers of the polynomial \( g(u, x) \), for example consider the collection \( g(u, x), xg(u, x), g^2(u, x) \). The monomials that appear in the collection of shift polynomials are 1, \( u \), \( x \), \( ux \), \( x^2 \), \( u^2 \). Remember that the algorithm of Coppersmith finds a small solution, if the product of the upper bounds of all monomials is smaller than the product of all moduli of the shift polynomials. Now, the key point in unravelled linearization is that it makes use of the relation among the variables that has been introduced by the linearization, in this case \( x^2 = u - y \). If we unravel the shift polynomials, i.e. we replace each occurrence of \( x^2 \) by \( u - y \), the monomials appearing in the collection of unravelled shift polynomials are 1, \( u \), \( x \), \( ux \), \( x^2 \), \( u^2 \). Thus, we basically replaced the monomial \( x^2 \) by \( y \) and did not change the moduli at all. Supposing that the variable \( y \) is smaller than \( x^2 \), this will lead to improved small root bounds.

This main contribution of this chapter gives a general strategy for the method of unravelled linearization.

Chapter 5

Random numbers are an important ingredient in modern cryptography. Unfortunately, in practice true randomness in a computing device is a very scarce resource. Therefore, it is necessary to create random looking bits using algorithms called pseudo random number generators. In this chapter we will investigate the security of the Blum-Blum-Shub pseudo random number generator \[BBS86\] and its generalization to arbitrary exponents \( e \). Recall that the Blum-Blum-Shub generator comes with an internal state and in each iteration the state is updated by the iterating function \( s_i = s_{i-1}^2 \mod N \) and a certain fraction of the state is output. These output bits form the sequence of pseudo random bits. Our attacks will recover the internal state given the output of the generator, and thereby make the generator predictable. In the first part of this chapter we present an attack on the BBS generator with just two consecutive output values \( k_1, k_2 \). We can split the internal states \( s_1 \) and \( s_2 \) in a known and an unknown part and write

\[
\begin{align*}
  s_1 &= k_1 + x_1 \\
  s_2 &= k_2 + x_2
\end{align*}
\]

for unknown values \( x_1, x_2 \). Notice that if the generator outputs a \((1 - \delta)\)-fraction of the most significant bits of the state, then the unknowns \( x_1, x_2 \) are upper bounded by \( N^\delta \). Using the iterating function we are able to link the two internal states and
derive the equation

\[ s_2 = s_1^2 \mod N \]
\[ \iff (k_2 + x_2) = (k_1 + x_1)^2 \mod N \]
\[ \iff x_1^2 - x_2 + 2k_1x_1 + k_1^2 - k_2 = 0 \mod N. \quad (1.1) \]

Our attack aims to find the small solution \((x_1, x_2)\) of this modular equation using the technique of unravelled linearization from Chapter 4. We will see that this attack works if the generator outputs at least \(\frac{2}{\delta} \log N\) bits in each iteration. This result improves upon known results from Blackburn, Gomez, Gutierrez and Shparlinski [BGPGS05] and Gomez, Gutierrez and Ibeas [IGGP06].

In Section 5.3 we further improve the bound to \(\delta = \frac{1}{2}\) by considering the output of more than two iterations of the generator. To the best of our knowledge this is the first attack appearing in literature that considers multiple iterations. Finally, we conclude this chapter with the observation that the attacks on the Blum-Blum-Shub generator carry over to the well known RSA-based generator which uses the iterating function \(s_i = s_{i-1} \mod N\) for an RSA modulus \(N\) and exponent \(e\). As a result we obtain an attack up to \(\delta = \frac{1}{e}\).

Chapter 6

In this chapter we investigate attacks on small secret exponent RSA. We recall the attack of Boneh and Durfee [BD00] which exploits the structure of the RSA key equation

\[ ed = 1 \mod \varphi(N) \quad (1.2) \]

using Coppersmith’s algorithm for finding small solutions of modular equations. At the center of Section 6.2.1 stands the improved attack of [BD00]. They observed that a superior bound on the size of the solutions is obtained if only a certain subset of shift polynomials is considered. In a natural application of Coppersmith’s algorithm the shift polynomials are chosen such that every subsequent shift polynomial introduces exactly one new monomial. This guarantees that necessary internal computations can be performed easily. However, taking only a subset as proposed by Boneh and Durfee destroys this property and the internal computations become very complicated. This is one of the reasons why an improvement similar to Boneh and Durfee’s has, to the best of our knowledge, not been applied to other problems so far. Using the technique of unravelled linearization introduced in Chapter 4 for the initial polynomial in the Boneh-Durfee attack solves this problem in an easy way. We show that using a certain linearization and unravelling the linearization where possible, allows to use the subset of shift polynomials suggested by Boneh and Durfee, and still retain the desired property of the shift polynomials for an easy computation. With this observation, we are able to prove the best known bound for small secret exponent RSA in a very natural and easy way.

As mentioned earlier, a second possibility to speed up RSA decryptions is choosing small CRT exponents \(d_p, d_q\). In Section 6.3 we will recall the basic ideas that led to the currently best known attack bound of \(d_p, d_q \leq N^{0.073}\) for CRT exponents. In [JM07] the authors observe that their experimental results are much better than their theoretical expectations. Such a behavior usually occurs if the considered set
of shift polynomials is not optimal. Jochemsz and May conjecture that an improved analysis by taking only a subset of the shift polynomials, as in [BD00], allows to increase the exponent 0.073. We are able to apply the technique of unravelled linearization in the scenario of small CRT exponents to identify the subset of shift polynomials that has been conjectured by Jochemsz and May. The theoretical analysis of our new attack perfectly matches the experimental results. Very disappointingly, however, as final result we also obtain the bound $d_p, d_q \leq N^{0.073}$ and cannot improve the exponent. Yet, our attack is more efficient in practice, since we are able to consider a smaller set of polynomials. A number of conducted experiments verifies this claim.

Summarizing, this thesis analyzes algorithmic aspects of Coppersmith’s algorithm for finding small solutions to modular or integer equations. We put forward a new interpretation of a certain optimization process used in analyzing the problem of solving equations modulo divisors, which has found a few practical applications so far [HM08, CJK09]. Further, we introduce a technique called unravelled linearization which results in the best known attack on a class of pseudo random number generators. Also, unravelled linearization can be successfully applied in the context of small secret exponent RSA. It allows for an easy and natural proof of the most important result in this context from Boneh and Durfee, and further identifies a hidden structure in the attack of Jochemsz and May, thereby answering an open question posed in [JM07]. The major results of the thesis have been published at conferences on cryptography [HM08, HM09, HM10].
Chapter 2

Preliminaries

Most of the results in this thesis emerge from lattice based algorithms. Therefore, we shortly introduce a few necessary definitions and facts about lattices. For a thorough introduction to the theory of lattices we refer to standard textbooks on the topic [Cas97, Gru87].

Let $b_1, \ldots, b_r \in \mathbb{R}^n$ be linearly independent vectors. Then the set

$$L := \{ v \in \mathbb{R}^n \mid v = \sum_{i=1}^{r} a_i b_i, \quad a_i \in \mathbb{Z} \}$$

is called a lattice. Thus, a lattice consists of all integral linear combinations of the basis vectors $b_1, \ldots, b_r$. The number $r$ of basis vectors is called rank of the lattice and $n$ is the dimension. If $r = n$ we speak of a full-rank lattice. Usually, a lattice is represented by a basis matrix, where we write the basis vectors $b_i$ as row vectors.

$$B(L) = \begin{pmatrix} b_1 \\ b_2 \\ \vdots \\ b_r \end{pmatrix} \in \mathbb{R}^{r \times n}$$

Every lattice has an infinite number of different basis matrices, which can be transformed into one another by unimodular transformations. A unimodular transformation is represented by a matrix $U \in \mathbb{Z}^{r \times r}$ with $\det(U) = \pm 1$. A two-dimensional lattice with basis vectors $b_1 = (1, 2)$ and $b_2 = (1, -1)$ is given in Figure 2.1. Figure 2.2 shows the same lattice with a different basis $b'_1 = (2, 1), b'_2 = (3, 3)$.

An invariant of a lattice is the volume of the parallelepiped spanned by the basis vectors. It is also called determinant of the lattice and can be computed as

$$\det(L) = \prod_{i=1}^{r} ||b_i^*||,$$

where $b_i^*$ denote the Gram-Schmidt orthogonalized basis vectors and $||x|| = \sqrt{\sum_i x_i^2}$ is the usual Euclidean norm. If the lattice is full-rank, the determinant of the lattice is equal to the absolute value of the determinant of a basis matrix.

$$\det(L) = |\det(B(L))|$$
The determinant of a lattice \( \mathcal{L} \) is closely related to the length of a shortest vector in the lattice. A theorem by Minkowski \([\text{Min10}]\) shows this relation.

**Theorem 1** (Minkowski). *Every rank \( r \) lattice \( \mathcal{L} \) contains a non-zero vector \( v \) with

\[
\|v\| < \sqrt{r \det(\mathcal{L})^{\frac{1}{r}}}
\]

The lengths of shortest vectors in the lattice themselves are called successive minima of the lattice and are defined as

**Definition 1** (Successive minima). *Let \( \mathcal{L} \) be a rank \( r \) lattice in \( \mathbb{R}^n \). For \( i \leq r \) we denote by \( \lambda_i(\mathcal{L}) \) the smallest radius of a ball \( B \) centered at \( O \), such that \( B \) contains \( i \) linear independent vectors. The value \( \lambda_i(\mathcal{L}) \) is called the \( i \)-th successive minimum of \( \mathcal{L} \).

Unfortunately, the proof of Minkowski’s theorem is just an existential one and does not tell us how to find a shortest vector. In dimension two we are able to find a shortest lattice vector efficiently using the Gauss algorithm, and even in any fixed dimension the shortest vector problem is solvable in polynomial time \([\text{Kan87}]\). For arbitrary lattice dimension, on the other hand, computing a shortest vector or even a good approximation is known to be NP-hard \([\text{Ajt98, Mic00, Din02}]\).

Nevertheless, a famous algorithm of Lentra, Lenstra and Lovasz \([\text{LLL82}]\) allows to compute a shortest vector to within an exponential approximation factor efficiently. Due to the large approximation factor this seems to be useless, but as we will see, this approximation is sufficient for the algorithm of Coppersmith. Even more, having an exact SVP algorithm would not improve the results of Coppersmith’s algorithm significantly.

For a given lattice, the LLL algorithm efficiently computes a basis which is nearly orthogonal. We will concentrate on full-rank integer lattices, since the results in this thesis use such lattices.
Theorem 2 (LLL). Let $\mathcal{L} \in \mathbb{Z}^n$ be an integer lattice spanned by basis vectors $\{b_1, \ldots, b_n\}$. The LLL algorithm outputs a reduced basis $\{v_1, \ldots, v_n\}$ with

$$\|v_i\| \leq 2 \frac{n(n-1)}{4(n-1)} \det(\mathcal{L})^{\frac{1}{n-1}}$$

for $i = 1, \ldots, n$

in time polynomial in $n$ and in the bit size of the entries of the basis matrix $B = (b_1, \ldots, b_n)^T$.

For a proof of Theorem 2 we refer to [May03].

Another property of an LLL-reduced basis is given by the following Lemma.

Lemma 1. Let $B = (b_1, \ldots, b_n)^T$ be an LLL-reduced basis of a lattice $\mathcal{L}$ and let $b_i^*$ denote the Gram-Schmidt orthogonalization of the $i$-th basis vector. Then

$$\|b_i^*\| \geq 2^{-\frac{n-1}{2}} \det(\mathcal{L})^{\frac{1}{n}}.$$

Currently the fastest implementation of the LLL algorithm is due to Nguyen and Stehlé [NS09].

Theorem 3 ($L^2$). Let $\mathcal{L} \in \mathbb{Z}^n$ be an integer lattice spanned by basis vectors $\{b_1, \ldots, b_d\}$ with $\max_i \|b_i\| \leq B_{\max}$. The $L^2$ algorithm computes an LLL-reduced basis in time

$$O(d^4 n (d + \log B_{\max}) \log B_{\max}).$$

Furthermore, we often use the Newton polytope as a visual representation of the monomials of a polynomial.
Definition 2 (Newton polytope). The Newton polytope of a polynomial $f(x_1, \ldots, x_n)$ is defined as the convex hull of the point set

$$\left\{ (i_1, \ldots, i_n) \in \mathbb{N}^n \left| x_1^{i_1} \cdots x_n^{i_n} \text{ is a monomial of } f(x_1, \ldots, x_n) \right. \right\}.$$
Chapter 3
Finding Small Solutions of Modular Equations

The problem of finding solutions of modular equations constantly arises in the research field of cryptography. For example, the RSA cryptosystem would be completely insecure, if it was possible to efficiently compute $e$-th roots modulo a hard to factor integer $N$. The computation of an $e$-th root for some value $a$ can be expressed as finding a solution to the univariate modular equation

$$x^e - a = 0 \mod N.$$  

Notice that it is easily possible to find an $e$-th root if we know that this root $x^{(0)}$ is rather small. To be more precise, if $x^{(0)} \leq N^{\frac{e}{e+1}}$ then $x^e$ is not reduced modulo $N$ and the solution can be computed by taking an $e$-th root over the integers, which can be done efficiently, for example by Newton iteration or Sturm sequences (cf. [SB02]).

In Section 3.1 we will introduce the algorithm of Coppersmith [Cop96b] which extends the simple idea from the previous example to finding small solutions of arbitrary modular polynomial equations. Suppose we want to find a small solution $(x^{(0)}_1,\ldots,x^{(0)}_n)$ of a polynomial equation $f_N(x_1,\ldots,x_n) = 0 \mod N$. Coppersmith’s algorithm begins by constructing a collection $C$ of polynomials that share the desired small root modulo some large integer $M$. Given this collection, one tries to find $n$ linear combinations $g_j = \sum f_i$ for $j = 1,\ldots,n$ and $f_i \in C$ of the polynomials in $C$ such that for the small solution $(x^{(0)}_1,\ldots,x^{(0)}_n)$ the value $g_j(x^{(0)}_1,\ldots,x^{(0)}_n)$ is not reduced modulo $M$. Then we are able to efficiently find the small root over the integers using standard algorithms from elimination theory ([CLO97, CLO98, Stu02]).

A crucial point in Coppersmith’s algorithm is the choice of the collection of polynomials $C$. Jochemsz and May presented a strategy to select these so-called shift polynomials for an arbitrary multivariate modular polynomial equation. We will recall their strategy in Section 3.2. While the strategy of Jochemsz and May is often the method of choice for arbitrary polynomials, it is sometimes beneficial to use a dedicated strategy for specific polynomials. In Section 3.3 we will investigate the special class of linear modular equations where the modulus is not known explicitly,

1We will see that a certain assumption is necessary to guarantee that the root can be extracted from the $g_j$.  

but we have only the promise that it is a divisor of some known integer. For this scenario we present a different strategy for selecting the shift polynomials. Our new strategy allows to derive a result for an arbitrary number of unknowns which would be very complicated using the strategy of Jochemsz and May.

### 3.1 Coppersmith’s Algorithm

In [Cop96b] Coppersmith presented a rigorous method to find small solutions of univariate modular equations. To be more precise, he investigated the following problem.

**Small Modular Solution Problem (Univariate)**

Given a composite integer $N$ of unknown factorization, a univariate polynomial $f_N(x) \in \mathbb{Z}[x]$ of degree $\delta$ and an upper bound $X$. Find in polynomial time in $\log N$ and $\delta$ all roots $x^{(0)} \in \mathbb{Z}$ of $f_N$ modulo $N$ with $|x^{(0)}| \leq X$.

The basic idea of Coppersmith’s algorithm has been nicely explained by Howgrave-Graham in [HG97]. The goal is to construct a polynomial $h(x)$ which has the same roots as $f_N(x)$ but over the integers. To construct this polynomial $h(x)$ we proceed as follows. We fix an integer $m$ and define the collection of polynomials $g_{i,k}(x) := x^i f_N(x)^k N^{m-k}$ for $k = 0, \ldots, m$ and some values $i$.

Notice that by construction all $g_{i,k}$ share the common root $x^{(0)}$ modulo $N^m$. Then the same holds for all linear combinations of polynomials $g_{i,k}$. Thus, if we can find a linear combination $h(x)$ such that $h(x^{(0)}) < N^m$ then we are able to compute the root $x^{(0)}$ over the integers, since $h(x^{(0)})$ is not reduced modulo $N^m$.

But how can we know the size of $h(x^{(0)})$ for an unknown value $x^{(0)}$? To upper bound the size of $h(x^{(0)})$ we make use of the knowledge of an upper bound for the small solution $|x^{(0)}| \leq X$. Thus, the norm of the coefficient vector of the polynomial $h(xX)$ gives an idea for the size of $h(x^{(0)})$. The exact condition is given in the following theorem by Howgrave-Graham [HG97].

**Theorem 4** (Howgrave-Graham). Let $h(x) \in \mathbb{Z}[x]$ be an integer polynomial consisting of at most $\omega$ monomials. Suppose that

1. $h(x^{(0)}) = 0 \mod M$ for an $|x^{(0)}| < X$ and some positive integer $M$, and
2. $\|h(xX)\| < \frac{M}{\sqrt{\omega}}$.

Then $h(x^{(0)}) = 0$ holds over the integers.

Thus, if we are able to find a polynomial $h(x)$ such that $\|h(xX)\| < \frac{N^m}{\sqrt{\omega}}$, then we are able to find the root $x^{(0)}$. It remains to find $h(x)$. For this purpose, a lattice reduction algorithm, for example the LLL-algorithm, is used. We build the basis matrix of a lattice by taking the coefficient vectors of the polynomials $g_{i,k}(xX)$ as basis vectors. Applying the LLL reduction algorithm to that basis, we obtain by Theorem 2 in polynomial time a reduced basis $\{r_1, \ldots, r_\omega\}$ with certain properties.
The important property in this case is that the norm of the first basis vector is upper bounded by an expression involving the determinant of the lattice:

\[ \|r_1\| \leq 2^{\frac{\omega}{4}} \det (L)^{\frac{1}{2^\omega}}. \] (3.2)

Combining the requirement for the size of \( h(xX) \) with this bound for the reduced basis vector, we conclude that if

\[ 2^{\frac{\omega}{4}} \det (L)^{\frac{1}{2^\omega}} < \frac{N^m}{\sqrt{\omega}} \] (3.3)

then the polynomial \( h(x) \) corresponding to the basis vector \( r_1 \) fulfills the requirements of Theorem 4.

Equation (3.3) is usually called enabling condition and it depends on the determinant of the lattice as well as its dimension. By the construction of the lattice using the coefficient vectors \( g_{i,k}(xX) \) as basis vectors, the value of the determinant clearly depends on the upper bound \( X \). The objective in Coppersmith’s method is to select the shift polynomials in such a way that the bound \( X \) is as large as possible and the enabling condition is still fulfilled.

It has already been mentioned by Coppersmith that we can extend this approach to handle multivariate modular equations of the form \( f(x_1, \ldots, x_n) = 0 \mod N \). In order to find small solutions \( (x_1^{(0)}, \ldots, x_n^{(0)}) \) we construct a collection of polynomials that share the desired root modulo \( N^m \) for some integer \( m \). That is, we define

\[ g_{k,i_1,\ldots,i_n}(x_1, \ldots, x_n) = x_1^{i_1} \cdot \ldots \cdot x_n^{i_n} f(x_1, \ldots, x_n)^k N^{m-k} \] (3.4)

for \( k = 0, \ldots, m \) and some values \( i_1, \ldots, i_n \).

However, in contrast to the univariate case, it is not sufficient to find one polynomial \( h \) with the root \( (x_1^{(0)}, \ldots, x_n^{(0)}) \) over the integers. Instead we need to find \( n \) polynomials \( h_1, \ldots, h_n \) with this property. In order to guarantee that the \( h_i \) have the desired root over the integers, we use a generalization of Theorem 4 to multivariate polynomials.

**Theorem 5** (Howgrave-Graham (multivariate)). Let \( h(x_1, \ldots, x_n) \in \mathbb{Z}[x_1, \ldots, x_n] \) be an integer polynomial consisting of at most \( \omega \) monomials. Suppose that

1. \( h(x_1^{(0)}, \ldots, x_n^{(0)}) = 0 \mod M \) with \( |x_1^{(0)}| < X_1, \ldots, |x_n^{(0)}| < X_n \) and a positive integer \( M \), and

2. \( \|h(x_1X_1, \ldots, x_nX_n)\| < \frac{M}{\sqrt{\omega}} \).

Then \( h(x_1^{(0)}, \ldots, x_n^{(0)}) = 0 \) holds over the integers.

Further, we know from Theorem 4 that the basis vectors \( r_1, \ldots, r_\omega \) of an LLL-reduced basis satisfy

\[ \|r_1\| \leq \ldots \leq \|r_\omega\| \leq 2^{\frac{\omega}{4(\omega+1-\eta)}} \det (L)^{\frac{1}{\omega+1-\eta}}. \]
Thus, if
\[ 2^{\frac{n(n-1)}{2}} \det (L)^{\frac{1}{n+1-n}} < \frac{N^m}{\sqrt{\omega}}, \] (3.5)
then we have \( n \) polynomials \( h_1(x_1,\ldots,x_n),\ldots,h_n(x_1,\ldots,x_n) \) derived from the basis vectors \( r_1,\ldots,r_n \) of the LLL-reduced basis that fulfill the requirement of Theorem 5.

However, in the multivariate scenario the problem arises that we can no longer guarantee that we are able to extract the root from \( h_1,\ldots,h_n \) efficiently. Instead, we have to rely on the following assumption.

**Assumption 1.** Let \( h_1,\ldots,h_n \in \mathbb{Z}[x_1,\ldots,x_n] \) be the polynomials that are found by Coppersmith’s algorithm. Then the ideal generated by the polynomial equations \( h_1(x_1,\ldots,x_n) = 0,\ldots,h_n(x_1,\ldots,x_n) = 0 \) has dimension zero.

This assumption makes Coppersmith’s algorithm heuristic in the multivariate case and it is necessary to verify this assumption by performing experiments.

### 3.2 Jochemsz-May Strategy

In the previous section we introduced the algorithm of Coppersmith for finding small solutions of modular polynomial equations. An integral part of the algorithm is the selection of a collection of polynomials that share a common root modulo some integer \( N^m \).

In [JM06] Jochemsz and May propose a general strategy to select the collection of shift polynomials for an arbitrary input equation. Their method works as follows.

Let \( \lambda \) be the leading monomial of \( f_N \) with respect to some order on the monomials. Suppose without loss of generality that \( f_N \) is a monic polynomial. Otherwise we multiply with the inverse of the leading coefficient. If this inverse does not exist we have found a factor of the hard to factor integer \( N \).

Fix an integer \( m \) and define for \( k \in \{0,\ldots,m+1\} \) the sets \( M_k \) of monomials as:

\[
M_k := \{ x_1^{i_1} x_2^{i_2} \ldots x_n^{i_n} | x_1^{i_1} x_2^{i_2} \ldots x_n^{i_n} \text{ is a monomial of } f_N^m \\
\text{and } x_1^1 x_2^2 \ldots x_n^n \lambda^k \text{ is a monomial of } f_N^{m-k} \}.
\] (3.6)

If \( \mathcal{M} \) is the set of all monomials in \( f_N^m \), then the set \( M_k \) basically contains all monomials of \( \mathcal{M} \) which can be introduced by taking the power \( f_N^k \) and multiply it by an appropriate monomial.

In certain situations it is beneficial to include so-called extrashifts. That is, we extend the set \( \mathcal{M} \) of all monomials beyond the monomials of \( f_N^m \). The sets \( M_k \) are then defined as:

\[
M_k := \bigcup_{0 \leq i_1 \leq t_1} \ldots \bigcup_{0 \leq j_n \leq t_n} \{ x_1^{i_1+j_1} x_2^{i_2+j_2} \ldots x_n^{i_n+j_n} | x_1^{i_1} x_2^{i_2} \ldots x_n^{i_n} \text{ is a monomial of } f_N^m \\
\text{and } x_1^1 x_2^2 \ldots x_n^n \lambda^k \text{ is a monomial of } f_N^{m-k} \},
\] (3.6)

for certain values \( t_1,\ldots,t_n \geq 0 \).
Once we defined the sets \( M_k \), we can construct the set of shift polynomials by enumerating all monomials in \( M \) and taking for each one the largest possible power of \( f_N \). Formally, the collection of shift polynomials is given by

\[
g_{i_1, \ldots, i_n, k}(x_1, \ldots, x_n) := \frac{x_1^{i_1} x_2^{i_2} \cdots x_n^{i_n}}{\lambda^k} f_N(x_1, \ldots, x_n)^k N^{m-k},
\]

for \( k = 0, \ldots, m \) and \( x_1^{i_1} x_2^{i_2} \cdots x_n^{i_n} \in M_k \setminus M_{k+1} \).

From this point on, we proceed with Coppersmith’s algorithm as described in Section 3.1 I.e. for upper bounds \( X_1, \ldots, X_n \) on the solution, we take the coefficient vectors \( g_{i_1, \ldots, i_n, k}(x_1 X_1, \ldots, x_n X_n) \) of these shift polynomials as basis vectors to form a lattice \( \mathcal{L} \). As shown in [Joc07] the basis matrix of this lattice will be lower triangular, if we choose an appropriate ordering of the columns. The terms appearing on the diagonal of the basis matrix are then exactly \( X_1^{i_1} X_2^{i_2} \cdots X_n^{i_n} N^{m-k} \) for given combinations of \( i_j \) and \( k \). Thus, we can compute the determinant of the lattice \( \mathcal{L} \) as

\[
\det(\mathcal{L}) = \prod_{k=0}^{m} x_1^{i_1} x_2^{i_2} \cdots x_n^{i_n} \in M_k \setminus M_{k+1} \prod_{k=0}^{m} X_1^{i_1} X_2^{i_2} \cdots X_n^{i_n} N^{m-k}.
\]

This can be simplified to

\[
\det(\mathcal{L}) = \left( \prod_{x_1^{i_1} x_2^{i_2} \cdots x_n^{i_n} \in M_0} X_1^{i_1} X_2^{i_2} \cdots X_n^{i_n} \right) \prod_{k=0}^{m} N(|M_k| - |M_{k+1}|)(m-k)
\]

or written differently as

\[
\det(\mathcal{L}) = \prod_{j=1}^{n} X_j^{s_j} N^{s_N}, \quad \text{for } \begin{cases} s_j = \sum_{x_1^{i_1} x_2^{i_2} \cdots x_n^{i_n} \in M_0} i_j \\ s_N = \sum_{k=0}^{m} (|M_k| - |M_{k+1}|)(m-k) \end{cases}.
\]

The expression for \( s_N \) can be further simplified to

\[
s_N = \sum_{k=0}^{m} (|M_k| - |M_{k+1}|)(m-k)
\]

\[
= m |M_0| - m |M_1| + (m-1) |M_1| - (m-1) |M_2| + \ldots + |M_{m-1}| - |M_m|
\]

\[
= m |M_0| - |M_1| - |M_2| - \ldots - |M_m|
\]

\[
= m |M_0| - \sum_{k=1}^{m} |M_k|.
\]

The next step would be to require the short vectors in the lattice to be sufficiently small to reveal \( n \) polynomials which have the same root over the integers. However, at this point we would like to make the strategy of Jochemsz-May slightly more general by considering polynomials modulo an unknown divisor of a known integer \( N \). This scenario often appears in the context of the RSA cryptosystem, e.g. if certain bits of the prime factor \( p \) of the RSA modulus \( N = pq \) are known. This so-called factoring with known bits problem has been initially investigated by Coppersmith in [Cop96a] using a bivariate polynomial over the integers, and has
been rephrased as the problem of finding a small root of an univariate polynomial modulo an unknown divisor by Howgrave-Graham \cite{HG97}. Similarly, we can formulate the problem of finding small solutions for multivariate polynomial equations modulo unknown divisors.

