A Hybrid SAT and Lattice Reduction Approach for Integer Factorization

By

Yameen Ajani

A Thesis
Submitted to the Faculty of Graduate Studies
through the School of Computer Science
in Partial Fulfillment of the Requirements for
the Degree of Master of Science
at the University of Windsor

Windsor, Ontario, Canada

2024

A Hybrid SAT and Lattice Reduction Approach for Integer Factorization

by

Yameen Ajani

APPROVED BY:

_____

I. Shapiro
Department of Mathematics and Statistics

_____

A. Biniaz
School of Computer Science

_____

C. Bright, Advisor
School of Computer Science

June 11, 2024

DECLARATION OF CO-AUTHORSHIP / PREVIOUS PUBLICATION

I. Co-Authorship

I hereby declare that this thesis incorporates material that is result of joint research, as follows:

In Chapters 4 and 5 of this thesis, the findings are the result of collaborative work with Dr. Curtis Bright as a co-author. The conceptual framework for the proposed methodology originated from Dr. Bright's initial idea. Additionally, all aspects of the research, including implementation, were accomplished through joint efforts and collaboration.

Chapter 5 integrates unpublished material co-authored with Dr. Bright, complementing the published findings. These additional experiments were conducted collaboratively to augment the existing results and represent a joint effort between the co-authors.

I am aware of the University of Windsor Senate Policy on Authorship and I certify that I have properly acknowledged the contribution of other researchers to my thesis, and have obtained written permission from each of the co-author(s) to include the above material(s) in my thesis.

I certify that, with the above qualification, this thesis, and the research to which it refers, is the product of my own work.

II. Previous Publication

This thesis includes two original papers that have been previously published/submitted to journals for publication, as follows:

Thesis Chapter	Publication Title/Full Citation	Publication Status
Chapter 4, 5	Ajani, Y. and Bright, C. (2023). A hybrid SAT and lattice reduction approach for integer factorization. In Ábrahám, E. and Sturm, T., editors, Proceedings of the 8th SC-Square Workshop co-located with the 48th International Symposium on Symbolic and Algebraic Computation, SC-Square@ISSAC 2023, Tromsø, Norway, July 28, 2023, volume 3455 of CEUR Workshop Proceedings, pages 39–43. CEUR-WS.org.	Published
Chapter 4, 5	Yameen Ajani and Curtis Bright. 2024. SAT and Lattice Reduction for Integer Factorization. In Proceedings of International Symposium on Symbolic and Algebraic Computation 2024 (ISSAC 2024). ACM, New York, NY, USA, 9 pages.	Accepted

I certify that I have obtained a written permission from the copyright owner(s) to include the above published material(s) in my thesis. I certify that the above material describes work completed during my registration as a graduate student at the University of Windsor.

III. General

I declare that, to the best of my knowledge, my thesis does not infringe upon anyone's copyright nor violate any proprietary rights and that any ideas, techniques, quotations, or any other material from the work of other people included in my thesis, published or otherwise, are fully acknowledged in accordance with the standard referencing practices. Furthermore, to the extent that I have included copyrighted

material that surpasses the bounds of fair dealing within the meaning of the Canada Copyright Act, I certify that I have obtained a written permission from the copyright owner(s) to include such material(s) in my thesis. I declare that this is a true copy of my thesis, including any final revisions, as approved by my thesis committee and the Graduate Studies office, and that this thesis has not been submitted for a higher degree to any other University or Institution.

ABSTRACT

The difficulty of factoring large integers into primes is the basis for cryptosystems such as RSA. Due to the widespread popularity of RSA, there have been many proposed attacks on the factorization problem such as side-channel attacks where some bits of the prime factors are available. When enough bits of the prime factors are known, two methods that are effective at solving the factorization problem are satisfiability (SAT) solvers and Coppersmith's method. The SAT approach reduces the factorization problem to a Boolean satisfiability problem, while Coppersmith's approach uses lattice basis reduction. Both methods have their advantages, but they also have their limitations: Coppersmith's method does not apply when the known bit positions are randomized, while SAT-based methods can take advantage of known bits in arbitrary locations but have no knowledge of the algebraic structure exploited by Coppersmith's method. In this thesis we describe a new hybrid SAT and computer algebra approach to efficiently solve random leaked-bit factorization problems. Specifically, Coppersmith's method is invoked by a SAT solver to determine whether a partial bit assignment can be extended to a complete assignment. Our hybrid implementation solves random leaked-bit factorization problems orders of magnitude faster than either a pure SAT or pure computer algebra approach.