**Small Solution Modulo Divisor Problem**

Let $N$ be an integer of unknown factorization with a divisor $b \geq N^\beta$ for $\beta \in (0,1]$. Furthermore, let $f_b$ be a multivariate polynomial in $n$ variables of degree $\delta$. Find all solutions $(x_1^{(0)}, \ldots, x_n^{(0)})$ of $f_b(x_1, \ldots, x_n) = 0 \mod b$ with $|x_1^{(0)}| \leq X_1, \ldots, |x_n^{(0)}| \leq X_n$ for some $X_1, \ldots, X_n$ in polynomial time in $\log N$ and $\delta$.

Note that the special case of a known modulus is obtained by setting the size $\beta$ of the divisor equal to one.

When considering such a polynomial equation modulo an unknown divisor, we can use in the lattice construction the identical shift polynomials as defined in (3.7), except with the polynomial $f_b$ instead of $f_N$.

Thereby, we use the fact that $b$ is a divisor of $N$ and, thus, a root modulo $N$ is also a root modulo $b$. This guarantees that all polynomials in the collection share the desired root modulo $b^m$. In particular notice that the determinant of the lattice constructed from the shift polynomials remains the same as in (3.8).

Recall the enabling condition obtained from the theorem of Howgrave-Graham and the properties of an LLL-reduced basis for a modulus $N$.

$$2^{\frac{\omega(\omega-1)}{4(\omega+1-n)}} \det (L) \leq \frac{N^m}{\sqrt{\omega}}$$

It is convenient to introduce an error term $\epsilon$ that captures terms that do not depend on $N$ as well as lower order terms in $m$. This way, we obtain for a fixed number of variables the asymptotic bound

$$\det (L) < N^{m \omega - \epsilon}.$$  \hspace{1cm} (3.10)

In the following we call this simplified enabling condition. In the present scenario we deal with an unknown modulus $b$ for which we know that $b \geq N^\beta$ for $0 < \beta \leq 1$. Thus, the simplified enabling condition translates to

$$\det (L) < \left(N^\beta \right)^{m \omega - \epsilon}.$$  \hspace{1cm} (3.11)

The dimension $\omega$ of the lattice equals the number of shift polynomials, and this in turn is given by $|M_0|$. Then, with the value of the determinant given in (3.8) we finally obtain the condition on the size of the upper bounds

$$\prod_{j=1}^{n} X_j^{s_j} < N^{s_N'}, \text{ for } \begin{cases} s_j = \sum_{i_1=1}^{i_j} x_1^{i_1} x_2^{i_2} \ldots x_n^{i_n} \in M_0 \ , & s_N' = (\beta - 1)m |M_0| + \sum_{k=1}^{m} |M_k| \ . \end{cases}$$  \hspace{1cm} (3.11)
If this condition is fulfilled then we obtain sufficiently many short vectors in the LLL-reduced lattice which, under Assumption 1, allow to efficiently recover all small roots \((x_1^{(0)}, \ldots, x_n^{(0)})\) with \(|x_1^{(0)}| \leq X_1, \ldots, |x_n^{(0)}| \leq X_n\).

**Remark.** For \(\beta = 1\) condition (3.11) exactly matches the result of Jochemsz and May.

### 3.3 Linear Equations

In this section we will investigate a very special class of modular polynomial equations, namely linear ones. These equations play a central role in modern cryptanalysis since many problems already admit a linear structure. However, even problems with an inherently non-linear structure can trivially be transformed into the problem of solving a linear equation by a process called linearization. Linearization usually means that one introduces a dedicated variable for each monomial occurring in the original problem description. Thus, any progress in solving linear equations automatically reveals a new algorithm for solving arbitrary polynomial equations.

Basically, the linear problems that we analyze in this section can be solved by Coppersmith’s algorithm with the selection of shift polynomials as it has been presented in Section 3.2. However, for linear equations with a known modulus there exists a specialized algorithm which requires only small dimensional lattices instead of high dimensional ones as in Coppersmith’s method. We will review this method in Subsection 3.3.1.

Besides finding small solutions of linear modular equations with a known modulus, it is in some scenarios necessary to solve a linear equation with an unknown modulus that is a divisor of some known integer. In introduction we already mentioned the connection of solving equations modulo divisors with the *factoring with known bits problem*. For linear equations the problem of finding small solutions modulo divisors can be stated as follows.

**Small Modular Solution Problem (Linear)**

Let \(N\) be an integer of unknown factorization with a divisor \(b \geq N^\beta\) for \(\beta \in (0, 1]\). Given a linear polynomial \(f \in \mathbb{Z}[x_1, \ldots, x_n]\) in \(n\) variables and integers \(X_1, \ldots, X_n\); find all solutions \((x_1^{(0)}, \ldots, x_n^{(0)})\) of \(f(x_1, \ldots, x_n) = 0 \mod b\) with \(|x_i^{(0)}| \leq X_i\).

We show a new selection process for the collection of shift polynomials that is different from the strategy of Jochemsz and May, and that allows a simpler analysis. This, in turn, makes it possible to analyze the problem for an arbitrary number of unknowns, which would be rather complicated using the technique of Section 3.2.

We explain the idea of our new selection of shift polynomials for bivariate polynomial equations in Subsection 3.3.2 and then present the main theorem for multivariate polynomial equations with an arbitrary number of variables in Subsection 3.3.3. Finally, we show two applications of our new algorithm. One will extend the problem of *factoring with known bits* to the scenario where the unknown bits are no longer in one consecutive block, and one will present an attack from Coron, Joux, Kizhvatov, Naccache and Paillier [CJK+09] on a signature scheme based on the RSA assumption.
3.3.1 Linear Equations with Known Modulus

In this subsection we briefly sketch the folklore method for finding small roots of linear modular equations

\[ a_1 x_1 + a_2 x_2 + \ldots + a_n x_n = 0 \mod N \]  \hspace{1cm} (3.12)

with a known modulus \( N \).

Usually equations of the form (3.12) have many solutions \((x_1^{(0)}, x_2^{(0)}, \ldots, x_n^{(0)}) \in \mathbb{Z}_N^n\), however, a counting argument shows that we can expect a unique solution if the product of the unknowns is smaller than the modulus. More precisely, let \( X_i \) be upper bounds on the variables, i.e. \(|x_i^{(0)}| \leq X_i \) for \( i = 1, \ldots, n \), then we expect a unique (non-trivial) solution if \( \prod X_i \leq N \). In the following we will show that this solution can also be computed efficiently. Notice that Equation (3.12) describes a homogeneous equation. To handle the inhomogeneous case, we simply introduce a dummy variable \( x_{n+1} \) with \( X_{n+1} = 1 \) to transform the inhomogeneous equation to a homogeneous one.

The algorithm that we will describe is, however, quite different from the other algorithms described in this thesis. Instead of computing an approximation of the shortest vector via LLL reduction, in this case we need to determine an actual shortest vector. This problem is NP-hard \(^2\) in general, but we are able to solve it here because we consider lattices of fixed dimension. Another difference is the heuristic that applies. In Coppersmith-type algorithms we obtain a system of equations over \( \mathbb{Z} \) and the employed heuristic is that this system allows to extract the roots efficiently (Assumption [1]). The current algorithm, however, assumes that there is only one linear independent vector which fulfills the Minkowski bound for a shortest vector, which then directly gives the desired solution.

Let us explain how the algorithm works. First of all, assume in (3.12) that \( \gcd(a_i, N) = 1 \) for at least one \( i \) and let w.l.o.g. \( \gcd(a_n, N) = 1 \). Further let \( X_i \) be upper bounds on the sizes of the \( x_i^{(0)} \). Multiplying Equation (3.12) by \( -a_n^{-1} \) gives

\[ b_1 x_1 + b_2 x_2 + \ldots + b_{n-1} x_{n-1} = x_n \mod N, \quad \text{where} \quad b_i = a_n^{-1} a_i. \]  \hspace{1cm} (3.13)

For a solution \((x_1^{(0)}, x_2^{(0)}, \ldots, x_n^{(0)})\) of (3.13) we know that \( \sum_{i=1}^{n-1} b_i x_i^{(0)} = x_n^{(0)} - y N \) for some \( y \in \mathbb{Z} \). Consider the lattice \( \mathcal{L} \) generated by the rows of the following basis matrix

\[ B = \begin{pmatrix}
1 & 0 & 0 & \ldots & 0 & b_1 \\
0 & 1 & 0 & \ldots & 0 & b_2 \\
\vdots & \ddots & \ddots & \ddots & \ddots & \ddots \\
0 & 0 & 0 & \ldots & 1 & b_{n-1} \\
0 & 0 & 0 & \ldots & 0 & N
\end{pmatrix} \]

This lattice contains the solution vector which we want to find. We have

\[ (x_1^{(0)}, x_2^{(0)}, \ldots, x_{n-1}, y) \cdot B = (x_1^{(0)}, x_2^{(0)}, \ldots, x_{n-1}^{(0)}, \sum_{i=1}^{n-1} b_i x_i^{(0)} + y N) \]

\[ = (x_1^{(0)}, x_2^{(0)}, \ldots, x_{n-1}^{(0)}, x_n^{(0)}). \]

\(^2\)under randomized reductions
Remember that we aim to find the unique small solution with $|x^{(0)}_i| \leq X_i$ and $\prod_i X_i \leq N$. The theorem of Minkowski (Theorem 1) now states that the first successive minimum of the lattice $\mathcal{L}$, i.e. the length of a shortest vector, is smaller than $\sqrt{n} \det(\mathcal{L})^{\frac{1}{n}} = \sqrt{n} N^{\frac{1}{n}}$. However, in general the vector $v = (x^{(0)}_1, x^{(0)}_2, \ldots, x^{(0)}_{n-1}, x^{(0)}_n)$ will have a norm of approximately $\sqrt{n} \max_i |x^{(0)}_i|$ and will probably not be a shortest vector, unless the components of the vector $v$ are balanced, i.e. $|x^{(0)}_1| \approx |x^{(0)}_2| \approx \ldots \approx |x^{(0)}_n| \approx N^{\frac{1}{n}}$. Then the vector $v$ would fulfill the Minkowski bound and by the heuristic that this vector is the only one which fulfills the bound, we could find it by solving an SVP instance.

What we basically do to handle this obstacle is to modify the lattice $\mathcal{L}$, such that a vector $v'$ which is related to the solution vector $v$, actually fulfills the Minkowski bound. To achieve this, we define $Y_i = \frac{X_i}{N}$ for $i = 1, \ldots, n$ and rebalance the columns of the basis matrix.

$$B' = \begin{pmatrix} Y_1 & 0 & 0 & \ldots & 0 & Y_n b_1 \\ 0 & Y_2 & 0 & \ldots & 0 & Y_n b_2 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & 0 & \ldots & Y_{n-1} & Y_n b_{n-1} \\ 0 & 0 & 0 & \ldots & 0 & Y_n N \end{pmatrix}$$

In the lattice $\mathcal{L}'$ described by basis matrix $B'$ the same linear combination as in (3.14) gives us the vector $v'$ as

$$(x^{(0)}_1, x^{(0)}_2, \ldots, x^{(0)}_{n-1}, y) \cdot B' = (Y_1 x^{(0)}_1, Y_2 x^{(0)}_2, \ldots, Y_{n-1} x^{(0)}_{n-1}, \sum_{i=1}^{n-1} Y_n b_i x^{(0)}_i + Y_n y N)
= (Y_1 x^{(0)}_1, Y_2 x^{(0)}_2, \ldots, Y_{n-1} x^{(0)}_{n-1}, Y_n x^{(0)}_n)$$

Notice that we can easily recover the $x^{(0)}_i$ from $v'$ by just dividing the $i$-th component of $v'$ by $Y_i$. Further note that each of the components of the vector is smaller than $N$, i.e. $Y_i x^{(0)}_i = \frac{X_i}{N} x^{(0)}_i \leq N$ since the $x^{(0)}_i \leq X_i$. The norm of this vector $v'$ is thus

$$\|v'\| \leq \sqrt{n} N.$$

For the new lattice $\mathcal{L}'$ we compute the determinant as

$$\det(\mathcal{L}') = N \prod_{i=1}^{n} Y_n = N^{n+1} \prod_{i=1}^{n} \frac{1}{X_i}$$

and remember that the Minkowski Theorem states $\lambda_1(\mathcal{L}') \leq \sqrt{n} \det(\mathcal{L}')^{\frac{1}{n}}$. Putting things together, we have that the vector $v'$ fulfills the Minkowski bound if

$$\sqrt{n} N \leq \sqrt{n} \det(\mathcal{L}')^{\frac{1}{n}}$$

$$\Leftrightarrow N^n \leq \det(\mathcal{L}')$$

$$\Leftrightarrow \prod_{i=1}^{n} X_i \leq N.$$
Thus, by solving an SVP in $\mathcal{L}'$ we will find the vector $v'$ if $\prod_{i=1}^{n} X_i \leq N$, under the heuristic assumption that no other vector fulfills the Minkowski bound, i.e. the second successive minimum of the lattice is larger than $\sqrt{n}N$.

We summarize the result in the following theorem.

**Theorem 6.** Let $N \in \mathbb{N}$ and $f_N(x_1, \ldots, x_n) = a_1x_1 + a_2x_2 + \ldots + a_nx_n$ be a linear polynomial in $n$ variables with $\gcd(a_i, N) = 1$ for at least one $i$. Further let $X_i \in \mathbb{N}$ be upper bounds on the solutions $x_i^{(0)}$ of the equation $f_N(x_1, \ldots, x_n) = 0 \mod N$.

Heuristically, we can find the solution $(x_1^{(0)}, \ldots, x_n^{(0)})$ if $\prod_{i=1}^{n} X_i \leq N$ in time polynomial in $\log N$.

### 3.3.2 Bivariate Linear Equations modulo Divisors

If the modulus is not explicitly known we can no longer use the previous construction, since we are not able to build the lattice basis. Therefore, we need to follow the approach of Coppersmith’s algorithm and construct a collection of polynomials that share a common root modulo $p^m$, where $p$ is the unknown divisor. As mentioned in the introduction of this section, we present a new strategy to select the collection, i.e. the shift polynomials.

We introduce our algorithm by considering the most simple case of a bivariate linear equation $f(x_1, x_2) = a_1x_1 + a_2x_2 + a_3 = 0 \mod p$, where $a_i \in \mathbb{Z}_p$ and $p$ is an unknown divisor of $N$. Without loss of generality we can assume that $f$ is a monic polynomial. If $a_1 \neq 1$, we multiply $f(x_1, x_2)$ with $a_1^{-1} \mod N$ and if this inverse does not exist, we have found a factor of the (hard to factor) integer $N$.

We will prove the following theorem.

**Theorem 7.** Let $\epsilon > 0$ and let $N$ be a sufficiently large composite integer of unknown factorization with a divisor $p \geq N^\beta$. Furthermore, let $f(x_1, x_2) \in \mathbb{Z}[x_1, x_2]$ be a linear polynomial in two variables. Under Assumption 1, we can find all solutions $(x_1^{(0)}, x_2^{(0)})$ of the equation $f(x_1, x_2) = 0 \mod p$ with $|x_1^{(0)}| \leq N^\gamma$ and $|x_2^{(0)}| \leq N^\delta$ if

$$\gamma + \delta \leq 3\beta - 2 + 2\left(1 - \beta\right)^{3/2} - \epsilon.$$

The algorithm’s time complexity is polynomial in $\log N$ and $\epsilon^{-1}$.

**Proof:** Set the integer $m = \left\lceil \frac{3\beta(1+\sqrt{1-\beta})}{\epsilon} \right\rceil$ and define a collection of shift polynomials that shares the common root $(x_1^{(0)}, x_2^{(0)})$ modulo $p^t$. We take the collection

$$g_{k,i}(x_1, x_2) := x_i^2 f^k(x_1, x_2) N^\max\{t-k,0\}$$

for $k = 0, \ldots, m; i = 0, \ldots, m - k$ and $t = \tau m$ for a value $0 < \tau \leq 1$ that will be optimized later.
We can define the following order on the shift polynomials in this collection: If \( k < \ell \) then \( g_{k,i} < g_{\ell,j} \), and if \( k = \ell \) then \( g_{k,i} < g_{\ell,j} \iff i < j \). Enumerating the shift polynomials in this order guarantees that each subsequent polynomial introduces exactly one new monomial and therefore the corresponding coefficient vectors define a lower triangular basis matrix. Figure 3.1 show a schematic of such a basis matrix for a lattice \( L \). Recall that

\[
B = \begin{pmatrix}
N^{\tau m} & X_2 N^{\tau m} & \cdots & X_2^m N^{\tau m} & X_1 N^{\tau m-1} \\
& \ddots & \cdots & \ddots & \cdots \\
& & X_1 X_2^{m-1} N^{\tau m-1} & X_1^2 N^{\tau m-2} & \cdots \\
& & & \ddots & \cdots \\
& & & & X_1^{m-1} \\
& & & & & X_1^{m-1} X_2 \\
& & & & & & X_1^m 
\end{pmatrix}
\]

Figure 3.1: Triangular basis matrix for approach that optimizes modulus

our goal is to find two sufficiently small vectors in the lattice \( L \) which in turn correspond to two polynomials that contain the desired small roots over the integers. By Theorem 2 we know that an LLL-reduced basis contains two basis vectors of norm smaller than \( 2^{\frac{\dim(L)}{4}} \det(L)^{\frac{1}{\dim(L)}} \). On the other hand, in order to reveal polynomials with the desired root over the integers, we require by Howgrave-Graham’s theorem (Theorem 5) that \( \|v_i\| \leq \dim(L)^{-\frac{1}{2}} (N^\beta)^{\tau m} \) for \( i = 1, 2 \). Thus, if

\[
2^{\frac{\dim(L)}{4}} \det(L)^{\frac{1}{\dim(L)}} \leq \dim(L)^{-\frac{1}{2}} (N^\beta)^{\tau m} \tag{3.15}
\]

then we will find two sufficiently short vectors. It remains to compute the necessary condition on the upper bounds \( X_1, X_2 \) for this condition to hold. Therefore, we need to know the determinant and dimension of the lattice \( L \). The determinant \( \det(L) = X_1^{s_{x_1}} X_2^{s_{x_2}} N^{s_N} \) can be easily computed. For \( s_{x_1} \) and \( s_{x_2} \) we simply count the monomials that occur in \( f^m \). Each of these monomials is placed on the diagonal of the basis matrix by one of the shift
polynomials. We get
\[
s_{x_1} = \sum_{i=0}^{m} \sum_{j=0}^{m-i} i = \frac{1}{6} \left( m^3 + 3m^2 + 2m \right),
\]
\[
s_{x_2} = \sum_{i=0}^{m} \sum_{j=0}^{m-i} j = \frac{1}{6} \left( m^3 + 3m^2 + 2m \right).
\]

Using the same argument, namely that each monomial is placed on the diagonal by exactly one shift polynomial, we can compute the dimension of the lattice, which is equal to the total number of shift polynomials.

\[
D := \dim \mathcal{L} = \sum_{i=0}^{m} \sum_{j=0}^{m-i} 1 = \frac{1}{2} \left( m^2 + 3m + 2 \right)
\]  
\[
(3.16)
\]

To derive the value \( s_N \) we look at the definition of the shift polynomials \( g_{k,i} \). For each \( k \) from 0 to \( t \), a factor of \( N^{t-k} \) is used, and the number of polynomials with that factor is exactly \( m + 1 - k \). Thus we have

\[
s_N = \sum_{k=0}^{t} (m + 1 - k) (t - k).
\]

Here we notice the major difference between the new approach to optimize the modulus and the previous one which optimized the bound via extrashifts. While in the latter case a parameter \( \tau \) that optimizes the values \( t_1, \ldots, t_n \) in 3.6 influences all parts of the determinant as well as the dimension, in the new analysis only the exponent \( s_N \) depends on the value \( \tau \). This turns out to be a significant advantage in the case of an arbitrary number of variables as we will see in Subsection 3.3.3.

Given the values \( s_{x_1}, s_{x_2}, s_N \) and the dimension \( D \), we are able to derive upper bounds \( X_1, X_2 \) on the solutions which can be efficiently found. Recall the enabling condition 3.15

\[
2 \frac{D(D-1)}{4} \det (\mathcal{L}) \leq D \frac{D-1}{2} \left( N^\beta \right)^{\tau m(D-1)}.
\]

If we plug in the values for the determinant and make use of the observation that \( s_{x_1} = s_{x_2} = \frac{Dm}{3} \), we can solve for \( X_1 X_2 \) and obtain

\[
X_1 X_2 \leq 2^{-\frac{3(D-1)}{4m}} D^{\frac{3(D-1)}{2m}} N^{\frac{3 \tau (D-1)m}{2m}} \frac{3s_N}{Dm}.
\]

In this expression, the power of 2 as well as the power of \( D \) is negligible for sufficiently large values of \( N \). Further, an analysis of the right hand side yields an asymptotically optimal value with \( \tau = 1 - \sqrt{1 - \beta} \).

\[^3\text{asymptotically with respect to the parameter } m\]
3.3 Linear Equations

It remains to upper bound the exponent of $N$. Using the values $D = \frac{1}{2}(m^2 + 3m + 2)$ and $s_N = \sum_{i=0}^{\tau m}(m + 1 - i)(\tau m - i)$ we get an upper bound of

$$\tau (3\beta - 3\tau + \tau^2) + \frac{-\tau - 6\beta \tau + \tau^3}{1 + m} - \frac{2(\tau - 3\beta \tau - 3\tau^2 + 2\tau^3)}{2 + m}.$$

Further, using the optimized value $\tau = 1 - \sqrt{(1 - \beta)}$ we have

$$-2 + 2\sqrt{1 - \beta} + 3\beta - 2\beta\sqrt{1 - \beta}$$

$$= \frac{3\sqrt{1 - \beta}}{1 + m} + \frac{6\sqrt{1 - \beta}}{2 + m} + \frac{7\beta\sqrt{1 - \beta}}{1 + m} = \frac{6(-1 + 2\beta)}{2 + m} + \frac{7\beta\sqrt{1 - \beta}}{2 + m} > 0 \quad \text{for all } \beta \in (0, 1).$$

(3.17)

Finally, we bound the positive term by $\frac{6(-1 + \sqrt{1 - \beta} + 2\beta)}{2m}$ and the negative term by $\frac{3\sqrt{1 - \beta} - 3\beta\sqrt{1 - \beta}}{m}$ and obtain

$$N^{\frac{3\beta m(D - 1)}{m}} - \frac{3\beta m}{N} \geq N^{-2 + 2\sqrt{1 - \beta} + 3\beta - 2\beta\sqrt{1 - \beta} - \frac{3\beta(1 + \sqrt{1 - \beta})}{m}}.$$

Thus, we are able to find all small solutions if the bounds fulfill condition

$$X_1X_2 \leq N^{3\beta - 2 + 2(1 - \beta)^{\frac{3}{2}} - \epsilon},$$

with an error term $\epsilon = \frac{3\beta(1 + \sqrt{1 - \beta})}{m}$.

It remains to show the algorithm’s complexity. The running time is dominated by the LLL reduction given in Theorem 3 which is polynomial in the bit size of the entries and the dimension of the corresponding lattice. Recall that in Equation (3.16) we derived that the dimension is $O(m^2)$ and thus by our choice of $m$ this is polynomial in $\epsilon^{-1}$. The entries of the basis matrix are defined by the (weighted) coefficient vectors of the shift polynomials. Assuming the coefficients of $f$ are reduced modulo $N$ (since we search for roots modulo $p$, a divisor of $N$), the entries in $f^k(x_1X_1, x_2X_2)$ are of size $O(2k\log N)$. Furthermore, in $g_{k,i}(x_1X_1, x_2X_2)$ an additional factor of $N^{\tau m - k}$ increases the bit size to $O((m + k)\log N)$ and finally the factor $X_2^j$ also contributes at most $O(m \log N)$. Altogether, since $k \leq m$, we have entries in the basis matrix of size $O(m \log N)$ and with $m = O(\epsilon^{-1})$ a running time which is polynomial in $\log N$ and $\epsilon^{-1}$. ■
Let us discuss the idea behind the new method of selecting the shift polynomials. Recall that the basic approach of Coppersmith’s algorithm selects an integer $m$ and defines a collection of shift polynomials that share a common root modulo $p^m$. When seeking a small root modulo divisors, the selection of the shift polynomials according to the strategy of Jochemsz and May suggests to use so-called extrashifts in order to improve the bound. Moreover, supposing the unknowns are of the same size, the strategy of Jochemsz and May should be used with the same amount of extrashifts in both variables. Then Newton polytope defined by the monomials that are used in the lattice construction forms an isosceles triangle as depicted in Figure 3.2.

![Figure 3.2: Extrashifts for balanced variables](image)

Recall that in this case, we constructed a set of polynomials that shared the common root modulo $p^m$, and to obtain the asymptotic bound, we optimized the amount of extrashifts via a parameter $t$ for both $x_1$ and $x_2$.

Our new strategy, on the other hand, fixes the monomials that are used in the lattice construction and instead of appending extrashifts we optimize the modulus that is used. This gives us a picture as shown in Figure 3.3.

![Figure 3.3: New approach of optimizing the modulus](image)

Figures 3.2 and 3.3 look very similar and the advantage of our new approach lies in the optimization stage. Namely, by fixing the set of monomials, the values
3.3 Linear Equations

$s_{x_1}, s_{x_2}$ and the lattice dimension no longer need to be optimized — in contrast to the standard approach.

Remark. It is worth noting that our new approach as well as the strategy of Jochemsz and May should be slightly modified in case of unbalanced variables. If, for example, the variable $x_1$ is much smaller than $x_2$, then the strategies should adapt this in the selection of the shift polynomials.

3.3.3 Multivariate Linear Equations

In the previous subsection we analyzed a special case of solving linear equations modulo an unknown divisor, namely we considered only two variables. In this subsection we will generalize the suggested approach to handle an arbitrary number of variables and thereby present the main result of this section.

Theorem 8. Let $\epsilon > 0$ and let $N$ be a sufficiently large composite integer of unknown factorization with a divisor $p \geq N^\beta$. Furthermore, let $f(x_1, \ldots, x_n) \in \mathbb{Z}[x_1, \ldots, x_n]$ be a linear polynomial in $n$ variables. Under Assumption 1, we can find all solutions $(x^{(0)}_1, \ldots, x^{(0)}_n)$ of the equation $f(x_1, \ldots, x_n) = 0 \mod p$ with $|x^{(0)}_i| \leq N^{\gamma_i}, \ldots, |x^{(0)}_n| \leq N^{\gamma_n}$ if

$$\sum_{i=1}^{n} \gamma_i \leq 1 - (1 - \beta) \frac{\gamma_i + 1}{n} - (n - 1)(1 - \sqrt{1 - \beta})(1 - \beta) - \epsilon. \quad (3.19)$$

The time complexity of the algorithm is polynomial in $\log N$ and $(\frac{\epsilon}{\epsilon})^n$, where $e$ is Euler’s constant.

Before we prove Theorem 8, we mention a few required lemmata.

Lemma 2. Let $P = \{(t_1, \ldots, t_n) \in \mathbb{Z}^n \mid \sum_{i=1}^{n} t_i \leq m$ and $t_i \geq 0$ for all $i\}$ be an $n$-dimensional simplex in $\mathbb{Z}^n$. Then

1. the number of points in $P$ is $\binom{m+n}{n}$, and
2. $s_i := \sum_{(t_1, \ldots, t_n) \in P} t_i = \binom{m+n}{m-1}$ for all $i \in \{1, \ldots, n\}$.

Proof: The first part of the lemma can be interpreted as counting the number of monomials in $n$ variables of degree at most $m$. This number is given by

$$\sum_{i=0}^{m} \binom{i + (n - 1)}{(n - 1)} = \binom{m + n}{n}.$$

The second part instead computes a weighted sum over these monomials. I.e. for $s_i$ take all possible exponents $k$ and sum over $k$ times the number of monomials in $n - 1$ variables of degree at most $m - k$.

$$s_i = \sum_{k=0}^{m} k \binom{m - k + n - 1}{n - 1} = \binom{m + n}{m - 1}$$

(cf. [GKP89] for useful binomial identities.)
Lemma 3. For an integer \( n > 0 \) and \( \beta \in (0, 1] \) we have
\[
n(1 - \sqrt[\beta]{1 - \beta}) \leq -\ln(1 - \beta)
\]

Proof: Since \( 1 - \frac{1}{x} \leq \ln x \) for \( x > 0 \), we can write
\[
n(1 - \sqrt[\beta]{1 - \beta}) = n \left(1 - \frac{1}{(1 - \beta)^{\frac{1}{n}}} \right) \leq n \ln \left((1 - \beta)^{-\frac{1}{n}} \right) = -\ln(1 - \beta)
\]

Lemma 4. For an integer \( n > 0, \beta \in (0, 1] \) and \( \tau = 1 - \sqrt[\beta]{1 - \beta} \) we have
\[
\frac{\prod_{k=0}^{n}(m(1 - \tau) + k)}{\prod_{k=0}^{n}(m + k)} \leq (1 - \beta)^{\frac{n+1}{n}} + \frac{1}{\pi} (1 - \beta)^{-0.278 \frac{n}{m}}.
\]

Proof: First notice that
\[
\frac{\prod_{k=0}^{n}(m(1 - \tau) + k)}{\prod_{k=0}^{n}(m + k)} = (1 - \tau)^{n+1} + \frac{\prod_{k=0}^{n}(m(1 - \tau) + k) - (1 - \tau)^{n+1} \prod_{k=0}^{n}(m + k)}{\prod_{k=0}^{n}(m + k)}.
\]

We analyze the second part of this sum. Its partial fraction expansion is
\[
\frac{\prod_{k=0}^{n}(m(1 - \tau) + k) - (1 - \tau)^{n+1} \prod_{k=0}^{n}(m + k)}{\prod_{k=0}^{n}(m + k)} = \frac{c_0}{m} + \frac{c_1}{m+1} + \ldots + \frac{c_n}{m+n}.
\]

Our goal is to determine the values of \( c_i \). Start by multiplying with \( \prod_{k=0}^{n}(m+k) \):
\[
\prod_{k=0}^{n}(m(1 - \tau) + k) - (1 - \tau)^{n+1} \prod_{k=0}^{n}(m + k) = \sum_{i=0}^{n} c_i \prod_{k \neq i}^{n}(m + k).
\]

Now we successively set \( m \) equal to the roots of the denominator and solve for \( c_i \). For the \( i \)-th root \( m = -i \) we obtain
\[
\prod_{k=0}^{n}(-i(1 - \tau) + k) = c_i \prod_{k \neq i}^{n}(k - i)
\]
\[
c_i = \frac{\prod_{k=0}^{n}(-i(1 - \tau) + k)}{\prod_{k \neq i}^{n}(k - i)}
\]

We can rewrite this in terms of the Gamma function as
\[
c_i = (-1)^i \frac{\Gamma(-i(1 - \tau) + n + 1)}{\Gamma(i + 1)\Gamma(n - i + 1)\Gamma(-i(1 - \tau))}.
\]

Using the identity \( \Gamma(-z) = -\frac{\pi}{\sin(\pi z)\Gamma(z+1)} \), we obtain
\[
c_i = (-1)^{i+1} \frac{\Gamma(-i(1 - \tau) + n + 1)\Gamma(i(1 - \tau) + 1)\sin(\pi i(1 - \tau))}{\Gamma(i + 1)\Gamma(n - i + 1)\pi}.
\]

In the following we use \( Q := \frac{\Gamma(-i(1 - \tau) + n + 1)\Gamma(i(1 - \tau) + 1)}{\Gamma(i + 1)\Gamma(n - i + 1)} \).
We now give an upper bound on the absolute value of $c_i$. Start by using the value $\tau = 1 - \sqrt{1 - \beta}$ and let $1 - \beta = e^{-c}$ for some $c > 0$. Consider

$$\ln \frac{\Gamma(ie^{-\frac{\pi}{2}} + 1)}{\Gamma(i + 1)} = \ln(\Gamma(ie^{-\frac{\pi}{2}} + 1)) - \ln(\Gamma(i + 1)) = -\int_0^{i - ie^{-\frac{\pi}{2}}} \Psi(1 + i - t)dt$$

and

$$\ln \frac{\Gamma(-ie^{-\frac{\pi}{2}} + n + 1)}{\Gamma(n - i + 1)} = \ln(\Gamma(-ie^{-\frac{\pi}{2}} + n + 1)) - \ln(\Gamma(n - i + 1)) = \int_0^{i - ie^{-\frac{\pi}{2}}} \Psi(n - i + 1 + t)dt.$$ 

Therefore

$$\ln Q = \int_0^{i - ie^{-\frac{\pi}{2}}} \Psi(n - i + 1 + t) - \Psi(1 + i - t)dt.$$ 

The Digamma function $\Psi$ is increasing and thus the integrand is increasing and we get the approximation

$$\ln Q \leq (i - ie^{-\frac{\pi}{2}})(\Psi(n + 1 - ie^{-\frac{\pi}{2}}) - \Psi(1 + ie^{-\frac{\pi}{2}})).$$ 

Let $i = tn$. Then for fixed $t$ the expression on the right-hand side converges for $n \to \infty$ to

$$\lim_{n \to \infty} (i - ie^{-\frac{\pi}{2}})(\Psi(n + 1 - ie^{-\frac{\pi}{2}}) - \Psi(1 + ie^{-\frac{\pi}{2}})) = ct \ln \left(\frac{1}{t} - 1\right).$$

By numeric computation, the maximum of $t \ln \left(\frac{1}{t} - 1\right)$ in the range $0 < t < 1$ is 0.278. Thus,

$$\ln Q \leq 0.278.$$ 

Putting things together, we have

$$c_i \leq (-1)^{i+1}(1 - \beta)^{-0.278} \sin(\pi i(1 - \tau))/\pi \leq \frac{1}{\pi}(1 - \beta)^{-0.278}.$$ 

The initial problem of estimating the partial fraction expansion from equation (3.20) now states

$$\prod_{k=0}^{n}(m(1 - \tau) + k) - (1 - \tau)^{n+1} \prod_{k=0}^{n}(m + k) = \frac{c_0}{m} + \frac{c_1}{m + 1} + \ldots + \frac{c_n}{m + n} \leq \frac{\sum c_i}{m} \leq \frac{n1(1 - \beta)^{-0.278}}{m}.$$ 

Now we are able to prove Theorem 8.

Proof of Theorem 8: Let $X_i$ be upper bounds on the solutions $x_{i}^{(0)}$, i.e. $|x_{i}^{(0)}| \leq X_i$. As in the bivariate case, we define a collection of polynomials which share the desired common root modulo $p^t$ as

$$g_{i_2, i_3, \ldots, i_n, k}(x_1, \ldots, x_n) := x_{i_2}^{x_{i_3}} \ldots x_{i_n}^{x_{i_n}} f(x_1, \ldots, x_n)^k N_{\max(t-k,0)}.$$
Finding Small Solutions of Modular Equations

where \( i_j \in \{0, \ldots, m-k\} \) such that \( \sum_{j=2}^{n} i_j \leq m-k \). The parameter \( t = \tau m \) represents the power of the used modulus and is by our strategy subject to the optimization.

To see that the coefficient vectors of the polynomials \( g_{i_2,i_3,\ldots,i_n,k}(x_1 X_1, \ldots, x_n X_n) \) form a triangular lattice basis we define an ordering as follows. \( g_{i_2,i_3,\ldots,i_n,k} < g_{i'_2,i'_3,\ldots,i'_n,k'} \) if \( k < k' \) and if \( k = k' \) we use the standard lexicographical monomial order on \( x_2, \ldots, x_n \). Using the shift polynomials in that order from smallest to largest, the constructed lattice basis introduces one new monomial for each shift polynomial and therefore can be represented in a triangular basis matrix. This allows for an easy computation of the determinant of the lattice as product of the diagonal entries. We write

\[
\det (\mathcal{L}) = \prod_{i=1}^{n} X_i^{s_{x_i}} N^{s_N}.
\]

For each monomial \( \omega \) there is a shift polynomial which places \( \omega \cdot N^j \) on the diagonal for some value of \( j \). Thus, to compute the values \( s_{x_i} \) we just need to sum up the exponents of \( x_i \) in all monomials. Because the set of occurring monomials is an \( n \)-dimensional simplex we get from Lemma 3.3.3 the values \( s_{x_i} = \binom{m+n}{m-1} \). Further, the dimension of the lattice equals the number of all lattice points in Lemma 3.3.3, thus \( D = \binom{m+n}{m} \). Notice the relation \( s_{x_i} = \frac{m}{n+1} D \). Finally, \( s_N = m D \tau - \binom{m+n}{m-1} + \binom{m(1-\tau)+n}{m(1-\tau)-1} \).

Now in order to apply Theorem 5 (Howgrave-Graham) to obtain a solution over \( \mathbb{Z} \), we have to ensure that the vectors obtained from the LLL reduction are sufficiently short. The required condition is

\[
2^{\frac{D(D-1)}{2n(n+1)}} \det (\mathcal{L})^{\frac{1}{n+1}} \leq D^{-\frac{1}{2}} N^{\beta \tau m}.
\]

With the previously computed value for the determinant and solved for the \( X_i \), we obtain

\[
\prod_{i=1}^{n} X_i \leq 2^{-\frac{(D-1)(n+1)}{4n}} D^{-\frac{(D-n+1)(n+1)}{2mD}} N^{(\beta \tau m(D-n+1)-mD\tau+\binom{m+n}{m-1}-(m(1-\tau)+n)) \frac{n+1}{mD}}.
\]  

(3.21)

For sufficiently large values of \( N \) we obtain the (in \( m \) asymptotically optimal value \( \tau = 1 - \sqrt{1-\beta} \).

Now we derive similar to the bivariate case a lower bound of the right hand side of Equation (3.21) as follows. At first, we note again that for sufficiently large \( N \), the powers of 2 and \( D \) are negligible and, thus, we consider in the following only the exponent of \( N \)

\[
(\beta \tau m(D-n+1)-mD\tau+\binom{m+n}{m-1}-(m(1-\tau)+n)) \frac{n+1}{mD} = \beta \tau (n+1) - \frac{\beta \tau (n-1)(n+1)}{D} - \tau (n+1) + 1 - \prod_{k=0}^{n-1} (m(1-\tau)+k) \frac{n! mD}{n! mD}.
\]
With $D = \binom{m+n}{m} = \frac{(m+n)!}{m!n!}$ we have

$$\beta \tau (n+1) - \tau (n+1) + 1 - \frac{\beta \tau (n-1)(n+1)!}{\prod_{k=1}^{n}(m+k)} - \frac{\prod_{k=0}^{n}(m(1-\tau) + k)}{\prod_{k=0}^{n}(m+k)}.$$ 

We analyze the last two terms separately. For the first one, if we choose $\tau = 1 - \sqrt{1 - \beta}$ we obtain

$$\beta \tau (n-1)(n+1)! \prod_{k=1}^{n}(m+k) \leq \beta (1 - \sqrt{1 - \beta})(n-1)(n+1)! \frac{(m+1)\prod_{k=2}^{n}k}{(m+1)\prod_{k=2}^{n}k} \leq \beta (1 - \sqrt{1 - \beta})n^2.$$ 

By Lemma 3 we obtain

$$\beta \tau (n-1)(n+1)! \prod_{k=1}^{n}(m+k) \leq - \ln(1 - \beta) \frac{n}{m}.$$ 

For the the second term get by Lemma 4

$$\frac{\prod_{k=0}^{n}(m(1-\tau) + k)}{\prod_{k=0}^{n}(m+k)} \leq (1 - \beta) \frac{n+1}{m} + \frac{1}{\pi} (1 - \beta)^{-0.278} \frac{n}{m}.$$ 

Thus, for sufficiently large $N$ we get the lower bound on the right hand side of Condition [3.21] as

$$2^{-\frac{(D-1)(n+1)}{2m}} D^{-\frac{(n+1)(D-n+1)}{2m}} N^{\beta m \tau (D-n+1) - Dm \tau + \frac{(m+n)}{m-1} - \frac{(m(1-\tau)+n)}{m-1}} \frac{n+1}{m}$$

and, thereby, get a stronger condition on the bounds $X_i$ for being able to efficiently determine the solutions of the initial linear polynomial. Writing the $X_i$ as $N^{\gamma_i}$ and switching to the condition in the exponents, we obtain

$$\sum_{i=1}^{n} \gamma_i \leq 1 - (1 - \beta) \frac{n+1}{m} - (n+1)(1 - \sqrt{1 - \beta})(1 - \beta) - \frac{n}{m} (1 - \beta)^{-0.278} + \beta \ln(1 - \beta) \frac{n}{m}.$$ 

Thus, if we identify the part depending on $\frac{1}{m}$ as error term $\epsilon$ we get the condition as stated in the theorem.

$$\sum_{i=1}^{n} \gamma_i \leq 1 - (1 - \beta) \frac{n+1}{m} - (n+1)(1 - \sqrt{1 - \beta})(1 - \beta) - \epsilon$$

The error term $\epsilon$ then tells us how to choose the parameter $m$ in order to achieve a certain exact bound.

$$\epsilon \leq - \frac{n}{m} (1 - \beta)^{-0.278} + \beta \ln(1 - \beta) \frac{n}{m} + \beta \ln(1 - \beta) \frac{n}{m}$$

$$\Leftrightarrow m \geq \frac{n}{\epsilon} \left( \frac{1}{\pi} (1 - \beta)^{-0.278} - \beta \ln(1 - \beta) \right) = O \left( \frac{n}{\epsilon} \right)$$
To conclude the proof we investigate the complexity of the algorithm. The running time is again dominated by the lattice reduction step which is polynomial in the lattice dimension and the bit size of the entries in the basis matrix. We notice that the dimension \( D = \binom{m+n}{m} = \mathcal{O}\left(\frac{2^n}{m^2}\right) = \mathcal{O}\left(\frac{n^2}{m}\right) \). For the entries in the basis matrix we note that the coefficients of \( f^k \) can be reduced modulo \( N^k \). To estimate the size of the entries in \( f(X_1 x_1, \ldots, X_n x_n)^k \) we further note that trivially \( X_i \leq N \) and therefore each entry is at most of size \( \mathcal{O}(2k \log N) \). The multiplication of \( f^k \) with a monomial \( x_2^{i_1} \ldots x_n^{i_n} \) increases the size by at most \( \mathcal{O}((m - k) \log N) \) since \( \sum_{i=2}^{n} t_j \leq m - k \). Finally, the shift polynomial is multiplied by a factor \( N^{\max(t-k,0)} \) and therefore increases the size of the entries by \( \mathcal{O}((m - k) \log N) \).

All in all, the entries of the basis matrix are of size at most \( \mathcal{O}(2^m \log N) \) and thus as required polynomial in \( \log N \).

**Experiments**

Remember that we need to find \( n \) sufficiently small vectors in the lattice such that we obtain \( n \) polynomials over the integers with the same small root as the initial polynomial. However, it is not sufficient that we find enough small vectors, but we also require that the ideal generated by the corresponding polynomial equations is zero-dimensional, i.e. the variety is a finite set. Unfortunately, we cannot prove this and therefore we have to rely on Assumption [1]. In order to verify this assumption it is necessary to conduct a series of experiments. Thus, we implemented our lattice-based algorithm using the \( L^2 \)-algorithm from Nguyen, Stehlé [NS05]. We tested the algorithm for instances of the *factoring with known bits* problem with \( n = 2, 3 \) and 4 blocks of unknown bits. Table 3.1 shows the experimental results for a 512-bit RSA modulus \( N \) with divisor \( p \) of size \( p \geq N^{\frac{1}{4}} \).

For given parameters \( m, t \) we computed the number of bits that one should theoretically be able to recover from \( p \) (column `pred` of Table 3.1). For each bound we made two experiments (column `exp`). The first experiment splits the bound into \( n \) equally sized pieces, whereas the second experiment unbalancedly splits the bound in one large piece and \( n - 1 \) small ones. In the unbalanced case, we were able to recover a larger number of bits than theoretically predicted.

This is consistent with Remark 3.3.2. I.e. for unbalanced variables we are able to improve the bounds if we adapt the strategy of the selection of shift polynomials.

In all of our experiments, we successfully recovered the desired small root, thereby deriving the factorization of \( N \). We were able to extract the root both by Gröbner basis reduction as well as by numerical methods in a fraction of a second.

For Gröbner basis computations, it turns out to be useful that our algorithm actually outputs more sufficiently small norm polynomials than predicted by the LLL-bounds. This in turn helps to speed up the computation a lot.

As a numerical method, we used multidimensional Newton iteration on the starting point \( \frac{1}{2}(X_1, \ldots, X_n) \). Usually this did already work. If not, we were successful with the vector of upper-bounds \( (X_1, \ldots, X_n) \) as a starting point. Although this approach worked well and highly efficient in practice, we are unaware of a starting point that provably lets the Newton method converge to the desired root.

\[\text{to be precise, the coefficients can be reduced modulo } N^{\max(k,t)}\]
Though Assumption 1 worked perfectly for the described experiments, we also considered two pathological cases, where one has to take special care.

First, a problem arises when we have a prediction of $k$ bits that can be recovered, but we use a much smaller sum of bits in our $n$ blocks. In this case, the smallest vector lies in a sublattice of small dimension. As a consequence, we discovered that then usually all of our small norm polynomials shared $f(x)$ as a common divisor. When we removed the gcd, the polynomials again generated a 0–dim ideal and we were able to retrieve the root efficiently. Notice that removing $f(x)$ does not eliminate the desired root, since $f(x)$ does not contain the root over the integers (but mod $p$).

A second problem may arise in the case of two closely adjacent unknown blocks, e.g. two blocks that are separated by one known bit only. Since in comparison with the $n$-block case the case of $n−1$ blocks gives a superior bound, it turns out to be better in some cases to merge two closely adjacent blocks into one variable. That is what implicitly seems to happen in our approach. The computations then yield the desired root only in those variables which are sufficiently separated. The others have to be merged before re-running the algorithm in order to obtain all the unknown bits. Alternatively, we confirmed experimentally that merging the nearby blocks from the beginning immediately yields the desired root.

Both pathological cases are no failure of Assumption 1, since one can still easily extract the desired root. All that one has to do is to either remove a gcd or to merge variables.

### Application to Factoring with Known Bits

A natural application of Theorem 3 occurs in the context of factoring with known bits. Recall that this problem basically analyzes the amount of additional information that is required to factorize an RSA modulus $N = pq$ efficiently, i.e. in polynomial time in the bit length of $N$. This information is given in terms of bits of one of the prime factors.

We already mentioned this problem in Section 3.2. However, in the analysis of Coppersmith [Cop96a] by finding a small root of a bivariate polynomial over the integers, and in the analysis in the language of finding solutions modulo unknown divisors [HG97], it is assumed that the unknown bits constitute only one consecutive block.

Usually, a practical motivation to consider the problem of factoring with known
bits is given by side channel attacks. A weak implementation of a cryptosystem may leak parts of the secret key to an attacker and in the case of RSA this may give the attacker some bits of a prime factor. From a practical point of view, however, there is no justification to why the given bits should be distributed such that there is only one unknown block of bits left.

This makes our theorem interesting as it allows to interpret each missing block of bits as one variable $x_i$. However, there is one limiting fact in our algorithm. Namely, the running time is exponential in the number $n$ of variables. Thus, in order to obtain a polynomial complexity one has to restrict $n = O\left(\frac{\log \log N}{1 + \log(\frac{1}{\epsilon})}\right)$.

Thus, for any constant error term $\epsilon$, our algorithm is polynomial time whenever $n = O(\log \log N)$.

The implication of our algorithm for the problem of factoring with known bits is asymptotically analyzed in the following theorem. It shows that the bound $X_1 \cdots X_n$ converges for $n \to \infty$ to $N^{\beta + (1 - \beta) \ln(1 - \beta)}$. For the special case of an RSA modulus $N$ with divisor $p \geq N^\beta$ where $\beta = \frac{1}{2}$, this yields the bound $N^{\frac{1}{2}(1 - \ln(2))} \approx N^{0.153}$. This in turn means that we are able to recover a $(1 - \ln(2)) \approx 0.306$-fraction of the bits of $p$, no matter how the unknown bits are distributed in $p$. In other words, an $\ln(2) \approx 0.694$-fraction of the bits of $p$ have to be known in order to recover the whole factor.

**Theorem 9.** Let $\epsilon > 0$ and suppose $N$ is a sufficiently large composite integer of unknown factorization with a divisor $p \geq N^\beta$. Further suppose we are given a fraction of $(1 - \frac{1}{\beta}) \ln(1 - \beta) + \epsilon$ of the bits of $p$ distributed among $n$ blocks of known location. Then, under Assumption 7, we can compute the unknown bits of $p$ in polynomial time in $\log(N)$ and $(\frac{e}{\epsilon})^n$, where $e$ is Euler’s constant.

**Proof:** From Theorem 8, we know that we can compute a solution to the equation

$$a_1 x_1 + a_2 x_2 + \ldots + a_n x_n + a_{n+1} = 0 \mod p$$

as long as the product of the unknowns is smaller than $N^\gamma$, where $\gamma = \sum_{i=1}^n \gamma_i$ is upper bounded as in 3.19. As noted in the previous discussion, this bound for $\gamma$ actually converges for $n \to \infty$ to a value different from zero. Namely,

$$\lim_{n \to \infty} \left(1 - (1 - \beta)\frac{n+1}{n} - (n - 1)(1 - \sqrt{1 - \beta})(1 - \beta)\right)$$

$$= \beta + (1 - \beta) \ln(1 - \beta).$$

Hence, this is the portion of $p$ that we can at least compute, no matter how many unknowns we have. Conversely, once we have $((\beta - 1) \ln(1 - \beta) + \epsilon) \log N$ bits of $p$ given together with their positions, we are able to compute the missing ones. Since $\log N \leq \frac{\log p}{\beta}$, we need at most an $\left(1 - \frac{1}{\beta}\right) \ln(1 - \beta) + \epsilon$-fraction of the bits of $p$. 


Application to Fault Attacks on RSA Signatures (cf. [CJK+09]) A widely used signature scheme is based on the RSA assumption. In the most simple case, a signer creates a signature by applying some encoding function $\mu$ to the message and finally computes the signature as

\[ \sigma = \mu(m)^d \mod N. \] (3.25)

In 1999 Boneh, DeMillo and Lipton [BDL97, BDL01] showed that implementations of this scheme which use the Chinese Remainder Theorem are vulnerable to so-called fault attacks. In a nutshell, in a fault attack an attacker tampers with the hardware during a computation such that the result of this computation becomes false. This can be done for example by temporarily increasing the current or heating up the chip with a laser.

To increase efficiency of the signature generation one might compute values $\sigma_p = \mu(m)^d \mod p$ and $\sigma_q = \mu(m)^d \mod q$, and use the Chinese Remainder Theorem to finally combine $\sigma_p$ and $\sigma_q$ to obtain the signature value $\sigma$. If an attacker manages to induce faults in the computation of one of the values, w.l.o.g. $\sigma_q$ and leave the other one untouched, then the resulting signature will satisfy

\[ \sigma^{\prime e} = \mu(m) \mod p \quad \sigma^{\prime e} \neq \mu(m) \mod q. \] (3.26)

This observation now allows the attacker to recover the factorization of the modulus $N$ by simply computing

\[ \gcd(\sigma^{\prime e} - \mu(m) \mod N, N). \] (3.27)

Note that this attack also works for any deterministic encoding, e.g. Full Domain Hash [BR96], where the signature is computed by $\sigma = H(m)^d \mod N$ for a hash function $H : \{0, 1\}^* \rightarrow \mathbb{Z}_N^*$. It even works for randomized encodings as long as the nonce used for randomization is sent along with the signature. An example for that is the Probabilistic Full Domain Hash [Cor02], where the value $\sigma$ is computed by $\sigma = H(m||r)^d \mod N$ and the actual signature contains $\sigma||r$.

However, once the underlying message is not completely given to the adversary, this attack is thwarted. For example in the previous case of Probabilistic Full Domain Hash if $r$ is unknown, then the attacker is unable to compute

\[ \gcd(\sigma^{\prime e} - (m||r) \mod N, N). \] (3.28)

Coron, Joux, Kizhvatov, Naccache and Paillier describe in [CJK+09] an successful attack on a signature mechanism, where the underlying message is only partially known. They consider the ISO/IEC 9796-2 standard, which encodes a message $m = m_1||m_2$ as

\[ \mu(m) = 6A_{16}||m_1||H(m)||BC_{16}. \] (3.29)

Such an encoding is used if bandwidth is a scarce resource, since it allows a partial message recovery. I.e. it is not necessary to send the whole message along with the signature because the $m_1$ part can be recovered from the signature itself.

In their attack, Coron et al. show that if the unknown part of $m_1$ is not too large then the previously described algorithm for finding small solutions of equations
modulo unknown divisors can be used to recover all of \( m_1 \) and thereby have access to the complete underlying message. This again enables to find the factorization of \( N \) by a gcd computation.

To be more precise, suppose \( m_1 = \alpha||r||\alpha' \) with known values \( \alpha \) and \( \alpha' \), then we have

\[
\sigma^e = \mu(m) = 6A_{16}|\alpha||r||\alpha'|-1H(m)|\cdot BC_{16} \mod p \tag{3.30}
\]

with unknown values \( r \) and \( H(m) \). This can be transformed into a bivariate linear polynomial with root \((r, H(m))\) modulo \( p \). If the size of the unknown message part \( r \) and the size of the hash value \( H(m) \) fulfill the condition given in Theorem 7 then this single faulty signature allows to factorize the modulus \( N \).

It is straightforward to extend the attack to the case of multiple unknown message parts, i.e. \( m_1 = \alpha_1||r_1||\alpha_2||r_2||\ldots||r_n||\alpha_{n+1} \) with unknowns \( r_1, \ldots, r_n \), using Theorem 8.

### 3.4 Coppersmith’s Algorithm: Integer Variant

Coppersmith [Cop96a, Cop97] also presented an algorithm to solve the problem of finding small solutions of bivariate equations over the integers. This problem is more general than finding solutions of univariate modular equations. Indeed, we can easily transform the problem of finding solutions of the polynomial \( f(x) = 0 \mod N \) into an equation in two variables over the integers as \( f(x) - yN = 0 \).

Remark. In this subsection we present Coppersmith’s result in the original notation which is quite different from the one of Howgrave-Graham [HG97] for the modular case. However, a formulation similar to the one of Howgrave-Graham has been given by Coron [Cor04, Cor07].

Let \( f(x, y) \in \mathbb{Z}[x, y] \) be a polynomial in two variables with integer coefficients that is irreducible over the integers. In order to find all small solution \( |x^{(0)}| \leq X, |y^{(0)}| \leq Y \) of the bivariate polynomial equation \( f(x, y) = 0 \), Coppersmith constructs a second polynomial \( g(x, y) \) which shares the small roots \((x^{(0)}, y^{(0)})\) with \( f(x, y) \) but is not a multiple of \( f(x, y) \). To construct this polynomial we use again a collection of shift polynomials. In the current integer case, however, this collection just consists of polynomial multiples of the original polynomial \( f(x, y) \) and not powers of it. Thus, we sometimes speak of a shift monomial if we refer to the polynomial factor of the shift polynomial. For example, the shift polynomial \( x^2 y \cdot f(x, y) \) has the shift monomial \( x^2 y \).