DEDICATION

This work is dedicated to my family, the unwavering pillars of my support.

To my mother, whose belief in me has been a guiding light through every twist and turn. In good times and bad, her unwavering faith has been my anchor. I owe everything I am today to her boundless love and encouragement.

To my father, whose steadfast financial support has enabled me to pursue my dreams. His sacrifices and commitment have paved the way for my academic journey.

And to my little sister, the joyful spirit who has kept me grounded with her infectious goofiness. In the midst of the academic challenges, her lightheartedness has been a source of sanity and balance.

This achievement is as much theirs as it is mine. Thank you for being my constant source of strength and inspiration.

ACKNOWLEDGEMENTS

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF ABBREVIATIONS

SAT   Boolean Satisfiability

CAS   Computer Algebra System

GNFS   General Number Field Sieve

RSA   Rivest–Shamir–Adleman Cryptosystem

SSL   Secure Sockets Layer

TLS   Transport Layer Security

CRT   Chinese Remainder Theorem

ECC   Elliptic Curve Cryptography

SRAM   Static Random-Access Memory

DRAM   Dynamic Random-Access Memory

SVP   Shortest Vector Problem

CVP   Closest Vector Problem

LLL   Lenstra–Lenstra–Lovász lattice basis reduction algorithm

CNF   Conjunctive Normal Form

DPP   Davis–Putnam Procedure

BCP   Boolean Constraint Propagation

DPLL   Davis–Putnam–Logemann–Loveland Procedure

CDCL   Conflict Driven Clause Learning

MSB   Most Significant Bit

LSB   Least Significant Bit

CHAPTER 1

Introduction

Integer factorization is a well-studied and important problem in the mathematics and computer science community, both because of its theoretical elegance but also because its difficulty forms the theoretical basis of popular cryptosystems such as RSA. Given an integer N, the factorization problem is to decompose N as a product $N = p_1 \cdots p_k$ where the p_i are prime numbers. Up to ordering of the prime factors p_i (some of which may appear multiple times) the factorization is unique—a fact that was essentially shown by Euclid around 300 BC but not stated in full completeness until 1801 by Gauss [20].

It is unknown if there exists an algorithm that can factor integers in polynomial time in the bitlength of N, at least on a classical computer. The fastest general algorithm discovered to date is the number field sieve [57] which heuristically runs in sub-exponential time. In addition, Shor's algorithm [85] is a quantum-based method that can factor composites in polynomial time subject to the availability of a fault-tolerant quantum computer. The difficulty of factoring integers—especially semiprimes (numbers with exactly two prime factors)—forms the basis many cryptosystems currently in wide usage such as RSA.

A lot of different approaches and methods have been proposed to solve the factorization problem. The most successful methods exploit the algebraic structure inherent in the problem. Another approach reduces the factorization problem to a Boolean satisfiability (SAT) problem that when solved reveals a nontrivial factor of N. In recent years, SAT solvers have achieved great success on many varied kinds of search problems—there are numerous practical and theoretical problems for which

SAT solvers are the most effective known way of solving the problem [12]. Some difficult mathematical problems—such as the resolution of the Boolean Pythagorean triples problem [46] or the computation of the fifth Schur number [44]—have *only* been solved using SAT solvers. Unfortunately, for the factoring problem specifically, the SAT approach is dramatically outperformed by algebraic algorithms [71]. This is not surprising, since although SAT solvers are great general-purpose search tools, they struggle with problems having a mathematical structure unknown to the solver [17].