Given a collection \( C \) of shift polynomials, let \( M \) be the set of all monomials occurring in the shift polynomials and \( S \) the set of all shift monomials. Further, let \( W \) be the largest coefficient of \( f(xX, yY) \), i.e. \( W := \|f(xX, yY)\|_{\infty} \), and let \( \lambda \) denote the monomial that belongs to \( W \). Define an ordering \( m_1, \ldots, m_{|C|} \) on the shift monomials, such that \( m_i \lambda \) is not a monomial of the shift polynomial \( m_j f(x, y) \) for all \( j < i \). Denote by \( AS \) the set \( \{\lambda m_i | m_i \in S\} \). To construct an \( |M| \times |M| \) basis matrix of a lattice we label the first \( \omega := |M \setminus S| \) rows by the monomials in \( M \setminus \lambda S \) in any order, say \( m_{|C|+1}, \ldots, m_{|M|} \), and the last \( |S| \) rows in the order defined on the shift monomials. The upper left \( \omega \times \omega \) submatrix is a diagonal matrix where the diagonal elements are the inverses of the upper bounds on the monomial of that
row label. Then, we take the shift polynomials according to the order on the shift monomials and write for the $i$-th shift polynomial the coefficient of monomial $j$ in column $\omega + i$ and the row labeled by monomial $j$.

**Example:** Let us look at an example for the construction of the basis matrix for finding small roots $x^{(0)}, y^{(0)}$ of the polynomial $f(x, y) = xy + ax + by + c$ with $|x^{(0)}| \leq X$ and $|y^{(0)}| \leq Y$. Suppose the largest coefficient of $f(xX, yY)$ is given by $bY$ and thus $\lambda = y$. For the collection $C = \{xf, f, xyf, yf\}$ with the order $x < 1 < xy < y$ on the shift monomials, the basis matrix $B$ of the lattice $\mathcal{L}$ is given by

$$
\begin{pmatrix}
1 & 1 & c \\
x & c & a \\
x^2 & a & 1 \\
x^2y & 1 & a \\
x^2y^2 & \frac{1}{x^2y^2} & 1 \\
x y & b & 1 \\
x y^2 & \frac{1}{x y^2} & b \\
y & b & bc \\
y^2 & b & 1
\end{pmatrix}
$$

By construction, i.e. because of the order on the shift monomials, the basis matrix
is upper triangular and it is straightforward to compute the determinant as
\[
\det(L) = b |S| \prod_{x^i y^j \in M \setminus S} \frac{1}{X^i Y^j} = W |S| \prod_{x^i y^j \in M \setminus S} \frac{1}{X^i Y^j},
\]
where \( W := \| f(x, y) \|_{\infty} \), i.e. in this case \( W = |b y| \).

Now consider the lattice vector
\[
v = \left( m_{|C|+1}^{(0)}, \ldots, m_{|M|}^{(0)}, m_{1}^{(0)}, \ldots, m_{|C|}^{(0)} \right) \cdot L = \left( \frac{m_{|C|+1}^{(0)}}{M_{|C|+1}}, \ldots, \frac{m_{|M|}^{(0)}}{M_{|M|}}, 0, \ldots, 0 \right),
\]
where \( m_i^{(0)} \) and \( M_i \) denote the monomial \( m_i \) evaluated at a small solution \( x^{(0)}, y^{(0)} \) and at the upper bounds \( X, Y \), respectively. The lattice vector \( v \) lies in a sublattice with the last \( |S| \) coordinates equal to zero. Further, the vector is relatively short because each component is upper bounded by \( \frac{|m_i^{(0)}|}{M_i} \leq 1 \). Thus,
\[
\|v\| \leq \sqrt{\omega}.
\]

Notice that once we found the vector \( v \), the solution \((x^{(0)}, y^{(0)})\) is usually easily extracted by multiplying the certain entries \( v_i \) in \( v \) by the appropriate \( M_i \). In order to find \( v \), Coppersmith proposes the following strategy. Since we know that the vector \( v \) is contained in the sublattice that has its last components equal to zero, we perform elementary operations on the basis matrix to transform it into the form
\[
\begin{pmatrix}
A & B \\
0 & D
\end{pmatrix} \Rightarrow \begin{pmatrix}
A' & 0 \\
C' & I_{|S|}
\end{pmatrix},
\]
where \( I_{|S|} \) is the \(|S| \times |S| \) identity matrix. Whether such a transformation is possible depends on the chosen collection of shift polynomials. Ritzenhofen [Rit10] introduced the notion of a determinant-preserving collection of shift polynomials to denote the existence of such a transformation, and gave certain general conditions for being determinant-preserving. However, the strategy for selecting shift polynomials that we consider in the next subsection does by construction lead to a determinant-preserving collection of shift polynomials.

Suppose we have a determinant-preserving collection of shift polynomials. Then, in the new form \([3.33]\) the first \( \omega \) rows define a basis of the sublattice that has the last \( |S| \) entries equal to zero. Furthermore, the vector \( \hat{v} \), which is \( v \) shortened to the first \( \omega \) components, is a vector in the lattice \( L_s \) defined by basis matrix \( A' \). Because in the new form the lower right block is the identity matrix, it follows
\[
\det(L_s) = \det(L).
\]

Now perform the LLL lattice reduction algorithm on \( A' \) to obtain a reduced basis \((b_1, \ldots, b_\omega)^T\). Note that by Lemma [1] we have a lower bound on the length of the Gram-Schmidt orthogonalization of the last basis vector.
\[
\|b_\omega^*\| \geq 2^{-\frac{\omega-1}{4}} \det(L_s)^{\frac{1}{2}}
\]
We already mentioned that the vector \( v \) and thus also the shortened version \( \hat{v} \) are rather small, namely \( \| \hat{v} \| = \|v\| \leq \sqrt{\omega} \). Further, Coppersmith noticed that a sufficiently short lattice vector lies in a certain hyperplane.
Lemma 5. Let \((b_1, \ldots, b_ω)\) be an LLL-reduced lattice basis of a lattice \(L\). If a lattice element \(s\) satisfies \(\|s\| < 2^{-\frac{ω - 1}{4}} \det (L)^{\frac{1}{2}}\) then \(s\) lies in the hyperplane spanned by \(b_1, b_2, \ldots, b_ω - 1\).

Proof: Each lattice element \(s\) can be expressed as integral linear combination \(s = \sum s_i b_i\) of the basis vectors. Notice that the norm of \(s\) is at least as large as the contribution of the orthogonal component of \(b_ω\), i.e. \(\|s\| \geq |s_ω| \cdot \|b_ω^∗\|\). Conversely, this means that if \(s\) satisfies \(\|s\| < \|b_ω^∗\|\) then \(s_ω\) must be equal to 0, because it is an integer. From Lemma 1 we know that \(\|b_ω^∗\|\) is lower bounded by \(2^{-\frac{ω - 1}{4}} \det (L)^{\frac{1}{2}}\) and therefore \(s\) must lie in the hyperplane spanned by \(b_1, b_2, \ldots, b_ω - 1\) if \(\|s\| < 2^{-\frac{ω - 1}{4}} \det (L)^{\frac{1}{2}}\).

Thus, the vector \(\hat{v}\) from above, lies in a hyperplane of the sublattice \(L_s\), if \(\|\hat{v}\| < 2^{-\frac{ω - 1}{4}} \det (L_s)^{\frac{1}{2}}\). Furthermore, if \(\hat{v}\) fulfills this condition, then this implies that \(\hat{v}\) is orthogonal to \(b_ω^∗\), i.e. the inner product \(\langle \hat{v}, b_ω^∗ \rangle\) equals zero. Thus, computing the vector \(b_ω^∗\) via the Gram-Schmidt orthogonalization of the reduced lattice basis and taking the unknowns \(x^{(0)}\) and \(y^{(0)}\) in \(\hat{v}\) as variables \(x\) and \(y\), we obtain from the inner product \(\langle \hat{v}, b_ω^∗ \rangle\) the desired second equation \(g(x, y)\) that shares the small root \(x^{(0)}\) and \(y^{(0)}\) with \(f(x, y)\). Note that this is true for all small roots. In \([BM05]\) it is proved that the resultant of \(f(x, y)\) and \(g(x, y)\) indeed does not vanish and allows to efficiently find the solution \((x^{(0)}, y^{(0)})\).

Remember that the condition for the vector \(\hat{v}\) to lie in a hyperplane is given by \(\|\hat{v}\| < 2^{-\frac{ω - 1}{4}} \det (L_s)^{\frac{1}{2}}\), and that by construction the vector \(\hat{v}\) has norm smaller than \(\sqrt{ω}\). Further, we have \(\det (L_s) = \det (L)\) and, thus, \(\hat{v}\) fulfills the requirement of Lemma 5 if

\[
\sqrt{ω} < 2^{-\frac{ω - 1}{4}} \det (L)^{\frac{1}{2}}.
\]

Similar to the modular case, we call this enabling condition. Letting all values that do not depend on the size of the coefficients of the input polynomial contribute to an error term \(\epsilon\), we obtain the simplified enabling condition

\[
1 < \det (L) W^{-\epsilon}.
\]

In upcoming chapters we will often implicitly assume the factor \(W^{-\epsilon}\) and write as simplified enabling condition just

\[
1 < \det (L).
\]

(3.36)

Recall that in \((3.31)\) we have derived \(\det (L) = W^{|S|} \prod_{x^iy^j \in M \setminus S} \frac{1}{X^iY^j}\), which then leads to the simplified condition

\[
1 < W^{|S|} \prod_{x^iy^j \in M \setminus S} \frac{1}{X^iY^j} W^{-\epsilon},
\]

or

\[
X^{s_x} Y^{s_y} < W^{|S| - \epsilon}, \text{ where } s_x = \sum_{x^iy^j \in M \setminus S} i, \quad s_y = \sum_{x^iy^j \in M \setminus S} j.
\]
With this condition we are able to compute for a specific collection of shift polynomial the upper bounds $X$ and $Y$ such that we can efficiently find all small solutions $|x(0)| \leq X$ and $|y(0)| \leq Y$.

Similar to the modular case, this technique can be generalized to handle multivariate equations. For finding all small solution of the equation $f(x_1, \ldots, x_n) = 0$, all steps of the algorithm stay the same, however, in order to finally compute the small solutions $(x(0)_1, \ldots, x(0)_n)$ we require $n - 1$ additional polynomials with the same root over the integers such that Assumption [4] is fulfilled. To that end, Coppersmith presented a generalization of Lemma [5].

**Lemma 6.** Let $(b_1, \ldots, b_\omega)$ be an LLL-reduced basis of a lattice $\mathcal{L}$. If a lattice element $s$ satisfies $\|s\| < \|b^*_i\|$ for all $i = k + 1, \ldots, \omega$, then $s$ lies in the space spanned by $b_1, \ldots, b_k$.

The proof of Lemma [6] is basically an iterated version of the proof of Lemma [5].

A lemma due to Jutla [Jut98] estimates the size of the Gram-Schmidt orthogonalized vectors $b^*_i$.

**Lemma 7** (Jutla). Let $A = (v_1, \ldots, v_\omega)$ be a basis of a lattice $\mathcal{L}$. Denote by $b_{\text{max}}$ the maximal length of the Gram-Schmidt orthogonalization of $A$, i.e. $\max_i \|v^*_i\|$. For an LLL-reduced basis $B = (b_1, \ldots, b_\omega)$ of the lattice $\mathcal{L}$ it holds that

$$\|b^*_i\| \geq 2^{-\frac{i-1}{4}} \left( \frac{\det(\mathcal{L})}{b_{\text{max}}^{\omega-1}} \right)^{\frac{1}{\omega-n+1}}.$$  

Thus, in order to obtain $n - 1$ polynomials over the integers that share the common root $(x(0)_1, \ldots, x(0)_n)$ with $f(x_1, \ldots, x_n)$, we require the last $n - 1$ orthogonalized basis vectors of the reduced basis to be larger than the vector $\hat{v}$. Thus, the enabling condition becomes

$$\|\hat{v}\| = \sqrt{\omega} \leq 2^{-\frac{n}{4}} \left( \frac{\det(\mathcal{L})}{b_{\text{max}}^{\omega-1}} \right)^{\frac{1}{\omega-n+1}}.$$  

(3.37)

### 3.4.1 Jochemsz-May Strategy for Small Integer Roots

In Section 3.2 we have presented a general strategy for the selection of the shift polynomials in the multivariate modular setting. Jochemsz and May [JM06] also introduced an according strategy for the case of equations over the integers. In their work they present this strategy in the framework of Coron [Cor04]. We will slightly deviate from the presentation in [JM06] and present their result in the original Coppersmith notation.

Their basic strategy proceeds as follows. Fix an integer $m$ and define the set $S$ of shift monomials as monomials of $f^{m-1}$. Thus, the shift polynomials are given by

$$g_{i_1, \ldots, i_n} := x_1^{i_1} \ldots x_n^{i_n} f(x_1, \ldots, x_n).$$  

(3.38)

Then the set $M$ of all monomials occurring in the set of shift polynomials is given by the monomials of $f^m$. Following the lattice construction described in the previous section, we obtain a lattice with determinant equal to

$$\det(\mathcal{L}) = W_{\{S\}} \prod_{x_1^{i_1} \ldots x_n^{i_n} \in M \setminus S} \frac{1}{X_1^{i_1} \ldots X_n^{i_n}}.$$  

(3.39)
Remark. To be perfectly precise, we obtain this determinant if there exists an ordering \( m_1, \ldots, m_{|S|} \) of the shift polynomials, such that \( m_j \lambda \) is not a monomial of the shift polynomial \( m_j f(x_1, \ldots, x_n) \) for all \( j < i \), where \( \lambda \) is the monomials with largest coefficient in \( f(x_1 X_1, \ldots, x_n X_n) \). This gives us a triangular basis matrix. Otherwise, if no such ordering exists, a proof of Coppersmith \([\text{Cop97, Lemma 3}]\) can be generalized to show that the determinant is still large in that case. However, since in our applications we have an appropriate ordering, we omit the details of a general analysis.

Proceeding with Coppersmith’s algorithm as described in 3.4 we know that there exists a short vector \( v \) with zeros in its last \(|S|\) components. Transforming the lattice basis according to (3.33) and performing an LLL reduction on the basis matrix \( A' \) we obtain a reduced basis \((b_1, \ldots, b_\omega)^T\) of a sublattice \( \mathcal{L}_s \), where \( \omega := |M \setminus S| \).

By Lemma 5 we know that the shortened version \( \hat{v} \) of \( v \) is orthogonal to \( b^*_\omega \) if \( \|\hat{v}\| < 2^{-\omega-n/4} \det(\mathcal{L})^{1/\omega} \). Since we require \( n-1 \) new polynomials with the same root over the integers, we use Lemma 7 to derive the condition

\[
\|\hat{v}\| = \sqrt{\omega} \leq 2^{-\omega-n/4} \left( \frac{\det(\mathcal{L})}{b^{n-1}_{\max}} \right)^{1/\omega-n+1}.
\]

Unfortunately, we cannot give a useful estimate for the value \( b_{\max} \) in general. We require that the contribution of the \( b_{\max} \) term is negligible compared to the determinant of the lattice. This has to be verified for a specific lattice construction.

If this is fulfilled, we let \( b^{n-1}_{\max} \) as well as terms that do not depend on \( W \) contribute to an error term to obtain again the simplified condition

\[ 1 < \det(\mathcal{L}) W^{-\epsilon}. \]

With the expression for the determinant given in (3.39) we get the final condition

\[
\prod_{j=1}^{n} X^{s_j}_j < W^{s_w}, \text{ for } s_j = \sum_{x_1^1 \cdots x_n^\omega} x_i \text{ and } s_w = |S|. \tag{3.40}
\]

Sometimes it is beneficial to enlarge the collection of shift polynomials by so-called extrashifts in certain variables (cf. Section 3.2). In that case, we define the set \( S \) of shift polynomials slightly different. For example, to include extrashifts in the variable \( x_1 \), we define

\[
S := \bigcup_{0 \leq j_1 \leq t} \left\{ x_1^{i_1+j_1} x_2^{i_2} \cdots x_n^{i_n} | x_1^{i_1} x_2^{i_2} \cdots x_n^{i_n} \text{ is a monomial of } f^{m-1} \right\}
\]

The definition of \( M \) follows as in the basic strategy as

\[
M := \left\{ \text{monomials of } x_1^{i_1} x_2^{i_2} \cdots x_n^{i_n} \cdot f | x_1^{i_1} x_2^{i_2} \cdots x_n^{i_n} \in S \right\}.
\]

As the final condition (3.40) is expressed in terms of \( S \) and \( M \) it remains the same for the extended strategy using extrashifts.
3.4.2 Embedding the Modular Case

In this thesis we will sometimes prefer the original Coppersmith notation over the one from Howgrave-Graham presented in Section 3.1 even in case of finding small solutions of modular equations. It is obvious that any modular equation \( f_N(x_1, \ldots, x_n) = 0 \mod N \) can be easily transformed into an equation over the integers by introducing a new variable \( y \) identifying the modular reduction.

\[
f_N(x_1, \ldots, x_n) = yN
\]

Given a collection of shift polynomials derived using the strategy of Jochemsz-May in the modular case, we can construct a lattice basis in the original Coppersmith notation. Recall, that in the strategy for selecting shift polynomials in the modular case, we fixed a leading monomial \( \lambda \) and defined for \( k = 0, \ldots, m \) the sets

\[
M_k := \{ x_1^{i_1} x_2^{i_2} \cdots x_n^{i_n} \mid x_1^{i_1} x_2^{i_2} \cdots x_n^{i_n} \text{ is a monomial of } f_N^{m} \}
\]

and \( \frac{x_1^{i_1} x_2^{i_2} \cdots x_n^{i_n}}{\lambda^k} \) is a monomial of \( f_N^{m-k} \). From these sets, we define a collection \( C \) of shift polynomials for a Coppersmith lattice via

\[
g_{i_1, \ldots, i_n, k}(x_1, \ldots, x_n) := \frac{x_1^{i_1} x_2^{i_2} \cdots x_n^{i_n}}{\lambda^k} \left( f_N(x_1, \ldots, x_n)^{k} - (yN)^k \right),
\]

for \( k = 1, \ldots, m \) and \( x_1^{i_1} x_2^{i_2} \cdots x_n^{i_n} \in M_k \setminus M_{k+1} \).

The collection of shift polynomials is used to construct an \( |M_0| + |C| \times |M_0| + |C| \) basis matrix, where the first \( |M_0| \) rows are labeled by the monomials in \( M_0 \). The upper left \( |M_0| \times |M_0| \) block of the basis matrix is filled with the inverses of the upper bounds of the monomial corresponding to that rows. The rightmost \( |C| \) columns of the basis matrix are filled with the coefficients of the shift polynomials – one polynomial per column – and the coefficient of monomial \( m_j \) placed in the row labeled with \( m_j \). Further, in the column of the shift polynomial \( g_{i_1, \ldots, i_n, k} \) we use the term \( N^k \) as diagonal entry.

**Example:** For the polynomial \( f_N(x_1, x_2) = x_1x_2 + ax_1 + b \) with a small root \( (x_1^{(0)}, x_2^{(0)}) \) modulo \( N \) with \( |x_1^{(0)}| \leq X_1, |x_2^{(0)}| \leq X_2 \), we select the leading monomial \( x_1x_2 \) and fix an integer \( m = 2 \). The Jochemsz-May strategy defines sets

\[
M_0 = \{ 1, x_1, x_1^2, x_1x_2, x_2^2, x_2x_1^2, x_2x_2^2 \}
\]

\[
M_1 = \{ x_1x_2, x_1^2x_2, x_2^2x_2 \}
\]

\[
M_2 = \{ x_1^2 \}
\]

\[
M_3 = \{ \}
\]

and we obtain the collection of shift polynomials for the Coppersmith construction as

\[
g_{1,1,1} = f_N(x_1, x_2) - yN
\]

\[
g_{1,2,1} = x_2 f_N(x_1, x_2) - x_2 yN
\]

\[
g_{2,2,2} = f_N(x_1, x_2) - y^2 N^2
\]
Building the lattice basis for the collection $C = \{g_{1,1,1}, g_{1,2,1}, g_{2,2,2}\}$ we obtain

$$
\begin{bmatrix}
1 & 1 \\
x_1 & x_1 \\
x_1x_2 & x_1x_2 \\
x_1^2 & x_1^2 \\
x_1^2x_2 & x_1^2x_2 \\
-x_1 & -x_1 \\
-x_1y & -x_1y \\
-y^2 & -y^2
\end{bmatrix}
\begin{bmatrix}
1 \\
1 \\
1 \\
1 \\
1 \\
1 \\
1 \\
1 \\
1
\end{bmatrix}
\begin{bmatrix}
\frac{1}{x_1} \\
\frac{1}{x_1x_2} \\
\frac{1}{x_1^2x_2} \\
\frac{1}{x_1^2x_2^2} \\
\frac{1}{x_1^2x_2^2} \\
\frac{1}{x_1^2x_2^2} \\
\frac{1}{x_1^2x_2^2} \\
\frac{1}{x_1^2x_2^2} \\
\frac{1}{x_1^2x_2^2}
\end{bmatrix}
\begin{bmatrix}
b \\
a \\
a \\
b \\
b \\
a \\
a \\
a \\
a
\end{bmatrix}
\begin{bmatrix}
a \\
b \\
2b \\
a \\
a \\
2b \\
a \\
a \\
a
\end{bmatrix}
$$

The described lattice basis construction naturally results in an upper triangular basis matrix. Thus, in order to compute the determinant $\det(L)$ we just need to find the product of the upper bound for all monomials that appear in the set $M_0$ from the Jocomsz-May strategy, and we need to compute the product of all occurring moduli. The latter can be derived from the sets $M_k$ as

$$
\sum_{k=1}^{m} k (|M_k| - |M_{k+1}|) = \sum_{k=1}^{m} |M_k|.
$$

The simplified enabling condition $\det(L) > 1$ then reduces to

$$
X_1^{s_1} \ldots X_n^{s_n} < N^{s_n}, \quad \text{where} \quad \begin{cases} 
 s_j = \sum_{x_1^{i_1} x_2^{i_2} \ldots x_n^{i_n} \in M_0} i_j \\
 s_N = \sum_{k=1}^{m} |M_k|
\end{cases},
$$

which matches the condition obtained in the analysis in Section 3.2.

**Remark.** We often describe the shift polynomials in the collection $C$ simply by

$$
g_{i_1, \ldots, i_n, k}(x_1, \ldots, x_n) := \frac{x_1^{i_1} x_2^{i_2} \ldots x_n^{i_n}}{\lambda^k} f_N(x_1, \ldots, x_n)^k,$$

i.e. we omit the term $N^k \frac{x_1^{i_1} x_2^{i_2} \ldots x_n^{i_n}}{\lambda^k} y^k$. We can do this because the coefficient $N^k$ is implicitly given by the power of the polynomial $f_N$ and the corresponding variable is irrelevant for Coppersmith’s algorithm.
Chapter 4

Unravelled Linearization

In the previous chapter we saw two basic approaches to find small solutions of a multivariate polynomial equation \( f(x_1, \ldots, x_n) = 0 \mod N \).

**Linearization.** The technique presented in Section 3.3 is able to find small solutions of modular linear equations. We mentioned already that any polynomial equation can be transformed into a linear one by introducing a new variable for each monomial. This process is referred to as **linearization**. But generally, linearization is more powerful than defining a dedicated variable for each monomial. It also allows to glue together certain monomials and then define a new variable for this combined term. To notice the advantage of such a clever linearization consider the polynomial equation \( x^2 + ax + b - y = 0 \mod N \). Suppose that the solutions \( x^{(0)} \) and \( y^{(0)} \) are approximately of the same size and let \( X = N^\delta \) be an upper bound on these solutions, i.e. \( |x^{(0)}| \leq X \) and \( |y^{(0)}| \leq X \). The basic linearization of using a dedicated variable for each monomial gives

\[
\underbrace{x^2}_{u_1} + a \underbrace{x}_{u_2} + b - \underbrace{y}_{u_3} = 0 \mod N,
\]

and the enabling condition of Theorem 6 tells us that we can find a solution as long as \( U_1 U_2 U_3 \leq N \). The upper bounds for \( U_1, U_2, U_3 \) are easily derived as \( N^{2\delta}, N^{\delta}, N^{\delta} \) and, thus, the enabling condition reduces to \( \delta \leq \frac{1}{4} \).

On the other hand, we can perform a clever linearization as

\[
x^2 - y + a \underbrace{x}_{u_2} + b = 0 \mod N, \tag{4.1}
\]

where we linearize the term \( x^2 - y \). The key point is that the size of the new variable \( u'_1 \) is approximately the same as the size of \( u_1 \) and we no longer have a variable \( u'_3 \). In terms of the enabling condition \( U'_1 U'_2 \leq N \) this yields the superior bound \( \delta \leq \frac{1}{3} \). We clearly see that gluing together the monomials \( x^2 \) and \( y \) allows to hide the variable \( y \) in \( x^2 \). This in turn results in a superior bound.

In the most general sense we speak of a linearization if we modify an initial polynomial by introducing at least one dedicated variable for a certain (nonlinear) monomial or a certain term. This means that we do not require the resulting polynomial to be linear. A prominent example is the RSA key equation in the attack of
Unravelled Linearization

Boneh and Durfee which we will study in the Section 6.2. In that case we consider the polynomial equation \( x(N - 1 + y_1 + y_2) + 1 = 0 \mod e \) and linearize the term \( y_1 + y_2 \).

An important requirement for a useful linearization is, however, that the coefficients of the monomials that are subject to the linearization are up to the sign identical or at most differ by a small factor. This is because differing coefficients will also be glued into the new variable and, thus, a large coefficient would dramatically increase the size of the new unknown. Consider for example Equation (4.1) with a coefficient of \( y \) being \( b \) instead of 1.

\[
x^2 - by + a \underbrace{x}_{u_1'} + b = 0 \mod N,
\]

Now an upper bound \( U_1' \) of the new variable \( u_1' \) is given by \( U_1' = \max(N^{2\delta}, N^{\delta+\gamma}) \), where \( b = N^\gamma \). Thus, once \( \gamma > \delta \) the coefficient will increase the size of the unknown \( u_1' \) and in turn decrease the bound for \( \delta \).

Coppersmith. While the linearization approach that we just described makes use of the similarity of the coefficients, the method of Coppersmith exploits the algebraic relations between the variables. In Chapter 3 we explained how Coppersmith’s algorithm works. Recall that a major step in the algorithm is the definition of a collection of so-called shift polynomials. Deriving the final bound as in (3.11), we notice that we aim to find as many shift polynomials as possible with a combined set of monomials as small as possible. Thus, a design goal is to reuse many of the monomials in many shift polynomials.

One can clearly see how Coppersmith’s method exploits the algebraic structure of the initial polynomial in the following example. Consider the univariate polynomial \( f(x) = x^2 + ax + b \) and search for a small root modulo \( N \). A linearization approach reveals by Theorem 6 the enabling condition \( X^2 \cdot X \leq N \), i.e. \( \delta \leq \frac{1}{3} \) for \( X = N^\delta \). Now consider the system of equations defined by

\[
\begin{align*}
  f : & \quad x^2 + ax + b = 0 \mod N \\
  xf : & \quad x^3 + ax^2 + bx = 0 \mod N \\
  f^2 : & \quad x^4 + 2ax^3 + (a^2 + 2b)x^2 + 2abx + b^2 = 0 \mod N^2.
\end{align*}
\]

Theorem 6 can be easily extended to consider such a system of modular equations, and it results in the general enabling condition that the product of the upper bounds of all occurring monomials has to be smaller than the product of all moduli. In this case we obtain \( X^{10} \leq N^4 \) which gives \( \delta \leq \frac{2}{5} \). This improvement of the bound is obtained because we are able to reuse certain monomials in many of the polynomial equations of the system.

It is obvious that the more relations between the individual monomials of \( f \) there are, the more monomials are reappearing in certain multiples and powers of \( f \).

Drawbacks. The linearization process is able to reduce the complexity of the root finding problem significantly by hiding certain monomials, but it thereby completely
removes the algebraic structures involving these monomials. The Coppersmith approach, on the other hand, fully captures any algebraic properties, but it completely ignores the coefficients of the polynomial.

Our goal is to design a method that combines the best of both worlds. I.e. we wish to make use of the similarity of the polynomials’ coefficients to combine certain monomials in order to simplify the problem itself, but on the other hand we wish to fully exploit any algebraic relation between the monomials of the initial polynomial in order to improve the bound by reusing certain monomials. This leads to a new technique called unravelled linearization, which is described in this chapter.

Unravelled Linearization. Let us exemplify the idea on how to combine the two worlds with the help of the polynomial \( f(x, y) = x^2 + ax + b - y \), where the solutions \( x^{(0)} \) and \( y^{(0)} \) are approximately of the same size. Let \( X = N^\delta \) be an upper bound on the solutions, i.e. \( |x^{(0)}| \leq X \) and \( |y^{(0)}| \leq X \). In order to find the small solutions of \( f(x, y) \) modulo \( N \), we apply Coppersmith’s algorithm as described in Section 3.1. Taking \( x^2 \) as leading monomial, we obtain with the strategy from Section 3.2 that we can efficiently find all small solutions for \( \delta \leq \frac{1}{3} \).

On the other hand, if we perform a clever linearization of the polynomial \( f(x, y) \), where we combine the monomials \( x^2 \) and \( y \) in a new variable \( u_1 \), we already saw in the beginning of this chapter that this also allows us to find small roots if \( \delta \leq \frac{1}{3} \).

Notice that the two techniques reveal the same bound on \( \delta \), but they exploit different properties. While Coppersmith’s algorithm makes use of the fact that the polynomial contains related monomials \( x \) and \( x^2 \), the linearization uses the similar coefficients of \( x^2 \) and \( y \).

The key observation that motivates unravelled linearization is that a linearization actually does not completely remove the relations between the monomials of the polynomial, it just hides them. In the current example, there still exists a relation between the variables \( u \) and \( x \) in the linearized polynomial. Recall that we performed a linearization which combines the monomials \( x^2 \) and \( y \) in a new variable \( u \). Because \( x \) and \( y \) are of the same size we know that the upper half of \( u \) is only influenced by \( x^2 \) and, therefore, is already determined by the variable \( x \). Thus, we still have a connection between the (seemingly) unrelated variables \( u \) and \( x \). The new technique of unravelled linearization applies the ideas of Coppersmith’s algorithm to exploit this relation between the variables in the linearized polynomial.

To be more precise, we use multiples and powers of the linear polynomial \( g(u, x) \) and build a lattice as described in Section 3.1. However, the basic method of Coppersmith just exploits apparent relations between the various variables in the polynomial. Since there is no direct relation between the variables \( u \) and \( x \), a basic application of Coppersmith’s method does not help to improve the bound. So the question remains how to make use of the described hidden connection between \( u \) and \( x \)?

It turns out that the key to an improvement is the relation that is implied by the linearization itself. Note that in the current example the linearization introduces the relation \( u = x^2 - y \). We can rearrange the equation to obtain

\[
x^2 = u + y.
\]  

Now when defining the multiples and powers of the linear polynomial \( g(u, x) \) we use
relation (4.2) as a replacement rule, i.e. we replace each occurrence of \(x^2\) by the term \(u + y\). For a polynomial \(h\) denote the polynomial after this replacement by \(\hat{h}\). The advantage of using the relation becomes apparent if we compare the monomials that occur in the set of shift polynomials \(S := \{g, xg, g^2\}\) with the set of unravelled shift polynomials \(\hat{S} := \{g, \hat{x}g, \hat{g}^2\}\). Remember that the enabling condition in Coppersmith's method basically states that the product of all monomials has to be smaller than the product of all moduli in order to efficiently find a small solution. This implies that if the moduli remain the same and we have less or smaller monomials, then in turn the bounds on those monomials can be larger. We first note that for the shift sets \(S\) and \(\hat{S}\) the moduli of the shift polynomials actually are identical, so a difference in the bound is only possible if the set of occurring monomials differ. For the set \(S\) we have monomials \(1, x, u, xu, u^2\) and \(x^2\). This leads to the enabling condition

\[X^4U^4 \leq N^4 \quad \Leftrightarrow \quad \delta \leq \frac{1}{3}.\]

On the other hand, the set \(\hat{S}\) results in monomials \(1, x, u, xu, u^2\) and \(y\), because replacing \(x^2\) by \(u + y\) allows to reuse the monomial \(u\). In this case the enabling condition states

\[X^2YU^4 \leq N^4 \quad \Leftrightarrow \quad \delta \leq \frac{4}{11} \approx 0.364.\]

Figure 4.1 shows the basis matrix for a basic application of Coppersmith’s algorithm written in the original Coppersmith notation (cf. Section 3.4.2 for details). In Figure 4.2 we see the result of unravelling the linearization, i.e. after replacing the monomial \(x^2\) with \(u + y\). We clearly observe that the replacement allows to reuse the variable \(u\) and thus we basically replace the monomial \(x^2\) by \(y\) which gives a benefit in the enabling condition.

\[
\begin{pmatrix}
g & xg & g^2 \\
1 & b & b^2 \\
x & a & 2ab \\
u & 1 & 2b \\
xu & a & 2a \\
x^2 & 1 & a^2 \\
u^2 & 1 & 1 \\
\end{pmatrix}
\]

\[
\begin{pmatrix}
g & \hat{x}g & \hat{g}^2 \\
1 & b & b^2 \\
x & a & 2ab \\
\hat{u} & 1 & 2b \\
\hat{xu} & a & 2a \\
\hat{x}^2 & 1 & a^2 \\
\hat{u}^2 & 1 & 1 \\
\end{pmatrix}
\]

Figure 4.1: Default basis matrix for linear polynomial \(g(u, x)\)  Figure 4.2: Basis matrix after unravelling the linearization

### 4.1 General Strategy

We now give a general strategy for applying the method of unravelling linearization to an arbitrary polynomial equation. Notice that the method works identical for
both modular equations and equations over the integers. The only difference appears in the construction of the lattice where we use to the customized process of Coppersmith’s algorithm in each case. For convenience we describe the strategy for a modular polynomial equation.

Suppose we want to find all small roots of the polynomial equation \( f(x_1, \ldots, x_n) = 0 \mod N \). The algorithm for unraveled linearization proceeds as follows.

1. **Linearization**
   Perform a suitable linearization on \( f(x_1, \ldots, x_n) \) to obtain a linear polynomial \( g(u_1, \ldots, u_r) \).

2. **Relations**
   Derive a relation between the variables \( u_1, \ldots, u_r \).
   If necessary introduce further variables \( u_{r+i} \) which are appropriate expressions in the variables \( x_1, \ldots, x_n \).

3. **Shift Polynomials**
   Select a set of shift polynomials as in the method of Coppersmith. Also include appropriate extrashift in the variables \( u_{r+i} \).

4. **Unravel**
   Use the relations derived in Step 2 in order to obtain the unraveled shift polynomials.

5. **Lattice Construction**
   Build a lattice basis from the unraveled shift polynomials and proceed from this point on with the usual algorithm of Coppersmith.

6. **Root Extraction**
   Compute the desired solution from the polynomials output by Coppersmith’s algorithm.

We now describe the individual steps in more detail.

**Linearization.** In this step we perform a full linearization of the initial polynomial. This means we combine all monomials with an up to the sign identical coefficient into a new variable and for convenience we also introduce a new variables for each remaining monomial. For example, a full linearization is given by

\[
ax_1 x_2 + b \underbrace{\left( x_2 - x_1 x_3 \right)}_{u_2} + c \underbrace{x_1}_{u_3} + x_3 + 1.
\]  

(4.3)

If a certain coefficient is rather small, it can be superior to treat the coefficient as part of the variable and perform a slightly different linearization. For example, if the coefficient \( c \) in (4.3) is so small that \( cx_1 \) is smaller than \( x_3 \), then it may be beneficial to merge \( cx_1 \) with \( x_3 + 1 \) and thereby hide \( cx_1 \) in \( x_3 + 1 \).

\[
a \underbrace{x_1 x_2}_{u_1} + b \left( \underbrace{x_2 - x_1 x_3}_{u_2} \right) + cx_1 + x_3 + 1.
\]  

(4.4)
Deriving the relation. The key to the improvement obtained by unravelled linearization lies in exploiting the relations between the various variables $u_1, \ldots, u_r$. But how can we find such a relation in general? The answer to this question is to consider the system of equations that is defined by the linearizations. In order to find an equation involving only the new variables $u_1, \ldots, u_r$, we aim to eliminate the old variables $x_1, \ldots, x_n$ from this system. For the example in Equation (4.3) we obtain the system

\[
\begin{align*}
    u_1 &= x_1 x_2 \\
    u_2 &= x_2 - x_1 x_3 \\
    u_3 &= x_1 \\
    u_4 &= x_3 + 1
\end{align*}
\]

Eliminating the variables $x_1, x_2, x_3$ from this system we obtain the identity

\[ u_1 = u_3^2 + u_4 - u_3, \]

which only involves the new variables $u_1, \ldots, u_4$.

The situation is slightly different if the system derived from the linearizations does not allow to extract a relation of the new variables. For example, the linearization in Equation (4.4) implies the system

\[
\begin{align*}
    u_1 &= x_1 x_2 \\
    u_2 &= x_2 - x_1 x_3 \\
    u_3 &= cx_1 + x_3 + 1
\end{align*}
\]

which does not allow to eliminate the old variables $x_1, x_2, x_3$. In this case, in order to exploit the relation among $u_1, u_2$ and $u_3$ we have to introduce an additional variable $u_4 = p(x_1, x_2, x_3, x_4)$ for some polynomial $p$ in the variables $x_1, \ldots, x_4$. Note that choosing different polynomials $p$ results in different relations between the $u_i$. It is an open problem how to select $p$ in an optimal way. In the current example we could choose $u_4 = x_2$ and obtain the relation

\[ u_1 u_3 u_4 = u_3^2 + u_1 u_4 - u_2 u_3. \]

Selecting the shift polynomials. The shift polynomials that are to be used in the lattice construction are chosen in such a way that the relations between the variables can be exploited as often as possible. The actual set of shift polynomials for a specific input equation $f(x_1, \ldots, x_n) = 0 \bmod N$ depends on the sizes of the variables and the relations between them, and therefore it is hard to give a general strategy. Nevertheless, the following approach should work for most scenarios. Start by selecting multiples and powers of the polynomial $f$ as proposed by the strategy of Jochemsz and May from Section 3.2. As already mentioned, this alone does not help to improve the bounds on the variables because a linear polynomial does not admit an algebraic structure that could be exploited by Coppersmith’s algorithm. But if we are able to use the identity given by the relation between the variables we can remove some of the monomials and replace them by possibly smaller ones and reuse certain other monomials. This allows an improvement of the final bounds.

Special care has to be taken if we introduced additional variables $u_{r+i}$ in Step 2. In that case, we need to perform certain extrashifts in those variables in order to
4.1 General Strategy

make the replacements possible. For example, due to the additional variable \( u_4 = x_2 \) which is required in order to obtain a relation in the example of Equation (4.4) we need to incorporate this variable in the selection of the shift polynomials – otherwise no replacements would be possible or lesser monomials could be reused. We also have to be careful that the shift polynomials do not admit a linear dependence through the relations between the variables. Thus, it is be necessary to remove certain shift polynomials in order to guarantee a determinant-preserving shift set.

**Unravel the shift polynomials.** Given a collection of shift polynomials we execute the heart of unravelled linearization, namely performing the substitutions induced by the relations derived in Step 2.

**Lattice Construction.** Once we unravelled the collection of shift polynomials, the method of Coppersmith proceeds as usual with the construction of a basis matrix for a lattice. Remember that in Chapter 3 we presented two frameworks for the construction of a lattice. One was due to Howgrave-Graham [HG97] (and Coron [Cor07] for the integer case), where the shift polynomials are designed to share a common root modulo \( N^m \) for some parameter \( m \). Then a lattice basis is constructed from the coefficient vectors of the (weighted) shift polynomials as row vectors. The second framework is the original one of Coppersmith [Cop96b, Cop96a, Cop97], which takes the coefficient vectors of the shift polynomials as column vectors.

The technique of unravelled linearization works without any problems in both frameworks.

**Recovering the root.** It is possible that the polynomial \( g(u_1, \ldots, u_r) \) derived from the linearization step has more variables than the initial polynomial \( f(x_1, \ldots, x_n) \). In that case, a straightforward approach to recover the roots \( u_1(0), \ldots, u_r(0) \) requires Coppersmith’s algorithm to output at least \( r \) polynomials that allow to extract the root — even \( r + k \) if we introduced \( k \) further variables in Step 2 — instead of \( n \) if we do not apply unravelled linearization. Furthermore, it is not even clear if the solutions \( u_1(0), \ldots, u_r(0) \) in the end allow to recover the solutions \( x_1(0), \ldots, x_n(0) \) of the initial problem.

To circumvent these problems, we simply let Coppersmith’s method output \( n \) polynomials, and in order to compute the desired solutions, we undo the linearization. Then we obtain \( n \) polynomials in the original \( n \) variables \( x_1, \ldots, x_n \)

In the remaining chapters of this thesis, we will apply the strategy of unravelled linearization to pseudo random number generators (Chapter 5) and small secret exponent RSA (Chapter 6). We will see that unravelled linearization allows to improve known bounds and simplify existing proofs.
Chapter 5

Pseudo Random Number Generators

Pseudo random number generators (PRGs) play a crucial role in modern cryptography. An especially simple construction is provided by iterating the RSA function

\[ s_i = s_{i-1}^e \mod N \]

for an RSA modulus \( N = pq \) of bit-size \( n \) and a seed \( s_0 \in \mathbb{Z}_N \). This so-called power generator outputs in each iteration a certain amount of bits of \( s_i \), usually the least significant bits. In order to minimize the amount of computation per iteration, one typically uses small \( e \) such as \( e = 3 \). With slight modifications one can choose \( e = 2 \) as well when replacing the iteration function by the so-called absolute Rabin function \([BBS86, BM84]\), where \( s^2 \mod N \) is defined to be \( \min\{s^2 \mod N, N - s^2 \mod N\} \), \( N \) is a Blum integer and \( s_0 \) is chosen from \( \{0, \ldots, N - 1\} \) with Jacobi symbol +1.

It is well-known that under the RSA assumption one can safely output up to \( \Theta(\log n) = \Theta(\log \log N) \) bits per iteration \([BOCS83, DBL84]\). At Asiacrypt 2006, Steinfeld, Pieprzyk and Wang \([SPW06]\) showed that under a stronger assumption regarding the optimality of some well-studied lattice attacks, one can securely output \( \left( \frac{1}{2} - \frac{1}{6e} - \epsilon - o(1) \right)n \) bits. The assumption is based on a specific RSA one-wayness problem, where one is given an RSA ciphertext \( c = m^e \mod N \) together with a certain fraction of the plaintext bits of \( m \), and one has to recover the whole plaintext \( m \). We call this generator the SPW generator. The SPW generator has the desirable property that one can output a constant fraction \( \Omega(\log N) \) of all bits per iteration. Using an even stronger assumption, Steinfeld, Pieprzyk and Wang could improve the output size to \( \left( \frac{1}{2} - \frac{1}{2e} - \epsilon - o(1) \right)n \) bits.

A natural question is whether the amount of output bits of the SPW generator is maximal. Steinfeld et al.’s security proof uses in a black-box fashion the security proof of Fischlin and Schnorr for RSA bits \([FS00]\). This proof unfortunately introduces a factor of \( \frac{1}{2} \) for the output rate of the generator. So, Steinfeld et al. conjecture that one might improve the rate to \( (1 - \frac{1}{e} - \epsilon)n \) using a different proof technique. Here, \( \epsilon \) is a security parameter and has to be chosen such that performing \( 2^n \) operations is infeasible. We show that this bound is essentially the best that one can hope for by giving an attack up to the bound \( (1 - \frac{1}{e})n \).

In previous cryptanalytic approaches, upper bounds for the number of output bits have been studied by Blackburn, Gomez-Perez, Gutierrez and Shpilrainski \([BGPGS06]\). For \( e = 2 \) and a class of PRGs similar to power generators (but
with prime moduli), they showed that provably $\frac{2}{3}n$ bits are sufficient to recover the secret seed $s_0$. Using Coppersmith’s method [Cop97] their result can be generalized for larger values of $e$ to $(1 - \frac{1}{e+1})n$, as mentioned in Steinfeld et al. [SPW06].

In this chapter we study new cryptanalytic attacks on RSA-based generators using the technique of unravelled linearization. In Section 5.2 we will show that the Blum-Blum-Shub generator [BBS86] is insecure if it outputs more than $\frac{2}{3}n$ of the most significant bits of two consecutive iterations. Since pseudo random number generators are usually used to generate a large output stream by performing many iterations it is natural to ask if we are able to improve the result by considering more iterations. In Section 5.3 we use the tool of unravelled linearization in this scenario. We are able to show that the BBS generator with $(1 - \frac{1}{2})n$ output bits is insecure. Finally, in Section 5.5 we extend our result to RSA-based generators and show that outputting $(1 - \frac{1}{2})n$ bits renders the generator insecure. Note that the results of this chapter also apply to the scenario considered in [BGPG06] and for small values of $e$ our results are a significant improvement.

5.1 Blum-Blum-Shub Generator

The Blum-Blum-Shub pseudo random number generator has been introduced in 1986. Although it is quite slow it has the desirable property that its security is based on the well-known quadratic residuosity assumption. In its basic form the BBS generator works as follows

1. Pick two primes $p, q$ of equal bit size, with $p \equiv q \equiv 3 \mod N$. Compute $N = pq$.

2. Choose a random seed $s_0 \in QR_N$, where $QR_N$ is the set of quadratic residues modulo $N$.

3. Apply the iterating function

$$s_{i+1} = s_i^2 \mod N$$

and output the least significant bit of $s_{i+1}$.

The sequence of output bits is given by $LSB(s_1), LSB(s_2), LSB(s_3), \ldots$. Clearly, the efficiency is in this case not very good since we require one squaring modulo $N$ in order to generate one random bit.

As described in the introduction of this chapter, under certain stronger assumptions it is possible to output more than one bit per iteration. In this chapter we look at the BBS generator that outputs a $(1 - \delta)$-fraction of the bits of the internal state $s_i$ in each iteration, and we aim to find the largest value of $\delta$ such that we are still able to recover the complete internal state efficiently. Furthermore, we consider a generator that outputs the most significant bits instead of the least significant ones. It is an open question, whether a similar attack works with given least significant bits.
5.2 Unravelling Pseudo Random Number Generators

In Chapter 4 we introduced a new technique called unravelled linearization for finding small roots of modular polynomial equations. Now, we will apply this technique to obtain a new result in the cryptanalysis of a certain class of pseudo random number generators.

Recall the iterating function of the BBS generator

\[ s_{i+1} = s_i^2 \mod N, \]

and suppose the generator output a \((1 - \delta)\)-fraction of the most significant bits of the internal state in each iteration. I.e. the state \(s_i\) divides in a known part \(k_i\) and an unknown part \(x_i\) of size \(N^\delta\).

\[ s_i = k_i + x_i \]

After two iterations of the generator we can derive the equation

\[ s_2 = s_1^2 \mod N \]

\[ k_2 + x_2 = (k_1 + x_1)^2 \mod N \]

\[ x_1^2 - x_2 + 2k_1 x_1 + k_1^2 - k_2 = 0 \mod N. \] (5.2)

Our goal is to efficiently find the solutions \((x_1^{(0)}, x_2^{(0)})\). The following theorem shows that using the technique of unravelled linearization we are able to find solutions up to a size of \(N^\frac{2}{5}\) using the output of two consecutive iterations.

**Theorem 10.** Under Assumption 1, for every \(\epsilon > 0\) and sufficiently large \(N\), the following holds:

Consider the Blum-Blum-Shub generator with modulus \(N\) that outputs a \((1 - \delta)\)-fraction of the internal state in each iteration. Given the output of two consecutive iterations, the internal states of both iterations can be recovered in time polynomial in \(\log(N)\) if

\[ \delta \leq \frac{2}{5} - \epsilon. \]

**Proof:** We already have seen that two iterations of the generator yield the polynomial equation (5.2). Now we employ the method of unravelled linearization to efficiently find small solutions to this equation. The first step requires to find a suitable linearization.

\[ x_1^2 - x_2 + a x_1 + b = 0 \mod N \]

This leaves us with the linear polynomial \(g(u_1, u_2) = u_1 + au_2 + b\).

The next step in unravelled linearization is to find a relation among the variables \(u_1\) and \(u_2\). To do so, we take the system of equations implied by the
linearizations and aim to construct an equation that does no longer contain the variables $x_1$ and $x_2$.

$$\begin{align*}
  u_1 &= x_1^2 - x_2 \\
  u_2 &= x_1 
\end{align*}$$

Unfortunately, $u_1$ and $u_2$ alone are not sufficient, so we need to introduce a third variable $u_3 = x_2$. The system of these three equations now yields the relation

$$u_2^2 = u_1 - u_3.$$

Now we have a linear polynomial $g(u_1, u_2) = u_1 + au_2 + b$ and a relation $u_2^2 = u_1 - u_3$. We select the shift polynomials according to the strategy of Jochemsz-May from Section 3.2. For this, we fix an integer $m$ depending on $\frac{1}{\epsilon}$ and choose $u_1$ as leading monomial. However, we proceed slightly different than the general strategy and select the shift polynomials to resemble the Newton polytope of $g^{2m}$ instead of $g^m$. A visual representation of $g^{2m}$ and the sets $M_k$ is given in Figure 5.1.

![Figure 5.1: Jochemsz-May strategy to select shift polynomials](image)

Thus we derive

$$u_1^{i_1} u_2^{i_2} \in M_k \iff i_1 = k, \ldots, 2m \quad i_2 = 0, \ldots, 2m - i_1.$$

From the sets $M_k$ we derive a collection of shift polynomials. We employ a lattice construction in the original Coppersmith notation and thus we describe the shift polynomials as presented in Section 3.4.2. I.e. we use shifts $g_{i,k} := u_2^i g^k$ for all monomials $u_2^i$ in $M_k \setminus M_{k+1}$ and $k \geq 1$. This gives the set of shift polynomials

$$g_{i,k} := u_2^i g(u_1, u_2)^k \quad \text{for } k = 1, \ldots, 2m \text{ and } i = 0, \ldots, 2m - k.$$

Following the construction presented in 3.4 we use the coefficient vectors of the polynomials as column vectors. However, we do not use the polynomials directly, but we first compute an unravelled form $\hat{g}_{i,k}$, where we replace each occurrence of the monomial $u_2^2$ by the term $u_1 - u_3$. Recall that a simplified
enabling condition can be derived by comparing the product of the upper bounds of all occurring monomials with the product of all involved moduli. The product of the moduli can simply be derived from the set of shift polynomials. Each shift polynomial \( g_{i,k} \) introduces a modulus \( N^k \). Thus the product of all moduli can be computed as

\[
p_N(m) = \prod_{k=1}^{2m} \prod_{i=0}^{2m-k} N^k = \prod_{k=1}^{2m} N^{(2m+1)k-k^2} = N^{\frac{4}{3}m^3 + o(m^3)}.
\] (5.3)

Computing the product for the upper bounds of all monomials is a little bit more involved. The reason is that the replacing of \( u_2^2 \) introduces new monomials and this becomes easily confusing. Therefore we propose a different strategy here.

Our goal is to write the product of monomials as

\[
U_{p_1}(m)U_{p_2}(m)U_{p_3}(m),
\]

where \( p_1(m), p_2(m) \) and \( p_3(m) \) are polynomial functions in \( m \) with rational coefficients — similar to the expression for the product of the moduli in (5.3). We know that in each of the unravelled shift polynomials any monomial is of maximal degree \( 2m \). Further, the number of shift polynomials that we have in the construction is \( \sum_{k=1}^{2m} \sum_{i=0}^{2m-k} 1 = m^2 + m \). Referring to Remark 5.3.2 in Section 5.3.2 we state that \( p_1(m), p_2(m) \) and \( p_3(m) \) are polynomial functions in \( m \) of degree at most three. Thus, we are able to find each polynomial \( p_i(m) \) using Lagrange interpolation for four data points. I.e. to compute \( p_1(m) \) we explicitly build a lattice basis for some specific values of \( m \) and count in each lattice basis the exact number of all occurring \( u_1 \). This gives us four points of the function \( p_1(m) \) and since it is of degree at most three, these four point uniquely determine \( p_1(m) \).

Remark. Because it is necessary to show that the \( p_i(m) \) are actually polynomial functions in \( m \), the interpolation technique is not suited to prove a certain theorem. Nevertheless, in practice the interpolation technique can very useful to get an idea of the final bound or to check if a selection strategy for the shift polynomials works well. In that case, we increase the number of sample point and check if the resulting \( p_i(m) \) are consistent.

Applying this interpolation technique to the unravelled lattice basis we obtain the functions

\[
\begin{align*}
p_1(m) &= \frac{4}{3} m^3 + o(m^3), \\
p_2(m) &= m^2 + o(m^2), \\
p_3(m) &= \frac{2}{3} m^3 + o(m^3).
\end{align*}
\]

Notice that \( p_2(m) \) is only of order \( m^2 \) while \( p_1(m), p_3(m) \) and \( p_N(m) \) are of order \( m^3 \). This means that asymptotically the influence of the factor \( U_{p_2}(m) \) is negligible. It is intuitively clear that the contribution of the \( u_2 \) is rather small, since each \( u_2 \) with a power larger than one gets replaced.

If we use the values we obtained from the interpolation to derive the
enabling condition we get
\[ U_1 \frac{4}{3} m^3 U_3 \frac{2}{3} m^3 < N \frac{4}{3} m^3 \iff 4 \frac{3}{3} \cdot 2 \delta + 2 \frac{3}{3} \delta < 4 \frac{3}{3} \]
\[ \iff \delta < \frac{2}{5}. \]

**Remark.** Notice that this is exactly the conjecture that was mentioned by Gomez et al. in [GGI05].

### 5.3 Using More Equations

In a practical attack scenario the adversary is probably not restricted to have just the output of two rounds of the generator. Instead he is given a possibly large number of output samples which he could use in his attack. Nevertheless, all known attacks on RSA-based pseudo random number generators cannot make use of such additional information. Recall that we deal with a pseudo random number generator with an iterating function

\[ s_{i+1} = s_i^2 \mod N. \]

The generator outputs in each iteration a \((1 - \delta)\)-fraction of the bits of the internal state \(s_i\). Suppose it outputs the most significant bits, then we can say that we are given approximations \(k_i\) of the internal state \(s_i\). The goal is to recover the complete internal state. In the previous sections we analyzed up which value of \(\delta\) it is possible to recover \(s_i\) in polynomial time given just two approximations \(k_1, k_2\). By setting \(x_i = s_i - k_i\) for \(i = 1, 2\) these two approximations led to a polynomial equation

\[ s_1^2 = s_2 \mod N \]
\[ (x_1 + k_1)^2 = x_2 + k_2 \mod N \]
\[ x_1^2 - x_2 + 2k_1 x_1 + k_1^2 - k_2 = 0 \mod N \]

with small root \((x_1^{(0)}, x_2^{(0)})\).

If we are given a third approximation \(k_3\) and set \(x_3 = s_3 - k_3\) we obtain a second polynomial equation

\[ s_2^2 = s_3 \mod N \]
\[ (x_2 + k_2)^2 = x_3 + k_3 \mod N \]
\[ x_2^2 - x_3 + 2k_2 x_2 + k_2^2 - k_3 = 0 \mod N \]

with small root \((x_2^{(0)}, x_3^{(0)})\).

Similarly we can derive further polynomials given further approximations. The observation that suggests the possibility of an improvement by considering the system of equations is the fact the polynomials have variables in common. Thus,
speaking in the language of Coppersmith, we can reuse variables in the construction of the lattice basis and thus hope for an improvement.

In this section we investigate how we can employ unravelled linearization if we have more than two output samples of the generator which in turn leads to more than one initial polynomial. We will begin by considering three output values. These three values give rise to two polynomials with a common small root. In the first subsection we consider a standard Coppersmith analysis of this system of equations. Although it is generally not known how to optimally handle systems of equations with Coppersmith’s algorithm, in this case we are able to derive useful sets of shift polynomials. We will see that multiple output values do not help to increase the bound in this case. The next subsection handles three output values, i.e. two polynomials with the technique of unravelled linearization and shows that now we actually are able to increase the bound of one polynomials. Finally the unravelled linearization approach will be generalized to an arbitrary number of polynomials.

5.3.1 Unravelled Linearization

In this subsection we will apply the method of unravelled linearization to the system of two equations derived from three output values of the Pollard generator. One can show that a standard analysis using the method of Coppersmith results in the bound \( \delta \leq \frac{3}{7} \). We will see that with the help of unravelled linearization we are able to improve this bound even further.

**Theorem 11.** Under Assumption 1, for every \( \epsilon > 0 \) and sufficiently large \( N \), the following holds:

Consider the Blum-Blum-Shub generator with modulus \( N \) that outputs a \((1 - \delta)\) fraction of the internal state in each iteration. Given the output of three consecutive iterations, the internal states of all three iterations can be recovered in time polynomial in \( \log(N) \) if

\[
\delta \leq \frac{6}{13} - \epsilon.
\]  

(5.6)

**Proof:** As described, the given output values give rise to a system of two polynomial equations with small solutions modulo \( N \). In order to find these small solutions, we apply the process of unravelled linearization. The first step is to define a suitable linearization.

\[
f_1 : \ x_1^2 - x_2 - a_1 \ x_1 + b_1 = 0 \mod N
\]

\[
f_2 : \ x_2^2 - x_3 - a_2 \ x_2 + b_2 = 0 \mod N
\]

From this linearization we obtain two linear polynomials \( g_1(u_1, u_2) \) and \( g_2(u_3, u_4) \).

If we eliminate the variables \( x_1, x_2, x_3 \) from the system of equations implied by the linearizations in order to find relations among the variables, we obtain the relation \( u_2^2 = u_1 + u_4 \). This relation is equivalent to the one that we obtained for a single polynomial after introducing an additional variable. In this case, the additional variable for \( f_1 \) is already given in \( f_2 \). On the
other hand, it seems reasonable to do the same for the polynomial $f_2$, i.e. we additionally introduce a variable $u_5 = x_3$. Then we obtain the two relations

\[
u_2^2 = u_1 + u_4 \\
u_4^2 = u_3 + u_5.
\]

It remains to select the shift polynomials for the lattice construction. We use a construction based on the method of Jochemsz and May that is tweaked for the current scenario. For the asymptotic analysis we use the set of shift polynomials given by

\[
g_{i,j,k} := x^k g_1^i g_2^j, \quad \text{for } \begin{cases} j = 0, \ldots, 2m \\ i = 0, \ldots, 4m - 2j \\ k = 0, \ldots, 4m - 2j - i \end{cases}. \tag{5.7}
\]

Remember from Chapter 3 that the simplified enabling condition is derived by comparing the product of the involved moduli with the product of the upper bounds on all monomials occurring in the lattice basis. Because the replacements make it rather hard to compute all occurring monomials we follow the interpolation approach taken in the previous subsection dealing with unravelled linearization for a single polynomial. I.e. write the product of the bounds on the monomials as

\[
U_1^{p_1} U_2^{p_2} U_3^{p_3} U_4^{p_4} U_5^{p_5}.
\]

Then we take a fixed number of values $m$, explicitly build the lattice basis for each $m$ and determine the exact set of monomials occurring in the lattice basis. From these data points we interpolate the function $p_1(m)$ through $p_5(m)$. Notice that in each shift polynomial every variable occurs with degree at most $m$. Further, by the definition of the shift polynomials in (5.7) we estimate the number of shift polynomials in the lattice construction to be of order $m^3$. Again, referring to Remark 5.3.2 we state that $p_i(m)$ are polynomial functions in $m$ of degree at most four. Thus, it is sufficient to compute 5 data points to uniquely determine each polynomial function $p_i(m)$.

Performing the interpolation we obtain the values

\[
p_1(m) = \frac{16}{3} m^4 + o(m^4) \quad p_2(m) = \frac{8}{3} m^3 + o(m^3) \\
p_3(m) = \frac{8}{3} m^4 + o(m^4) \quad p_4(m) = \frac{8}{3} m^3 + o(m^3) \\
p_5(m) = \frac{4}{3} m^4 + o(m^4) \quad p_N(m) = 8m^4 + o(m^4).
\]

Notice that because of the replacements of $u_2^2$ and $u_4^2$ their asymptotic contribution is negligible. With these values and the upper bounds $U_2 = U_4 = U_5 = N^8$ and $U_1 = U_3 = N^{2\delta}$ the simplified enabling condition asymptotically reduces to

\[
U_1^{16m^4} U_2^{8m^3} U_3^{8m^4} U_4^{8m^3} U_5^{4m^4} < N^{8m^4} \quad \iff \quad \delta < \frac{6}{13}.
\]
Remark. This result shows that unravelled linearization is able to improve the result for two polynomials from \( \frac{3}{7} \approx 0.429 \) using a standard Coppersmith analysis to \( \frac{6}{13} \approx 0.461 \).

### 5.3.2 Arbitrary Number of Iterations

Up to now we have seen in this section that a third output value of the BBS generator gives rise to a second equation with small roots and that a thorough analysis using the technique of unravelled linearization revealed a significant improvement over the result for just one equation. It is straightforward to ask how these results improve if we have a forth, a fifth, etc. output value of the generator.

In this subsection we will derive a general formula for the simplified enabling condition for an arbitrary number of output samples. For this, we require a few auxiliary lemmata.

**Lemma 8.** For constants \( a_1, \ldots, a_n \) the number of non-negative integer solutions of the inequality

\[
a_1 x_1 + \ldots + a_n x_n \leq c
\]

is

\[
\frac{c^n}{n! \prod_{i=1}^{n} a_i} + \mathcal{O}(c^{n-1}).
\]

**Proof:** Beged-Dov showed in [BD72] that lower and upper bounds on the number of solutions \( N_{sol} \) of equation (5.8) are given by

\[
\frac{c^n}{n! \prod_{i=1}^{n} a_i} \leq N_{sol} \leq \frac{(c + a_1 + \ldots + a_n)^n}{n! \prod_{i=1}^{n} a_i} = \frac{c^n + \mathcal{O}(c^{n-1})}{n! \prod_{i=1}^{n} a_i}.
\]

**Lemma 9.** For one value \( \alpha_k \) fixed, the number of solutions to the inequality

\[
\alpha_1 + \beta_1 + \ldots + 2^{n-1} \alpha_n + 2^{n-1} \beta_n + 2^n \alpha_{n+1} \leq 2^m
\]

with \( \beta_1, \ldots, \beta_n \in \{0, 1\} \)

is given by

\[
\frac{2^{nk-1}}{n! \prod_{i=1}^{n} 2^i} \left( 2^m - 2^{k-1} \alpha_k \right)^n + \mathcal{O}((m + \alpha_k)^{n-1}).
\]

**Proof:** We can view the \( \beta_i, i = 1, \ldots, n \) with the appropriate coefficient as a binary representation of a number between 0 and \( 2^n - 1 \). Thus, with the fixed value of \( \alpha_k \), we need to count the number of solutions of

\[
\alpha_1 + \ldots + 2^{k-2} \alpha_{k-1} + 2^k \alpha_{k+1} + \ldots + 2^{n-1} \alpha_n + 2^n \alpha_{n+1} + \beta
\]

\[
\leq 2^m - 2^{k-1} \alpha_k \quad \text{with} \quad \beta \in \{0, \ldots, 2^n - 1\}.
\]

To handle the restriction on \( \beta \) we simply sum over the number of solutions for each value of \( \beta \).

\[
\sum_{\beta=0}^{2^n-1} \# \left\{ \alpha_1 + \ldots + 2^{k-2} \alpha_{k-1} + 2^k \alpha_{k+1} + \ldots + 2^{n-1} \alpha_n + 2^n \alpha_{n+1} \leq 2^m - 2^{k-1} \alpha_k - \beta \right\}
\]
By Lemma 8 we obtain
\[ \sum_{\beta=0}^{2^n-1} \left( \frac{2^n m - 2^{k-1} \alpha_k - \beta}{n! \prod_{i=1, i \neq k-1}^{2^n}} \right)^n + O\left( \left( \frac{2^n m - 2^{k-1} \alpha_k}{n!} \right)^n \right) \]
\[ = \frac{2^n 2^{k-1}}{n! \prod_{i=1}^{2^n} 2^i} \left( 2^n m - 2^{k-1} \alpha_k \right)^n + O\left( \left( m + \alpha_k \right)^n \right) \]
\]

**Theorem 12.** Under Assumption 1, for every \( \epsilon > 0 \) and sufficiently large \( N \), the following holds:
Consider the Blum-Blum-Shub generator with modulus \( N \) that outputs a \( (1 - \delta) \)-fraction of the internal state in each iteration. Given the output of \( n+1 \) consecutive iterations, the internal states of all iterations can be recovered in time polynomial in \( \log(N) \) and time exponential in \( n \), if
\[ \delta \leq \frac{2^{n+1} - 2}{2^{n+2} - 3} - \epsilon. \]

**Proof:** For \( i = 1, \ldots, n+1 \) let \( k_i \) be \( \delta \)-approximations of \( n+1 \) consecutive internal states \( s_i \) of the BBS generator. As described in the beginning of this section, we can derive \( n \) polynomials \( f_1, \ldots, f_n \)
\[ f_1(x_1, x_2) : x_1^2 - x_2 + a_1 x_1 + b_1 = 0 \mod N \]
\[ \vdots \]
\[ f_n(x_n, x_n+1) : x_n^2 - x_{n+1} + a_n x_n + b_n = 0 \mod N \]

The next step of unravelling linearization performs a linearization of the polynomials \( f_i \). For convenience we distinguish between variables \( u_i \) and \( v_i \) defined as \( u_i := x_i^2 - x_{i+1} \) for \( i = 1, \ldots, n \) and \( v_i := x_i \) for \( i = 1, \ldots, n+1 \). We obtain the system of equations
\[ g_1(u_1, v_1) : u_1 + a_1 v_2 + b_1 = 0 \mod N \]
\[ \vdots \]
\[ g_n(u_n, v_n) : u_n + a_n v_n + b_n = 0 \mod N, \]

Together with the relations \( v_i^2 = u_i + v_{i+1} \).

We define a collection of shift polynomials from this system using again a slightly modified version of the Jochemsz-May strategy. We use the polynomials
\[ g_{i_1, \ldots, i_n, k} := v_1^{i_1} g_1^{i_1} \cdots g_n^{i_n}, \]
for \( \{i_1, i_2, \ldots, i_n, k \geq 0 \mid i_1 + 2i_2 + \ldots + 2^{n-1}i_n + k \leq 2^n m \} \)

Now we want to find the enabling condition for this system to be solvable in polynomial time. This reduces, as we have seen before, to finding all monomials in the lattice basis and computing the product of the moduli of
all shift polynomials. Unfortunately, we cannot apply the process of interpolation any more, since we deal with an arbitrary number of polynomials. Thus, in order to identify the occurring monomials we have to go into greater detail.

When investigating the definition of the shift polynomials in (5.9) we notice that the set of monomials of all shift polynomials is not increased by the shift polynomials with index $k \geq 1$. For example, the monomials in $g_{i_1, i_2, \ldots, i_n, k}$ are completely contained in the monomials of $g_{i_1 + k, i_2, \ldots, i_n, 0}$. This follows from the definition of the upper bound of the index $k$. Thus, in order to find all occurring monomials it is sufficient to look at the shifts $g_{i_1, \ldots, i_n, 0}$.

By definition of the shift polynomials in (5.9) and the fact that we deal with linear polynomials $g_i$, we can easily describe the set of monomials occurring prior to the substitutions as

$$\left\{ u_{\alpha_1}^{\alpha_1} v_1^{\beta_1} \cdots u_n^{\alpha_n} v_n^{\beta_n} \left| \begin{array}{c} \alpha_1 + \beta_1 + 2\alpha_2 + 2\beta_2 + \ldots + 2^{n-1}\alpha_n + 2^{n-1}\beta_n \leq 2^n m \\ \beta_1, \ldots, \beta_n \in \{0, 1\} \end{array} \right. \right\} .$$

A crucial observation is that all substitutions $v_i^2 = u_i + v_{i} + 1$ for $i = 1, \ldots, n - 1$ do not introduce any new monomials. Instead, replacing these $v_i^2$ merely restricts the exponents $\beta_i$ to 0 or 1. The last substitution $v_n^2 = u_n + u_{n+1}$, on the other hand, introduces the new variable $u_{n+1}$ and the set of monomials after all the substitutions can be described as

$$\left\{ u_{\alpha_1}^{\alpha_1} v_1^{\beta_1} \cdots u_n^{\alpha_n} v_n^{\beta_n} u_{n+1} \left| \begin{array}{c} \alpha_1 + \beta_1 + \ldots + 2^{n-1}\alpha_n + 2^{n-1}\beta_n + 2^n \alpha_{n+1} + 2^n \beta_{n+1} \leq 2^n m \\ \beta_1, \ldots, \beta_n \in \{0, 1\} \end{array} \right. \right\} .$$

In order to count the weighted sum of the occurring exponents sum over the possible values of $\alpha_k$.

$$p_{uk} = \sum_{\alpha_k = 0}^{2^{n-(k-1)m}} \alpha_k \cdot sol_{k, \alpha_k},$$

where $sol_{k, \alpha_k}$ equals the number of non-negative integer solutions to the inequality

$$\alpha_1 + \beta_1 + \ldots + 2^{n-1}\alpha_n + 2^{n-1}\beta_n + 2^n \alpha_{n+1} + 2^n \beta_{n+1} \leq 2^n m,$$

$\beta_1, \ldots, \beta_n \in \{0, 1\}$

with fixed $\alpha_k$.

An easy calculation with the help of Lemma 9 shows that we obtain for $k = 1, \ldots, n + 1$

$$p_{uk} = \frac{2^{n^2 + 3n - k + 1}}{(n + 2)! \prod_{i=1}^{n} 2^i m^{n+2} + O(m^{n+1})}$$

Modifying Lemma 9 to consider the $\beta_i$, it is easy to see that the expressions for $v_i, i = 1, \ldots, n$ are of highest degree $n + 1$ in $m_i$, i.e. in $O(m^{n+1})$. 

Therefore, we do not explicitly compute them here, since in the final condition they will be negligible.

To compute the product of the moduli of the collection of shift polynomials we proceed similar. Recall that we defined the shift polynomials as

\[ g_{i_1, \ldots, i_n, k} := v_{i_1}^k g_{i_1} \cdots g_{i_n}^k, \]

for \( \{i_1, i_2, \ldots, i_n, k \geq 0 \mid i_1 + 2i_2 + \ldots + 2^{n-1}i_n + 2^{n-1}k \leq 2^nm \} \).

Thus, the product \( p_N \) of the occurring moduli can be computed as a polynomial in \( m \) as follows. Let \( \mathcal{P} \) be the lattice polytope defined by \( i_1, i_2, \ldots, i_n, k \geq 0 \) with \( i_1 + 2i_2 + \ldots + 2^{n-1}i_n + k \leq 2^nm \). Then

\[ p_N = \sum_{(i_1, \ldots, i_n, k) \in \mathcal{P}} i_1 + \ldots + i_n = \sum_{\ell=1}^n \sum_{(i_1, \ldots, i_n, k) \in \mathcal{P}} i_\ell. \]

In order to derive the inner sum, we proceed as in the case of \( p_{u_k} \), i.e. we write

\[ \sum_{(i_1, \ldots, i_n, k) \in \mathcal{P}} i_\ell = \sum_{i_\ell=0}^{2^{n+1-\ell}m} i_\ell \cdot sol_{\ell, i_\ell}, \]

where \( sol_{\ell, i_\ell} \) is the number of solutions of \( i_1 + \ldots + 2^{\ell-2}i_{\ell-1} + 2^{\ell}i_{\ell+1} + \ldots + 2^{n-1}i_n + k \leq 2^nm - 2^{\ell-1}i_\ell \).

Using Lemma 8 we find

\[ \sum_{(i_1, \ldots, i_n, k) \in \mathcal{P}} i_\ell = \frac{2^{(n+1)^2 - \ell}}{(n + 2)! \prod_{i_1=1}^{n-1} 2^i} m^{n+2} + \mathcal{O}(m^{n+1}), \]

and finally

\[ p_N = \frac{2^{(n+1)^2}}{(n + 2)! \prod_{i_1=1}^{n-1} 2^i} (2^n - 1) m^{n+2} + \mathcal{O}(m^{n+1}) \]

With the given values we can compute the simplified enabling condition

\[ U_1^{p_{u_1}} \cdots U_n^{p_{u_n+1}} V_1^{p_{v_1}} \cdots V_n^{p_{v_n}} < N^{p_N}. \]

Remember that we have upper bounds \( U_1 = \ldots = U_n = N^{2\delta} \) and \( V_1 = \ldots = V_n = U_{n+1} = N^{\delta} \). Thus, we obtain

\[ N^{2\delta} \left( \sum_{k=1}^n \frac{2^{n^2+3n-k+1}}{(n+2)! \prod_{i_1=1}^{n-1} 2^i} m^{n+2} + \mathcal{O}(m^{n+1}) \right) \cdot N^{\delta} \left( \frac{2^{(n+1)^2}}{(n + 2)! \prod_{i_1=1}^{n-1} 2^i} m^{n+2} + \mathcal{O}(m^{n+1}) \right) \]

\[ < N^{(n+2)! \prod_{i_1=1}^{n-1} 2^i} \left( 2^n - 1 \right) m^{n+2} + \mathcal{O}(m^{n+1}) \].
5.3 Using More Equations

Let the $O(m^{n+1})$ factors contribute to an error term $\epsilon$ and consider the condition on the exponents

$$2\delta \left( \sum_{k=1}^{n} \frac{2^{n+1+3n-k+1}}{(n+2)! \prod_{i=1}^{n} 2^i} \right) + \delta \left( \frac{2^{n(n+2)}}{(n+2)! \prod_{i=1}^{n} 2^i} \right) < \frac{2^{n+1}}{(n+1)! \prod_{i=1}^{n} 2^i} (2^n - 1)$$

$$\Leftrightarrow \delta < \frac{2^{n+1} - 2}{2^{n+2} - 3}$$

The running time of the algorithm is dominated by the running time of the LLL algorithm. By Theorem 3 an LLL reduction runs in time polynomial in the bit size of the entries in the basis matrix and the dimension of the lattice. The dimension of the lattice that is subject to the LLL reduction equals the number of monomials that appear in all shift polynomials. We simply derive this value as the number of solutions to the inequality given by the description of all monomials.

$$\{ u_1^{\alpha_1} v_1^{\beta_1} \ldots u_n^{\alpha_n} v_n^{\beta_n} u_{n+1}^{\alpha_{n+1}} \mid \alpha_1 + \beta_1 + \ldots + 2^{n-1} \alpha_n + 2^{n-1} \beta_n + 2^n \alpha_{n+1} \leq 2^n m, \beta_1, \ldots, \beta_n \in \{0, 1\} \}.$$ 

With Lemma 8 we compute the dimension as

$$\dim(L) = \frac{2^{1/2} n^2 + \frac{3}{2}}{(n+1)!} m^{n+1} + O(m^n).$$

For the error term we have $\epsilon = O(2^n/m)$ which gives $m = O(2^n)$ and thus the determinant is exponential in $n$.

The entries of the basis matrix are of bit size $O(\log(N^{2^n} m))$, which is again exponential in $n$. 

**Corollary 1.** Under Assumption 1, for every $\epsilon > 0$ and sufficiently large $N$, the following holds:

Given $n \to \infty$ many output samples of the Blum-Blum-Shub generator with modulus $N$ that outputs a $(1 - \delta)$-fraction of the internal state in each iteration. We can recover the internal states of all iterations if

$$\delta < \frac{1}{2} - \epsilon$$

in time polynomial in $\log N$ and exponential in $n$.

**Remark.** In the previous analysis we were only concerned with the coefficients of the highest powers of $m$ as we consider an asymptotic bound. This allowed a simple presentation. However, in order to justify the interpolation technique used in the previous sections, we need to show that each $p_i$ is actually a polynomial in $m$. We remark here, that an exact computation of the values in the previous section shows this.
5.4 Experiments

Since the results that have been presented in this chapter use the multivariate extension of Coppersmith’s method \[3.1\] we have to rely on the assumption that the polynomials that are output by Coppersmith’s algorithm allow to extract the small roots efficiently (Assumption \[1\]). In order to verify this assumption, we ran a number of experiments. We implemented the unravelled linearization using SAGE 3.4.1. including the $L^2$ reduction algorithm from Nguyen and Stehlé [NS05]. In every experiment we were able to extract the small solution from the output of Coppersmith’s algorithm. Table 5.1 shows some experimental results for the Blum-Blum-Shub generator and 256 bit modulus $N$. In the first column we denote the number of polynomials. The second column shows the chosen parameter $m$, which has a direct influence on how close we approach the asymptotic bound. On the other hand, the parameter $m$ increases the lattice dimension and therefore the time required to compute a solution. The theoretically expected $\delta$ is given in the third column, whereas the experimentally verified $\delta$ is given in the fourth column. The last column denotes the time required to find the solution on a Core2 Duo 2.2 GHz running Linux 2.6.24.

<table>
<thead>
<tr>
<th>polys</th>
<th>m</th>
<th>$\delta$</th>
<th>exp. $\delta$</th>
<th>dim($L$)</th>
<th>time(s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>4</td>
<td>0.377</td>
<td>0.364</td>
<td>15</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>6</td>
<td>0.383</td>
<td>0.377</td>
<td>28</td>
<td>5</td>
</tr>
<tr>
<td>1</td>
<td>8</td>
<td>0.387</td>
<td>0.379</td>
<td>45</td>
<td>45</td>
</tr>
<tr>
<td>2</td>
<td>4</td>
<td>0.405</td>
<td>0.390</td>
<td>22</td>
<td>10</td>
</tr>
<tr>
<td>2</td>
<td>6</td>
<td>0.418</td>
<td>0.408</td>
<td>50</td>
<td>1250</td>
</tr>
<tr>
<td>3</td>
<td>4</td>
<td>0.407</td>
<td>0.400</td>
<td>23</td>
<td>5</td>
</tr>
</tbody>
</table>

Table 5.1: Experimental results for the Blum-Blum-Shub generator

Notice that the experimental data for three polynomials is not much better than that for only two. The reason for this is that the full power of our unravelled linearization only kicks in for three polynomials with a value of $m = 8$, where a maximum of variables can be reused.

5.5 RSA-based Generator

Our attack works similar if we instantiate the PRNG with the RSA-function $x \mapsto x^e \mod N$ instead of the Rabin function $x \mapsto x^2 \mod N$. An RSA-based generator works as follows.

1. Pick two primes $p, q$ of equal bit size, compute $N = pq$ and select $e \in \mathbb{Z}_{\phi(N)}^*$.
2. Choose a random seed $s_0 \in \mathbb{Z}_N$.
3. Apply the iterating function
   \[ s_{i+1} = s_i^e \mod N \]
   and output the least significant bit of $s_{i+1}$.
Similar to the Blum-Blum-Shub generator, the output rate in the is very small if we only output one bit per iteration. Therefore, suppose we output a constant fraction \((1-\delta)\) of the internal state bits per iteration. Furthermore, suppose the generator outputs most significant bits instead of least significant bits. It is an open problem to extend our attack to least significant bits.

Given two consecutive output values \(k_1, k_2\) we can write the internal state as 
\[
s_i = k_i + x_i \quad \text{for} \quad i = 1, 2,
\]
where \(x_i\) denotes the unknown part of the corresponding state. Using the iteration function, we can derive the polynomial equation.

\[
(k_1 + x_1)^e = k_2 + x_2 \mod N
\]

\[
x_1^e - x_2 + ek_1 x_1^{e-1} + \ldots + ek_1^{e-1} x_1 + k_1^e - k_2 = 0 \mod N.
\]

Furthermore, given \(n + 1\) output values, we derive in the same way a system of \(n\) polynomial equations.

\[
f_1(x_1, x_2) : \quad x_1^e - x_2 + ek_1 x_1^{e-1} \ldots ek_1^{e-1} x_1 + k_1^e - k_2 = 0 \mod N
\]

\[
\vdots
\]

\[
f_n(x_n, x_{n+1}) : \quad x_n^e - x_{n+1} + ek_n x_n^{e-1} \ldots ek_n^{e-1} x_n + k_n^e - k_{n+1} = 0 \mod N
\]

Before we prove the main theorem of this section, we require a generalization of Lemma 9 to values \(e > 2\).

**Lemma 10.** For one value \(\alpha_k\) fixed, the number of non-negative integer solutions to the inequality

\[
(e - 1)\alpha_1 + \beta_1 + e(e - 1)\alpha_2 + e\beta_2 + \ldots + (e - 1)e^{n-1}\alpha_n + e^{n-1}\beta_n + e^n\alpha_{n+1} \leq e^n(e - 1)m \quad \text{with} \quad \beta_1, \ldots, \beta_n \in \{0, \ldots, e - 1\}
\]

is given by

\[
(e - 1) \frac{e^n e^{k-1}}{n!} \prod_{i=1}^{n} e^i \left(e^n (e-1)m - e^{k-1} \alpha_k\right)^n + O \left((m + \alpha_k)^{n-1}\right).
\]

**Proof:** The proof of this lemma is very similar to the proof of Lemma 9. We interpret the possible combinations of the \(\beta_i, i = 1, \ldots, n\) as representation of a number \(\bar{\beta}\) between 0 and \(e^n - 1\) in base \(e\). Summing over all values \(\bar{\beta}\) and computing in each case the number of solutions of the remaining inequality by Lemma 8 gives the desired result.

**Theorem 13.** Under Assumption 1, for every \(\epsilon > 0\) and sufficiently large \(N\), the following holds:

Consider the RSA-based pseudo random number generator with modulus \(N\) and exponent \(e\) that outputs a \((1-\delta)\)-fraction of the internal state in each iteration. Given the output of \(n + 1\) consecutive iterations, the internal states of all iterations can be recovered in time polynomial in \(\log(N)\) and time exponential in \(n\), if

\[
\delta \leq \frac{e^{n+1} - e}{e^{n+2} - 2e + 1} - \epsilon.
\]
Proof: We apply the method of unravelled linearization to the system of equations defined by the output values of the RSA generator. By using the linearization $u_i := v_i^e - v_{i+1}$ we obtain the system

$$g_1(u_1, v_1) : u_1 + ek_1v_1^{e-1} + \ldots + ek_1^{e-1}v_1 + b_1 = 0 \mod N$$

$$\vdots$$

$$g_n(u_n, v_n) : u_n + ek_nv_n^{e-1} + \ldots + ek_n^{e-1}v_n + b_n = 0 \mod N,$$

together with the relations $v_i^e = u_i + v_{i+1}$.

We use shift polynomials

$$g_{i_1,\ldots,i_n, k} := u_1^{i_1}g_1^{i_1} \ldots g_n^{i_n},$$

for

$$\left\{ i_1, i_2, \ldots, i_n, k \geq 0 \right\} \left\{ (e-1)i_1 + e(e-1)i_2 + \ldots + e^{n-1}(e-1)i_n + k \leq e^n(e-1)m \right\}.$$

The next step of unravelled linearization is to apply Coppersmith’s algorithm for the given collection of shift polynomials. Recall that the enabling condition for being able to find a small solution is that the product of the upper bounds for all monomials has to be smaller than the product of all moduli. In order to compute the required values, we first need to describe the monomials that appear in the shift polynomials after performing the unravelling step.

We start by investigating the monomials that appear in the shift polynomial prior to the substitutions. Note that, similar to the case of the BBS generator, we can restrict ourselves to the case $k = 0$, since larger values of $k$ do not bring any new monomials.

The monomials that appear are given by

$$\left\{ u_1^{\alpha_1}v_1^{\beta_1} \ldots u_n^{\alpha_n}v_n^{\beta_n} \left| (e-1)\alpha_1 + \beta_1 + e(e-1)\alpha_2 + e\beta_2 + \ldots + e^{n-1}(e-1)\alpha_n + e^{n-1}\beta_n \leq e^n(e-1)m \right. \right\}.$$

Again, the crucial observation is that all substitutions $v_i^e = u_i + v_{i+1}$ for $i = 1, \ldots, n - 1$ do not introduce any new monomials. Instead, replacing these $v_i^e$ merely restricts the exponents $\beta_i$ to values in $\{0, \ldots, e - 1\}$. The last substitution $v_n^e = u_n + u_{n+1}$, on the other hand, introduces the new variable $u_{n+1}$ and the set of monomials after all the substitutions can be described as

$$\left\{ u_1^{\alpha_1}v_1^{\beta_1} \ldots u_n^{\alpha_n}v_n^{\beta_n}u_{n+1}^{\alpha_{n+1}} \left| (e-1)\alpha_1 + \beta_1 + e(e-1)\alpha_2 + e\beta_2 + \ldots \right. + e^{n-1}\beta_n + e^n\alpha_{n+1} \leq e^n(e-1)m \right. \right\}.$$

$$\beta_1, \ldots, \beta_n \in \{0, \ldots, e - 1\}.$$

We aim to write the enabling condition as

$$U_1^{p_1} \ldots U_{n+1}^{p_{n+1}} V_1^{p_1} \ldots V_n^{p_n} < N^{p_N}$$

for appropriate polynomial functions $p_k$, $p_{n+1}$, and $p_N$ depending on the parameter $m$. 

The polynomials $p_{u_k}$ are derived in the same way as in the proof of Theorem 12. i.e. we write

$$p_{u_k} = \sum_{\alpha_k = 0}^{e^n-(k-1)m} \alpha_k \cdot \text{sol}_{k,\alpha_k},$$

where $\text{sol}_{k,\alpha_k}$ equals the number of non-negative integer solutions to the inequality

$$(e - 1)\alpha_1 + e(e - 1)\alpha_2 + e\beta_2 \ldots + e^{n-1}\beta_n + e^n\alpha_{n+1} \leq e^n(e - 1)m, \quad \beta_1, \ldots, \beta_n \in \{0, \ldots, e - 1\}$$

with fixed $\alpha_k$.

An easy calculation with the help of Lemma 10 shows that we obtain for $k = 1, \ldots, n$

$$p_{u_k} = (e - 1)\frac{e^{n^2+3n-k+1}}{(n + 2)!} \prod_{i=1}^n e_i^{m+i+2} + \mathcal{O}(m^{n+1}).$$

For the last value $p_{u_{n+1}}$ we need to be a little bit more careful, because in contrast to the other $\alpha_k$, the variable $\alpha_{n+1}$ has no coefficient $(e - 1)$ in (5.10). Nevertheless, we easily compute with Lemma 8

$$p_{u_{n+1}} = (e - 1)^2\frac{e^{n^2+2n}}{(n + 2)!} \prod_{i=1}^n e_i^{m+i+2} + \mathcal{O}(m^{n+1}).$$

As in the proof of Theorem 12 we argue that the exponent polynomials $p_{u_k}$ are of degree at most $n + 1$ in $m$, i.e. $\mathcal{O}(m^{n+1})$. We easily see this, if we modify Lemma 10 to estimate the number of solutions for one value $\beta_k$ fixed, and compute the sums analogous to $p_{u_k}$. While the number of solutions computed via an appropriate lemma gives $\mathcal{O}((m + \beta_k)^{n+1})$, the sum over $\beta_k$ with a fixed range does not increase this number asymptotically.

In order to compute the enabling condition it remains to derive the value $p_N$, which is the product of all moduli of the shift polynomials. Recall that the shift polynomials are given by

$$g_{i_1, \ldots, i_n, k} := e_1^{k}g_1^{i_1} \ldots g_n^{i_n}, \quad \text{for} \quad \begin{cases} i_1, i_2, \ldots, i_n, k \geq 0 \mid (e - 1)i_1 + e(e - 1)i_2 + \ldots + e^{n-1}(e - 1)i_n + k \leq e^n(e - 1)m \end{cases}.$$  

Let $P$ be the lattice polytope defined by $i_1, i_2, \ldots, i_n, k \geq 0$ with $(e - 1)i_1 + e(e - 1)i_2 + \ldots + e^{n-1}(e - 1)i_n + k \leq e^n(e - 1)m$. Then

$$p_N = \sum_{(i_1, \ldots, i_n, k) \in P} i_1 + \ldots + i_n = \sum_{\ell=1}^n \sum_{(i_1, \ldots, i_n, k) \in P} i_\ell.$$  

Proceeding as in the proof of Theorem 12 we compute using Lemma 10

$$p_N = \frac{e^{(n+1)^2}}{(n + 2)!} \prod_{i=1}^n e_i^{e^n(e - 1)m+i+2} + \mathcal{O}(m^{n+1}).$$
With the given values we can compute the simplified enabling condition

\[ U_1^{P_{a_1}} \cdot \ldots \cdot U_{n+1}^{P_{a_{n+1}}} V_1^{P_{b_1}} \cdot \ldots \cdot V_n^{P_{b_n}} < N^{P_N}. \]

Remember that we have upper bounds \( U_1 = \ldots = U_n = N^{\delta} \) and \( V_1 = \ldots = V_n = U_{n+1} = N^{\delta} \). Thus, we obtain

\[
N^{e \delta} \left( \sum_{k=1}^{n} e_k^{n^2+3n-k+1} \epsilon_k \right) + N^{e \delta} (e-1)^2 \left( \frac{e^{n(n+2)}}{(n+2)! \prod_{i=1}^{n} \epsilon^i} \right) \leq N^{e \delta} \left( e^{n+1} - \epsilon \right) \]

\[ \leq e^{(n+1)^2} \prod_{i=1}^{n} \epsilon^i (e^n - 1) \leq e^{e \delta} \prod_{i=1}^{n} \epsilon^i (e^n - 1) \leq e^{n+1} - e < e^{2n+2} - e + 1 \]

\[ \Leftrightarrow \delta < \frac{n+1-e}{2n+2 - 2e + 1} \]

The running time of the algorithm is dominated by the time it takes to LLL-reduce a certain lattice basis. We followed the basis construction using the original Coppersmith notation 3.4.2, and therefore the dimension of the lattice that is subject to the LLL reduction is given by the number of monomials appearing in all shift polynomials. We simply derive this value as the number of solutions to the inequality given by the description of all monomials.

\[
\left\{ u_1^{\alpha_1} v_1^{\beta_1} \ldots u_n^{\alpha_n} v_n^{\beta_n} u_{n+1}^{\alpha_{n+1}} \right| (e-1) \alpha_1 + \beta_1 + e(e-1) \alpha_2 + e \beta_2 + \ldots + e^{n-1} \beta_n + e^n \alpha_n + 1 \leq e^n (e-1) m \quad \beta_1, \ldots, \beta_n \in \{0, \ldots, e-1\} \right\}
\]

With Lemma 8 we compute the dimension as

\[ \dim(L) = e^{\frac{n^2+3}{2}} (e-1) m^{n+1} + O(m^n). \]

For the error term we have \( \epsilon = O\left( \frac{n^m}{m} \right) \) which gives \( m = O\left( \frac{n^m}{\epsilon} \right) \) and thus the determinant is exponential in \( n \) and polynomial in \( \epsilon \).

The entries of the basis matrix are of bit size \( O\left( \log(N^{\epsilon^m}) \right) \), which is again exponential in \( n \) and polynomial in \( \epsilon \).

\[ \text{Remark.} \] For a polynomial time algorithm we require the number \( \epsilon \) to be of size \( O(\log N) \). However, in a practical implementation of an RSA-based PRNG the value of \( \epsilon \) is probably chosen rather small for reasons of efficiency.
Corollary 2. Under Assumption 1, for every $\epsilon > 0$ and sufficiently large $N$, the following holds:

Given $n \to \infty$ many output samples of the RSA-based generator with modulus $N$ and exponent $e$ that outputs a $(1 - \delta)$-fraction of the internal state in each iteration. We can recover the internal states of all iterations if

$$\delta < \frac{1}{e} - \epsilon$$

in time polynomial in $\log N$ and $e$, and time exponential in $n$. 

Chapter 6

Attacks on RSA

In this chapter we present a number of improvements to well known attacks on RSA. Since RSA is one of the most widely used public key cryptosystem to date, its cryptanalysis is a very important task, and even a slight improvement of a specific attack could render currently employed parameters insecure. The reason why practical parameters are often chosen at the edge of a cryptanalytic attack, i.e. only slightly larger than some bound for which the scheme is insecure, is mostly for reasons of efficiency. A prominent example is RSA with small private keys. The most basic approach to improve the efficiency of RSA decryption is to use a small secret parameter $d$. However, according to a result of Wiener [Wie90] and later Boneh and Durfee [BD00], the value $d$ may not chosen to be too small. So, in practice the value $d$ is chosen larger than this attack bound, but yet as small as possible. Therefore, a small improvement of the bound on $d$ for which RSA turns insecure or even just a more efficient attack technique, could question the security of practical systems.

In the first section of this chapter we will recall the basic RSA cryptosystem as well as a variant called RSA-CRT [QC82]. The next section is dedicated to an attack on basic RSA with a small secret exponent $d$. We present the famous attack from Boneh and Durfee [BD00], which uses the algorithm of Coppersmith to find a small solution to a certain modular equation. In its basic form, their result states that values of $d$ that are smaller than $N^{0.284}$ can be recovered from the public parameters in polynomial time. A more sophisticated analysis allows this bound to be increased to $N^{0.292}$, but the required techniques are rather unnatural and complicated. In Subsection 6.2.2 we apply the technique of unravelled linearization to the attack of Boneh and Durfee. We will see that this tool easily explains the best known bound $N^{0.292}$ and we will present a simple and natural proof of the attack from [BD00].

The second half of this chapter is devoted to the analysis of RSA-CRT, to be more precise, to the analysis of RSA-CRT with small secret exponents. We begin by presenting the basic underlying idea of a cryptanalysis, which led to the currently best known attack on RSA-CRT by Jochemsz and May [JM07]. However, in [JM07] there is an open question concerning an improvement of the result. Motivated by the observation that the practical experiments are better than the theoretical expectation, Jochemsz and May conjecture that it might be possible to improve their result $d_p, d_q \leq N^{0.073}$ by a sophisticated analysis as in [BD00]. In Section 6.3.2 we answer their open question by using the technique of unravelled linearization. We will present an attack which perfectly predicts the experimental results. Unfortunately,
it does not help to improve the bound $N^{0.073}$, but it is still of practical interest since it is uses a smaller lattice dimension than the attack in [JM07].

### 6.1 The RSA Cryptosystem

RSA, named after its inventors Rivest, Shamir and Adleman, is one of the oldest public key cryptosystems. There exist other cryptosystems with better security properties and efficiency but nevertheless RSA is still widely used in practice. Mainly this is because RSA is very simple to describe and implement.

**Key generation**

On input a security parameter $n$, select two primes of equal bit length $\frac{n}{2}$ and compute $N = pq$. The public key is the pair $pk = (N,e)$, where $e$ is an integer coprime to $\varphi(N)$, i.e. $gcd((p-1)(q-1), e) = 1$. To derive the corresponding private key we compute the integer $d \equiv e^{-1} \mod \varphi(N)$ and set $sk = (N,d)$.

**Encryption**

On input a public key $(N,e)$ and a message $m \in \mathbb{Z}_N$, the encryption function computes the ciphertext

$$c = m^e \mod N.$$

**Decryption**

Given a ciphertext $c \in \mathbb{Z}_N$ and the private key $(N,d)$, the message is recovered by computing

$$c^d = (m^e)^d = m^{ed} \equiv m \mod N.$$

Notice that the running time for performing a decryption is given by the time required to compute the modular exponentiation $c^d \mod N$. For an arbitrary value $c \in \mathbb{Z}_N$ this running time is $O(\log d \log^2 N)$. To make the decryption process faster in practice, it is possible to choose a small value $d$. However, as we will see in Section 6.2.1, taking $d \leq N^{0.292}$ is insecure.

Another option for speeding up the decryption has been mentioned by Quisquater and Courr\^e [QCS2]. The idea is to perform the decryption in the isomorphic ring $\mathbb{Z}_p \times \mathbb{Z}_q$ instead of $\mathbb{Z}_N$. This is possible because the party holding the private key also knows the prime factors of $N$. In this so-called RSA-CRT variant the key generation and decryption are modified as follows.

**Key generation**

On input a security parameter $n$, select two primes of equal bit length $\frac{n}{2}$ and compute $N = pq$. The public key is the pair $pk = (N,e)$, where $e$ is an integer coprime to $\varphi(N)$, i.e. $gcd((p-1)(q-1), e) = 1$. To derive the corresponding private key we compute the integer $d \equiv e^{-1} \mod \varphi(N)$. Compute values $d_p = d \mod p - 1$ and $d_q = q \mod q - 1$, and set $sk = (N,p,q,d_p,d_q)$. 


6.2 Attacking RSA with Small Secret Exponent $d$

**Encryption**

On input a public key $(N,e)$ and a message $m \in \mathbb{Z}_N$, the encryption function computes the ciphertext

$$c = m^e \mod N.$$ 

**Decryption**

Given a ciphertext $c \in \mathbb{Z}_N$ and the private key $(N, p, q, d_p, d_q)$, compute

$$m_p = c^{d_p} \mod p,$$

$$m_q = c^{d_q} \mod q,$$

and combine $c_p, c_q$ using the Chinese Remainder Theorem to obtain the value $m = c^d \mod N$.

The running time of the decryption in RSA-CRT is dominated by the exponentiations, and the time required to combine the partial results $m_p, m_q$ using the CRT can be neglected. Since compared to the decryption in standard RSA, we are working with numbers of half the bit size. This results in much faster exponentiations, and theoretically a speed-up of 4 is possible.

To even further speed-up the decryption one might additionally choose small values of $d_p$ and $d_q$. The best known attack in this scenario has been given by Jochemsz and May [JM07] in 2007.

### 6.2.1 The Boneh-Durfee Attack

We shortly recall the famous attack of Boneh and Durfee on small secret exponent RSA. First, we prove the basic result that once $d \leq N^{0.284}$ we can efficiently recover $d$ from $N$ and $e$. Second, we state the improved result $d \leq N^{0.292}$ but give only an intuition of the proof.

**Theorem 14** (Boneh-Durfee). Under Assumption 1, for every $\epsilon > 0$ and sufficiently large $N$ the following holds: Let $N = pq$ be an RSA modulus and $e$ a public exponent. Let further $d$ be the private exponent corresponding to $e$ and $\delta = \log_N(d)$. Given $(N, e)$, if

$$\delta < \frac{1}{6} \left( 7 - 2\sqrt{7} \right) - \epsilon$$

then the value $d$ can be recovered in time polynomial in $\log N$.

**Proof:** Consider the RSA key equation with the linearization $g := p + q$.

$$ed = 1 + \frac{k}{x} \left( N + 1 - (p + q) \right)$$

From Equation (6.1) Boneh and Durfee derive the polynomial

$$f_e(x, y) := 1 + x(A + y),$$
which has the small root \((x^{(0)}, y^{(0)}) = (k, -(p+q))\) modulo \(e\). The size of the solution can be upper bounded by \(|x^{(0)}| \leq N^\delta =: X\) and \(|y^{(0)}| \leq 3N^{\frac{1}{2}} =: Y\).

In order to find this small root of \(f_e(x, y)\) the method of Coppersmith is used. For the selection of the shift polynomials we apply the method of Jochemsz and May from Section 3.2. Thus, fix as leading monomial \(\lambda = xy\), and define sets

\[
M_k := \bigcup_{0 \leq \ell \leq t} \left\{ x^i y^j \bigg| \begin{array}{c} \text{\(x^i y^j\) is a monomial of \(f_e^m\)} \\
\text{and \(x^i y^j\) is a monomial of \(f_{e^{-k}}^m\)}
\end{array} \right\}.
\] (6.2)

Notice that we include extrashifts in the variable \(y\).

![Figure 6.1: The sets \(M_k\) for the basic Boneh-Durfee attack](image)

From Figure 6.1 we can easily read of a description of the sets \(M_k\) for all \(k\).

\[x^i y^j \in M_k \iff i = k, \ldots, m \quad j = k, \ldots, i + t\]

Remember that given the definition of \(M_k\) and the upper bounds \(X = N^\delta\) and \(Y = N^{\frac{1}{2}}\), the Jochemsz-May strategy allows to derive a condition for \(\delta\) on when we are able to efficiently find the small solution. In this case we obtain

\[\delta < \frac{1}{6} \left(7 - 2\sqrt{7}\right) \approx 0.284.\]

The major contribution from Boneh and Durfee was the observation that the result \(\delta < 0.284\) is indeed not optimal. They were able to prove the following theorem with a superior bound on \(\delta\) by removing some of the polynomials from the collection taken in the previous analysis.

**Theorem 15** (Boneh-Durfee (improved)). Under Assumption 1, for every \(\epsilon > 0\) and sufficiently large \(N\) the following holds: Let \(N = pq\) be an RSA modulus and
6.2 Attacking RSA with Small Secret Exponent $d$

A public exponent. Let further $d$ be the private exponent corresponding to $e$ and \( \delta = \log_N(d) \). Given \((N,e)\), if
\[
\delta < \frac{2 - \sqrt{2}}{2} - \epsilon
\]
then the value $d$ can be recovered in time polynomial in $\log N$.

We only give an idea of the proof in [BD00] and refer to the original paper for the complete proof.

Let us take a closer look at the lattice basis that is constructed in the proof of Theorem 14. Remember that the shift polynomials used in the lattice construction are defined in the method of Jochemsz and May as follows.

\[
g_{i,j} := \frac{x^i y^j}{k} f_e(x, y)^k e^{m-k}, \quad \text{for } x^i y^j \in M_k \setminus M_{k+1} \text{ and } k = 0, \ldots, m.
\]

For example, we construct the sets $M_k$ as described in (6.2) for values $m = 2, t = 1$ and obtain

\[
\begin{align*}
M_0 &= \{1, x, x^2, y, xy, x^2 y, x^2 y^2, x^2 y^3\} \\
M_1 &= \{xy, x^2 y, xy^2, x^2 y^2, x^2 y^3\} \\
M_2 &= \{x^2 y^2, x^2 y^3\}.
\end{align*}
\]

This results in the shift polynomials

\[
\begin{align*}
g_{0,0} &= e^2 \\
g_{1,0} &= xe^2 \\
g_{2,0} &= x^2 e^2 \\
g_{1,1} &= f_e e \\
g_{2,1} &= xf_e e \\
g_{1,2} &= yf_e e \\
g_{2,2} &= f_e^2 \\
g_{3,2} &= xf_e^2.
\end{align*}
\]

Following the strategy of Jochemsz and May we use the coefficient vectors of the polynomials $g_{i,j}(xX, yY)$ as row vectors in a lattice basis $L_{BD}$.

\[
\begin{pmatrix}
e^2 \\
x e^2 \\
f_e e \\
x f_e e \\
y e^2 \\
y f_e e
\end{pmatrix}
\begin{pmatrix}
e^2 X \\
eAX \\
eX \\
eAX^2 \\
eAXY \\
eAXY
\end{pmatrix}
\begin{pmatrix}
1 \\
2AX \\
2AXY \\
A^2X^2 \\
2AX^2Y \\
X^2Y^2
\end{pmatrix}
\begin{pmatrix}
e^2Y \\
eY \\
eXY^2 \\
eY \\
eXY^2 \\
eY
\end{pmatrix}
\begin{pmatrix}
x \\
y \\
x^2 \\
x^2y \\
x^2y^2
\end{pmatrix}
\begin{pmatrix}
x \\
y \\
x^2y \\
x^2y^2
\end{pmatrix}
\begin{pmatrix}
x^2y \\
x^2y^2 \\
x^2y^3
\end{pmatrix}
\]

Figure 6.2: Basis matrix of $L_{BD}$ for parameters $m = 2$ and $t = 1$

An interesting general observation in Coppersmith’s algorithm is the following. Recall the simplified enabling condition that we defined in Equation (3.10),
\[
\det (L_{BD}) < (e^2)^{\omega-\epsilon},
\]
where $\omega$ is the dimension of the lattice $L$. Now, suppose we add a certain polynomial to the initial collection of polynomials such that this polynomial introduces exactly one new monomial. What happens with the simplified enabling condition is that the determinant is increased by a factor corresponding to the new monomial and the dimension is increased by one. This has the effect on the bounds of the unknowns that we increase the bound if the new contribution is smaller than $c^2$ and similarly decrease the bound if the contribution is larger than $c^2$. Thus, we can say:

Every polynomial that introduces a factor to the determinant that is smaller than $c^2$ improves the bounds on the unknowns — all others do not.

If we keep this rule of thumb in mind and take a closer look at the lattice basis given in Figure 6.2, we notice that the extrashifts $yf_e e$ and $ye^2$ actually are in this sense not helpful because each one introduces a factor to the determinant that is larger than $c^2$.

On the other hand, a crucial point in the strategy of Jochemsz-May is that the basis matrix is triangular. Because this allows an easy derivation of the determinant as the product of the diagonal entries. Note that we need to derive the determinant of a basis matrix which is parameterized by values $m$ and $t$. So, if we simply remove the non-helpful extrashifts $yf_e e$ and $ye^2$, we obtain a non-triangular lattice basis, which seems to be an obstacle.

Boneh and Durfee, however, showed that for the particular polynomial $f_e(x, y) = 1 + Ax + xy$ it is actually possible to evaluate the determinant as a function of $m$ and $t$ even if we remove a certain set of extrashifts and thereby obtain a non-triangular lattice basis.

For the previous example we obtain the lattice basis for a sublattice $L_S$ of $L_{BD}$ given in Figure 6.3

$$
\begin{pmatrix}
e^2 & 1 & x & xy & x^2 & x^2y & x^2y^2 & y & xy^2 & x^2y^3 \\
e^2x^2 & e & eAX & eXY & e^2X^2 & eAX^2 & eX^2Y & 1 & 2AX & 2XY & A^2X^2 & 2AX^2Y & X^2Y^2 \\
f_e e & eX & eAX^2 & eX^2Y & 2AXY & AX^2Y & 2AX^2Y^2 & Y & 2XY^2 & X^2Y^3 & 1
\end{pmatrix}
$$

Figure 6.3: Improved but non-triangular lattice basis for parameters $m = 2$ and $t = 1$

The analysis of the determinant of this non-triangular lattice basis is however quite complicated and at this point we refer the reader to the original proof in [BD00]. The final result of the analysis from Boneh and Durfee yields

$$
\delta < \frac{1}{2} \left( 2 - \sqrt{2} \right) \approx 0.292.
$$
A different method to obtain this result has been presented by Blömer and May in [BM05]. They basically use the fact that the determinant of the non-triangular lattice $L_S$ is given by

$$\det (L_S) := \prod_{i=0}^{6} \|b_i^*\|,$$

where $b_i$ are the basis vectors and $b_i^*$ their Gram-Schmidt orthogonalization. The crucial observation is that for the last basis vector $b_6$ the value of $\|b_6^*\|$ is mainly influenced by the term $X^2Y^3$ and the contribution of $Y$ and $2XY^2$ is negligible. They proceed by constructing a different lattice $L'_S$ in which the columns corresponding to the monomials $y$ and $xy^2$ are simply removed. On the one hand, this new lattice $L'_S$ is triangular and has basically the same determinant as the lattice $L_S$, and on the other hand, Blömer and May showed that a small vector $v'$ in $L'_S$ corresponds to a small vector $v$ in $L_S$ and that $v$ can be reconstructed from $v'$. Thus, they perform Coppersmith’s algorithm on the lattice $L'_S$ and transform the solution back into the lattice $L_S$. The details of this method from Blömer and May, however, are again quite complicated and unnatural. In the next subsection we will see a nice and simple analysis of the sublattice $L_S$ using the technique of unravelled linearization.

### 6.2.2 Unravelling the Boneh-Durfee attack

We will now apply the method of unravelled linearization in the scenario of the Boneh-Durfee attack on small secret exponent RSA. It will allow a much simpler and very natural analysis of the improved bound $\delta < 0.292$ and it shows that unravelled linearization is very useful in optimizing Coppersmith-type lattice based attacks.

For convenience, we state the theorem of Boneh and Durfee here again.

**Theorem 16 (Boneh-Durfee (improved)).** Under Assumption 1, for every $\epsilon > 0$ and sufficiently large $N$ the following holds: Let $N = pq$ be an RSA modulus and $e$ a public exponent. Let further $d$ be the private exponent corresponding to $e$ and $\delta = \log_N(d)$. Given $(N, e)$, if

$$\delta < \frac{2 - \sqrt{2}}{2} - \epsilon$$

then the value $d$ can be recovered in time polynomial in $\log N$.

**Proof (Using unravelled linearization):** Start with the initial polynomial

$$f_e(x, y) := 1 + Ax + xy$$

which has a small root modulo $e$. We perform the following linearization

$$\bar{f}_e(u_1, u_2) := 1 + xy + A \underbrace{x}_{u_1} \underbrace{u_2}_{u_2}$$

This leaves us with a bivariate linear polynomial $\bar{f}_e$ in the variables $u_1$ and $u_2$, and the system of equations induced by the linearizations

$$u_1 = 1 + xy$$
$$u_2 = x.$$
In order to get a relation between $u_1$ and $u_2$ we need to introduce a third variable $u_3$ that corresponds to $y$ in the original polynomial. Then we obtain the identity

$$u_2u_3 = u_1 - 1.$$ (6.3)

Next we build a lattice basis for the polynomial $f_e(u_1, u_2)$ in Coppersmith fashion, where we perform the selection of the shift polynomials according to the strategy of Jochemsz and May from Section 3.2 with leading monomial $\lambda = u_1$.

$$g_{i,k} := u_1^i f_e^k(u_1, u_2) e^{m-k}, \quad \text{for} \quad k = 0, \ldots, m \quad i = 0, \ldots, m - k$$

To be able to exploit Relation (6.3) we need to additionally perform extrashifts in the variable $u_3$. We use the same set of extrashifts as in [BD00].

$$\bar{h}_{j,k} = u_3^j f_e^k(u_1, u_2) e^{m-k} \text{ for } j = 1, \ldots, t \text{ and } k = \left\lfloor \frac{m}{t} \right\rfloor j, \ldots, m.$$

In the next step we build a lattice basis from the coefficient vectors of the unravelled shift polynomials. I.e. we use the identity given in (6.3) to replace the monomial $u_2u_3$ by $u_1 - 1$. The key point of this proof is that the basis matrix formed by these coefficient vectors is triangular and therefore the required computation of the determinant is easy. To show that the basis matrix is indeed triangular, we look at an arbitrary shift $u_3^j f_e^k$. Notice that for the ease of notation we will omit the factor $e^{m-\ell}$ as it does not influence the set of monomials. Since $f_e = u_1 + Au_2$ we can expand $u_3^j f_e^k$ by the binomial theorem

$$u_1^j u_3^k + \binom{\ell}{1} A u_1^{\ell-1} u_2 u_3^j + \ldots + \binom{\ell}{t} A^t u_2 u_3^k.$$

The first term introduces a new monomial $u_1^j u_3^k$. However, we will now derive a certain restriction under which all other monomials are already present in the lattice basis. Therefore, let us look at the monomials of the second term after the substitution of $u_2u_3$

$$u_1^{\ell-1} u_2 u_3^j = u_1^{\ell-1} (u_1 - 1) u_3^{i-1} = u_1^i u_3^{i-1} - u_1^{\ell-1} u_3^{i-1}.$$  

The monomials $u_1^i u_3^{i-1}$ and $u_1^{\ell-1} u_3^{i-1}$ appear in $u_3^{i-1} f_e^k$ and $u_3^{i-1} \bar{f}_e^k$, respectively. In general, the $(j+1)^{th}$ term of the binomial expansion contains monomials that appear in $u_3^{i-j} \bar{f}_e^{\ell-k}$ for $k = 0, \ldots, j$.

Thus, the shift $u_3^j \bar{f}_e^k$ introduces exactly one new monomial $u_1^i u_3^k$ if all shifts $u_3^{i-j} \bar{f}_e^{\ell-k}$ for $j = 1, \ldots, i - 1$ and $k = 0, \ldots j$ were used in the construction of the lattice basis.

It remains to show that the chosen set of $u_3$-shifts $\bar{h}_{j,k}$ satisfies this requirement, i.e. we show that if $u_3^i \bar{f}_e^k$ is a $u_3$-shift, then all of $u_3^{i-j} \bar{f}_e^{\ell-k}$ for
6.2 Attacking RSA with Small Secret Exponent $d$

$j = 1, \ldots, i - 1$ and $k = 0, \ldots, j$ are also used as shifts. Notice that it is sufficient to show $u_{t}^{i-j} f_{t}^{\ell-j}$ is used as a shift.

Since $u_{t}^{i-j} f_{t}$ is in the set of $u_{t}$-shifts, we know that $\ell \in \{ \lceil m/3 \rceil i, \ldots, m \}$ and therefore $\ell - j \in \{ \lceil m/3 \rceil i - j, \ldots, m - j \}$. For $u_{t}^{i-j} f_{t}^{\ell-j}$ on the other hand, we have $\ell - j \in \{ \lceil m/3 \rceil (i - j), \ldots, m \}$. Our requirement is thus fulfilled if the condition

$$\left\lfloor \frac{m}{t} \right\rfloor (i - j) \leq \left\lfloor \frac{m}{t} \right\rfloor i - j$$

holds. We can rewrite this as $\left\lfloor \frac{m}{t} \right\rfloor \geq 1$, which holds if $m \geq t$.

To obtain the final bound $\delta$, which is the size of the unknown variable $x$, we proceed with the usual analysis for modular polynomials. I.e. we compute the determinant of the lattice and require the condition

$$\det (L) < e^{\dim L \cdot m}.$$  \hfill (6.4)

Since the basis matrix is by construction triangular, we can easily compute the determinant as the product of the diagonal entries. Note that each shift polynomial $u_{t}^{i}$ introduces a diagonal term $U_{t}^{i} U_{t}^{3} e^{m-k}$ and each extrashift $h_{t}$ contributes a diagonal term $U_{t}^{3} U_{t}^{3} e^{m-k}$. Thus, setting $t = \tau m$, we compute the determinant of the lattice as $U_{1}^{s_{1}} U_{2}^{s_{2}} U_{3}^{s_{3}} e^{s_{e}}$ for values

$$s_{1} = \sum_{k=0}^{m} \sum_{i=0}^{m-k} k + \sum_{i=1}^{\tau m} \sum_{k=0}^{m} k = \left( \frac{1}{6} + \tau \right) m^{3} + o(m^{3})$$

$$s_{2} = \sum_{k=0}^{m} \sum_{i=0}^{m-k} i = \frac{1}{6} m^{3} + o(m^{3})$$

$$s_{3} = \sum_{i=1}^{\tau m} \sum_{k=0}^{m} i = \frac{\tau^{2}}{6} m^{3} + o(m^{3})$$

$$s_{e} = \sum_{k=0}^{m} \sum_{i=0}^{m-k} (m - k) + \sum_{i=1}^{\tau m} \sum_{k=0}^{m} (m - k) = \left( \frac{1}{3} + \frac{\tau}{6} \right) m^{3} + o(m^{3}).$$

In order to evaluate the enabling condition, we further need to compute the dimension of the lattice, which in this case is equal to the number of shift polynomials plus extrashifts.

$$\dim L = \sum_{k=0}^{m} \sum_{i=0}^{m-k} 1 + \sum_{i=1}^{\tau m} \sum_{k=0}^{m} 1 = \left( \frac{1}{2} + \frac{\tau}{2} \right) m^{2} + o(m^{2})$$

Overlay, Condition [6.4] reduces to

$$(e^{\frac{1}{2} + \frac{\tau}{2}} m^{3} + o(m^{3})) U_{1}^{1} m^{3} + o(m^{3}) U_{2}^{2} m^{3} + o(m^{3}) U_{3}^{3} m^{3} + o(m^{3}) e^{(\frac{1}{2} + \frac{\tau}{2}) m^{3} + o(m^{3})} \leq e^{(\frac{1}{3} + \frac{\tau}{6}) m^{3} + o(m^{3})}.$$}

Substitute the upper bounds $U_{1} = e^{\frac{1}{2} + \delta}, U_{2} = X = e^{\delta}, U_{3} = Y = e^{\frac{\delta}{2}}$. Further let $m$ grow to infinity an let all terms of order $o(m^{3})$ contribute to some error
term $\epsilon$, we obtain the asymptotic condition
\[
\delta < \frac{1 + 2\tau - \tau^2}{4 + 4\tau} - \epsilon
\]

Maximizing this bound, we derive the optimal value $\tau = 1 - 2\delta$ and finally get
\[
\delta < \frac{2 - \sqrt{2}}{2} - \epsilon.
\]

This is exactly the bound obtained in the original attack due to Boneh and Durfee. To summarize, the method of unravelled linearization allows a simple and natural analysis of the sublattice attack of Boneh and Durfee. In previous proofs of the best bound $\delta < 0.292$ complicated techniques were required to show that a certain sublattice has a particular determinant. Using unravelled linearization we are able to directly construct the required sublattice in a triangular lattice basis and therefore the determinant computation is very easy.

It is still an open question if the bound 0.292 can be further improved. Boneh and Durfee conjecture in their original work that it might be possible to raise the bound to $\delta < \frac{1}{2}$. There might be possibilities to obtain a superior bound if one considers a different polynomial, e.g. by introducing individual variables for the values $p$ and $q$, and additionally using the fact that $N = pq$, instead of linearizing the sum $p + q$ in a single variable, which is unrelated to $N$.

### 6.3 Attack on RSA-CRT

In Section 6.1 we presented a variant of the RSA cryptosystem, which improves the efficiency of the decryption process. The fact that the prime factors $p$ and $q$ are known to the decrypting party allows to use the Chinese Remainder Theorem to compute a decryption modulo $p$ and a decryption modulo $q$, and finally combine them to the desired decryption modulo $N$.

Similar to standard RSA, we have for the CRT exponents $d_p$ and $d_q$ used in the partial decryptions a system of key equations
\[
ed_p = 1 \mod p - 1
\]
\[
ed_q = 1 \mod q - 1.
\]

As mentioned previously, a possibility to further increase the efficiency is to use small CRT exponents. Notice that small $d_p$ and $d_q$ does not imply a small decryption exponent $d$. Thus, the attack of Boneh-Durfee does not work in this case.

The currently best attacks on small CRT exponents derive a single equation from the system of key equations together with the fact that $pq = N$. Write
\[
ed_p - 1 + k = kp
\]
\[
ed_q - 1 + \ell = \ell q
\]
and multiply those two equations to get
\[(ed_p - 1 + k)(ed_q - 1 + \ell) = k\ell N\]
\[e^2d_p d_q - ed_p - ed_q + ed_p \ell + ed_q k + 1 - \ell - k - k\ell(N - 1) = 0 \tag{6.5}\]

with unknowns \(d_p, d_q, k, \ell\).

In this section we will revisit the result on small CRT exponents by Jochemsz and May [JM07]. Their attack works by applying the algorithm of Coppersmith to find a small solution of Equation (6.5). Next, we will employ the concept of unravelled linearization to the polynomial used by Jochemsz and May. Although unravelled linearization will not be able to increase the asymptotic result over the one of [JM07], our new attack works with much smaller lattice bases and therefore allows to attack larger values \(d_p, d_q\) in practice.

### 6.3.1 Jochemsz-May Attack

The attack of Jochemsz and May from Crypto 2007 was the first polynomial time attack on small CRT exponents that worked for full size public exponents \(e\). Their result states that RSA with small CRT exponents and \(e\) of the size of \(N\) is insecure if \(d_p\) and \(d_q\) are chosen to be smaller than \(N^\delta\) for \(\delta < 0.0734 - \epsilon\). The proof of this result basically uses the strategy of Jochemsz and May from Section 3.4.1 to select appropriate shift polynomials for the analysis of Equation (6.5). For the full proof we refer the reader to the original publication [JM07].

It is an important part of their analysis to perform experiments, since an application of Coppersmith’s algorithm in the multivariate scenario is only heuristic. Jochemsz and May performed many experiments to justify the assumption that the root can be efficiently extracted from the polynomials that are output by Coppersmith’s algorithm (Assumption [1]). They observed the strange behavior that the experiments yielded better results than theoretically predicted by the analysis. For example, taking parameters \(m = 2\) and \(t = 1\), the enabling condition does not allow to find small roots at all, while in practice it was possible to find solutions for \(d_p, d_q < N^{0.01}\).

Motivated by this observation, Jochemsz and May conjecture that there might be a hidden sublattice structure similar to the one in the Boneh-Durfee analysis. The hope is that finding and analyzing this hidden sublattice might improve the bound \(d_p, d_q < 0.0734\). However, up to now, there have been no new ideas which allowed to improve the bound from Jochemsz-May.

### 6.3.2 Unravelling Jochemsz-May attack

In this subsection we will apply the technique of unravelled linearization to the attack of Jochemsz and May on small CRT exponents. It turns out that we are able to fully explain the gap between the theoretical and practical results obtained by Jochemsz and May. Unfortunately, the asymptotic result of our new analysis does not improve the bound 0.0734. Thus, we answer the open question posed by Jochemsz and May about increasing the final bound by considering a sublattice, in the negative. Nevertheless, the lattices we consider in our method are of much smaller dimension than the ones required in the original attack in [JM07]. This way,
our attack is more effective in practice, i.e. with the same amount of computing power we are able to attack larger values $d_p, d_q$ than Jochemsz and May.

\textbf{Theorem 17.} Under Assumption 1, for every $\epsilon > 0$ and sufficiently large $N$, the following holds: Let $N = pq$ be an RSA modulus and $e$ be a public exponent. Further, let $d_p, d_q < N^\delta$ be private CRT exponents satisfying $ed_p = 1 \mod p - 1$ and $ed_q = 1 \mod q - 1$. Given $(N, e)$, if 

$$\delta < 0.0734 - \epsilon,$$

then the values $d_p$ and $d_q$ can be recovered in time polynomial in $\log N$.

\textbf{Proof (Using unravelled linearization):} We start with the same polynomial equation as Jochemsz-May (6.5), i.e. Equation (6.5). Instead of identifying each unknown value as a single variable, we introduce a linearization.

$$e^2 \frac{d_p d_q}{u_1} - e \left( d_p + d_q \right) \frac{u_2}{u_3} + e \left( d_p \ell + d_q k \right) \frac{u_4}{u_5} + \ell + k - (N - 1) \frac{k \ell}{u_5} - 1 = 0$$

This leaves us with a linear polynomial

$$\bar{f} := e^2 u_1 - eu_2 + eu_3 + u_4 - \left( N - 1 \right) u_5 - 1$$

with a root

$$\left( u_1^{(0)}, u_2^{(0)}, u_3^{(0)}, u_4^{(0)}, u_5^{(0)} \right) = \left( d_p d_q, d_p + d_q, d_p \ell + d_q k, \ell + k, k \ell \right).$$

Further, we have the system of equations introduced by the linearizations.

$$\begin{align*}
  u_1 &= d_p d_q \\
  u_2 &= d_p + d_q \\
  u_3 &= d_p \ell + d_q k \\
  u_4 &= \ell + k \\
  u_5 &= k \ell
\end{align*}$$

Eliminating $d_p, d_q, k, \ell$ we obtain the identity

$$u_2 u_3 u_4 = u_1 u_2^2 + u_2^2 u_5 + u_3^2 - 4u_1 u_5 \quad (6.6)$$

We select a collection of shift polynomials using the extended strategy of Jochemsz and May. I.e. we construct a set of shift monomials $S$ and use for each $s \in S$ the polynomial $s \cdot \bar{f}$ as shift polynomial. The set $S$ is constructed to contain all monomials of $f^{m-1}$ as well as extrashifts in $u_1$ and $u_2$. However, because of Relation (6.6) we have to be more careful with the selection of the shift polynomials. It is necessary to remove all multiples of $u_2 u_3 u_4$ from the set $S$ since otherwise we get a linear dependency between the shift polynomials.
\[ \tilde{S} = \bigcup_{t_1=0}^{t} \bigcup_{t_2=0}^{t-t_1} \{ u_1^{e_1+t_1} u_2^{e_2+t_2} u_3^{e_3} u_4^{e_4} u_5^{e_5} \mid u_1^{e_1} u_2^{e_2} u_3^{e_3} u_4^{e_4} u_5^{e_5} \text{ is monomial of } \tilde{m}^{-1} \} \] and \( u_2 u_3 u_4 \{ u_1^{e_1+t_1} u_2^{e_2+t_2} u_3^{e_3} u_4^{e_4} u_5^{e_5} \}. \]

In the next step we construct a lattice basis from the shift polynomials as described in Section 3.4.2. Notice that we replace each occurrence of \( u_2 u_3 u_4 \) by the expression given in Relation (6.6). It remains to derive the enabling condition \( \det(\mathcal{L}) > 1 \). Therefore, write the determinant as \( \det(\mathcal{L}) = U_1^{−s u_1} U_2^{−s u_2} U_3^{−s u_3} U_4^{−s u_4} A^{s A} \) and compute the exponent values. We can characterize the monomials in \( \tilde{S} \) as

\[ \tilde{S} = \tilde{S}_1 - \tilde{S}_2, \]

where

\[ u_1^{e_1} u_2^{e_2} u_3^{e_3} u_4^{e_4} u_5^{e_5} \in \tilde{S}_1 \iff \begin{cases} e_5 = 0,\ldots,m-1 \\ e_4 = 0,\ldots,m-1-e_5 \\ e_3 = 0,\ldots,m-1-e_5-e_4 \\ e_2 = 0,\ldots,m-1-e_5-e_4-e_3+t \\ e_1 = 0,\ldots,m-1-e_5-e_4-e_3-e_2+t \end{cases} \]

and the multiples of \( u_2 u_3 u_4 \) that we need to remove are given by

\[ u_1^{e_1} u_2^{e_2} u_3^{e_3} u_4^{e_4} u_5^{e_5} \in \tilde{S}_2 \iff \begin{cases} e_5 = 0,\ldots,m-1 \\ e_4 = 1,\ldots,m-1-e_5 \\ e_3 = 1,\ldots,m-1-e_5-e_4 \\ e_2 = 1,\ldots,m-1-e_5-e_4-e_3+t \\ e_1 = 0,\ldots,m-1-e_5-e_4-e_3-e_2+t \end{cases} \]

By setting \( t = \tau m \) we are able to derive the number of used shift polynomials, which directly reveals the exponent \( s_A \) in the determinant expression:

\[ s_A = |\tilde{S}| = |\tilde{S}_1 \setminus \tilde{S}_2| = \left( \frac{1}{8} + \frac{\tau}{2} + \frac{\tau^2}{2} \right) m^4 + o(m^4). \]

To compute the remaining exponents in the determinant expression we proceed similar. We need to describe the monomials that appear in the lattice basis after the substitution of \( u_2 u_3 u_4 \). It is easy to see that, because of the given set of shift polynomials, the substitutions do not introduce any new monomials. Also, to identify the monomials in the lattice basis, we can simply consider the set of shift polynomials given by \( S \) instead of \( \tilde{S} \), where \( S \) contains all monomials of \( \tilde{m}^{-1} \). In that case, the shift polynomials which are multiples of \( u_2 u_3 u_4 \) are removed on the monomial-level by removing all monomials that are multiples of \( u_2 u_3 u_4 \). Thus, we can describe the set of
occurring monomials after the substitutions via Relation 6.6 as $\bar{M} = M_1 - M_2$
with
\[
\begin{align*}
    u_1^{e_1} u_2^{e_2} u_3^{e_3} u_4^{e_4} u_5^{e_5} &\in \bar{M}_1 \iff \\
    &\begin{cases}
    e_5 = 0, \ldots, m \\
    e_4 = 0, \ldots, m - e_5 \\
    e_3 = 0, \ldots, m - e_5 - e_4 \\
    e_2 = 0, \ldots, m - e_5 - e_4 - e_3 + t \\
    e_1 = 0, \ldots, m - e_5 - e_4 - e_3 - e_2 + t
    \end{cases}
\end{align*}
\]
and
\[
\begin{align*}
    u_1^{e_1} u_2^{e_2} u_3^{e_3} u_4^{e_4} u_5^{e_5} &\in \bar{M}_2 \iff \\
    &\begin{cases}
    e_5 = 0, \ldots, m \\
    e_4 = 1, \ldots, m - e_5 \\
    e_3 = 1, \ldots, m - e_5 - e_4 \\
    e_2 = 1, \ldots, m - e_5 - e_4 - e_3 + t \\
    e_1 = 0, \ldots, m - e_5 - e_4 - e_3 - e_2 + t
    \end{cases}
\end{align*}
\]

Given the set $\bar{M}$ we are able to easily compute the exponents $s_{u_1}, s_{u_2}, s_{u_3}$ and $s_{u_4}$ as
\[
\begin{align*}
    s_{u_1} &= \sum_{e_4=0}^{m} \sum_{e_3=0}^{m-e_4} \sum_{e_2=0}^{m-e_4-e_3+t} \sum_{e_1=0}^{m-e_4-e_3-e_2+t} e_1 \\
    &= \left(\frac{1}{8} + \frac{\tau}{2} + \frac{3\tau^2}{4} + \frac{\tau^3}{3}\right) m^4 + o(m^4) \\
    s_{u_2} &= \left(\frac{1}{12} + \frac{\tau}{3} + \frac{\tau^2}{2} + \frac{\tau^3}{3}\right) m^4 + o(m^4) \\
    s_{u_3} &= \left(\frac{1}{12} + \frac{\tau}{3} + \frac{\tau^2}{4}\right) m^4 + o(m^4) \\
    s_{u_4} &= \left(\frac{1}{12} + \frac{\tau}{3} + \frac{\tau^2}{4}\right) m^4 + o(m^4).
\end{align*}
\]

Using these values in the expression for the determinant and recalling the upper bounds $(U_1, U_2, U_3, U_4) = (N^{2\delta}, N^{\delta}, N^{\frac{1}{2}+2\delta}, N^{\frac{1}{2}+\delta})$, we can compute the asymptotic enabling condition
\[
2\delta \left(\frac{1}{8} + \frac{\tau}{2} + \frac{3\tau^2}{4} + \frac{\tau^3}{3}\right) + \delta \left(\frac{1}{12} + \frac{\tau}{3} + \frac{\tau^2}{2} + \frac{\tau^3}{3}\right) \\
+ \left(\frac{1}{2} + 2\delta\right) \left(\frac{1}{12} + \frac{\tau}{3} + \frac{\tau^2}{4}\right) + \left(\frac{1}{2} + \delta\right) \left(\frac{1}{12} + \frac{\tau}{3} + \frac{\tau^2}{4}\right) \\
> \left(\frac{1}{8} + \frac{\tau}{2} + \frac{\tau^2}{2}\right),
\]
With the optimal value $\tau \approx 0.381788$ this condition reduces to
\[
\delta < 0.0734.
\]
This is exactly the same asymptotic bound that was derived by Jochemsz and May in [JM07]. So, unfortunately, we cannot improve their result despite of identifying the conjectured sublattice that is responsible for the gap between the theoretical and experimental results in [JM07].

Nevertheless, our new attack has a notable advantage. Namely, the dimension of the involved lattice is significantly smaller. This is of practical relevance because the actual running time of the attack is dominated by the time to LLL reduce the lattice basis. As we see in Theorem 3, the running time of the lattice reduction is mainly influenced by the dimension of the input lattice. Thus, our attack is in practice faster than the attack of Jochemsz and May. To show this advantage, Figure 6.4 compares the two approaches. We clearly see, that for reasonable lattice dimensions the actual value of $\delta$ that can be attacked is significantly larger for our method.

**Example:** To see the benefit of unravelled linearization in the RSA-CRT scenario, consider the lattice basis that is constructed using parameters $(m, t) = (2, 1)$. This is the smallest set of parameters, where the experimental results of [JM07] are better than theoretically predicted.

Figure 6.5 shows the lattice basis $B$, where we use the compact notation for the Coppersmith method over the integers as introduced in Section 3.4. Recall that for Coppersmith’s method we construct a lattice basis with the coefficient vectors of the shift polynomials as column vectors. For simplicity we omit in the compact notation the left hand side of the basis matrix, which contains the inverses of the corresponding upper bounds of the monomials on its diagonal. Notice that the shift vectors $u_2u_3 \cdot \bar{f}$ and $u_2u_4 \cdot \bar{f}$ contain the monomial $u_2u_3u_4$ that is subject to the substitution by (6.6). Thus, when constructing the lattice basis we replace the occurrence of $u_2u_3u_4$ in these polynomials. The entries in the basis matrix that came from these substitutions are printed in bold letters.

Let us compute the enabling condition for the specific lattice $L$ corresponding to the basis matrix given in Figure 6.5. Recall that we have in case of integer
Figure 6.5: Matrix of unravelled linearized polynomial for $m = 2, t = 1$

Equations the simplified enabling condition $\det(\mathcal{L}) > 1$. For the determinant of $\mathcal{L}$ we obtain

$$\det(\mathcal{L}) = U_1^{-21} U_2^{-20} U_3^{-14} U_4^{-14} A^{15}.$$ 

Remember that we are looking for the small root

$$\left( u_1^{(0)}, u_2^{(0)}, u_3^{(0)}, u_4^{(0)}, u_5^{(0)} \right) = (d_p d_q, d_p + d_q, d_p \ell + d_q k, \ell + k, k \ell)$$
with the upper bounds

\[(U_1, U_2, U_3, U_4) = (N^{2\delta}, N^\delta, N^{1+2\delta}, N^{1+\delta})\].

Further, the value \(A = N - 1\) that we introduced can be approximated by \(N\) and we get that the enabling condition \(\det(L) > 1\) reduces to \(\delta < \frac{1}{101} \approx 0.01\). This perfectly matches the experimental results of Jochemsz and May for parameters \((m, t) = (2, 1)\).

**Experiments**  The reason for carrying out various experiments for attacking CRT-RSA is twofold. First, we want to show that our analysis from Subsection 6.3.2 is indeed optimal. That is, the experimental behavior can be perfectly predicted by the analysis and there is no hope to improve the bound by this approach. Second, as our lattice-based approach is heuristic, we have to verify that Assumption 1 is fulfilled. I.e. the polynomials that we obtain after the lattice reduction allow for an efficient recovery of the common roots.

We reimplemented the attack of [JM07] and used in the experiments the same modulus sizes and lattice parameters as done in [JM07]. Table 6.1 clearly shows the speedup for the LLL reduction. For example with parameters \(m = 3\) and \(t = 1\) our method is 20 to 30 times faster than the one of Jochemsz and May. As previously mentioned, this is due to the reduced lattice dimension\(^1\). While Jochemsz and May required the reduction of a lattice of dimension 115, our lattice only has dimension 60. Because of this smaller lattice dimension we were able to perform experiments on parameter sets that have been out of reach before.

\(^1\)The lattice we are considering here is the one that serves as input to the LLL reduction routine. That is the sublattice containing zeros in the coordinates corresponding to the shift polynomials.

<table>
<thead>
<tr>
<th>(N)</th>
<th>(d_p, d_q)</th>
<th>(\delta)</th>
<th>((m, t, \text{dim}))</th>
<th>(\text{dim}^{JM})</th>
<th>(T_{L, LLL}^{JM})</th>
<th>(T_{L, LLL})</th>
</tr>
</thead>
<tbody>
<tr>
<td>1000 bit</td>
<td>11 bit</td>
<td>0.0096</td>
<td>(2, 1, 30)</td>
<td>56</td>
<td>14</td>
<td>2</td>
</tr>
<tr>
<td>1000 bit</td>
<td>18 bit</td>
<td>0.0178</td>
<td>(3, 1, 60)</td>
<td>115</td>
<td>6100</td>
<td>258</td>
</tr>
<tr>
<td>1000 bit</td>
<td>22 bit</td>
<td>0.0226</td>
<td>(3, 2, 93)</td>
<td>–</td>
<td>–</td>
<td>3393</td>
</tr>
<tr>
<td>1000 bit</td>
<td>24 bit</td>
<td>0.0244</td>
<td>(4, 1, 105)</td>
<td>–</td>
<td>–</td>
<td>7572</td>
</tr>
<tr>
<td>1000 bit</td>
<td>29 bit</td>
<td>0.0291</td>
<td>(4, 2, 154)</td>
<td>–</td>
<td>–</td>
<td>61298</td>
</tr>
<tr>
<td>2000 bit</td>
<td>21 bit</td>
<td>0.0096</td>
<td>(2, 1, 30)</td>
<td>56</td>
<td>40</td>
<td>4</td>
</tr>
<tr>
<td>2000 bit</td>
<td>35 bit</td>
<td>0.0178</td>
<td>(3, 1, 60)</td>
<td>115</td>
<td>20700</td>
<td>613</td>
</tr>
<tr>
<td>2000 bit</td>
<td>45 bit</td>
<td>0.0226</td>
<td>(3, 2, 93)</td>
<td>–</td>
<td>–</td>
<td>13516</td>
</tr>
<tr>
<td>2000 bit</td>
<td>47 bit</td>
<td>0.0244</td>
<td>(4, 1, 105)</td>
<td>–</td>
<td>–</td>
<td>34305</td>
</tr>
<tr>
<td>5000 bit</td>
<td>48 bit</td>
<td>0.0096</td>
<td>(2, 1, 30)</td>
<td>56</td>
<td>379</td>
<td>39</td>
</tr>
<tr>
<td>5000 bit</td>
<td>89 bit</td>
<td>0.0178</td>
<td>(3, 1, 60)</td>
<td>–</td>
<td>–</td>
<td>5783</td>
</tr>
<tr>
<td>5000 bit</td>
<td>113 bit</td>
<td>0.0226</td>
<td>(3, 2, 93)</td>
<td>–</td>
<td>–</td>
<td>74417</td>
</tr>
<tr>
<td>10000 bit</td>
<td>96 bit</td>
<td>0.0096</td>
<td>(2, 1, 30)</td>
<td>56</td>
<td>2500</td>
<td>360</td>
</tr>
<tr>
<td>10000 bit</td>
<td>179 bit</td>
<td>0.0178</td>
<td>(3, 1, 60)</td>
<td>–</td>
<td>–</td>
<td>31226</td>
</tr>
</tbody>
</table>

Table 6.1: Experimental Results
Notice that the experimental results on the achievable sizes of $d_p$ and $d_q$ perfectly match the theoretically predicted bound $\delta$. This is a strong indication that our approach is indeed optimal.

We ran our experiments using Sage 4.1.1 and used the $L^2$ reduction algorithm from Nguyen and Stehlé [NS09]. The calculations were performed on an Quad Core Intel Xeon processor running at 2.66 GHz.
Bibliography


<table>
<thead>
<tr>
<th>Reference</th>
<th>Description</th>
</tr>
</thead>
</table>