Recently there have been SAT solvers augmented with a programmatic interface supporting the injection of logical facts as the solver is running [36, 32]. Such an approach has successfully resolved mathematical problems that were beyond the reach of SAT solvers or algebraic methods alone [15]. For example, progress was made on certain mathematical conjectures by extracting mathematical facts from a computer algebra system (CAS) and programmatically passing them to a SAT solver as the solver is running [99]. Augmenting a SAT solver in this way can dramatically improve its effectiveness—intuitively, it is no longer restricted to reasoning on the level of Boolean logic. On the other hand, such a solver can also outperform pure algebraic methods, especially on problems that benefit from efficient search routines. Intuitively, this is because traditionally CASs have not exploited the effective search-with-learning algorithms developed for SAT solvers [1].

In the past decade, the lines between SAT solving and computer algebra is starting to change with the development of numerous hybrid methods exploiting SAT solvers in conjunction with computer algebra. For example, the "SC-square" project combines SAT and computer algebra and has been applied to fields as diverse as economics, dynamic geometry, and knot theory [29].

1.1 The Factorization Problem

The factorization problem, a classical mathematical conundrum, plays a pivotal role in various realms of mathematics and extends its influence into diverse applications, with particular significance in the field of cryptography. At its core, the factorization

problem involves decomposing a composite number into its prime factors. While seemingly straightforward, the computational complexity of this task is not precisely determined; however, it's notable that the Number Field Sieve (NFS), the best-known algebraic method for integer factorization, operates within subexponential time.

In mathematics, factorization has been a subject of intrigue for centuries, with deep connections to number theory and algebra. Its significance is not confined to pure mathematical exploration; rather, it permeates various practical domains. In cryptography, the factorization problem serves as the foundation for widely used security mechanisms, such as RSA encryption. The security of RSA relies on the presumed difficulty of factoring the product of two large prime numbers—a task that becomes prohibitively challenging as the size of the primes increases.

The applications of the factorization problem extend beyond cryptography. It finds utility in coding theory, error correction, and optimization algorithms. Understanding the intricacies of factorization is essential for developing robust cryptographic protocols, ensuring secure communication, and safeguarding sensitive information in an increasingly interconnected digital landscape.

A compelling real-world example showcasing the importance of the factorization problem is its application in securing online transactions. When you make a secure online purchase, the underlying encryption protocols often involve the factorization of large numbers. If the factorization problem were efficiently solvable, the foundation of these cryptographic protocols would crumble, jeopardizing the confidentiality and integrity of sensitive data in e-commerce, banking, and communication.

As of now, the factorization problem remains computationally challenging, and state-of-the-art methods involve algorithms such as the General Number Field Sieve (GNFS). The continued complexity of factorization is vital for the resilience of cryptographic systems. However, the ongoing evolution of computational power prompts a perpetual need for advancing cryptographic techniques to maintain their efficacy in the face of potential future breakthroughs in factorization algorithms.

Understanding the intricacies of the factorization problem, its significance in cryptography, and its broader applications is paramount for researchers and practitioners

navigating the ever-evolving landscape of secure information exchange.

1.2   RSA Cryptosystem

Having established the fundamental role of the factorization problem in cryptography, we delve into one of the most widely employed cryptographic systems that leverages the inherent difficulty of factorization—the RSA (Rivest–Shamir–Adleman) cryptosystem. RSA is a public-key cryptosystem, which means it uses a pair of keys: a public key for encryption and a private key for decryption.

In the context of RSA, let N be the product of two large prime numbers, p and q, i.e., $N = p \cdot q$. The totient of N, denoted as $\phi(N)$, is crucial for RSA key generation. The public key (e, N) consists of an exponent e (often chosen as 65537 for its efficiency) and N. The private key (d, N) involves the exponent d, calculated as the modular multiplicative inverse of e modulo $\phi(N)$.

Mathematically, the public key operation involves encrypting a message M into ciphertext C using the formula $C \equiv M^e \pmod{N}$, while the private key operation decrypts C back to M with $M \equiv C^d \pmod{N}$.

The RSA cryptosystem finds widespread use in securing digital communications, digital signatures, and online transactions. For instance, when you securely transmit sensitive information over the internet, protocols like SSL/TLS employ RSA for key exchange and secure communication.

Chinese Remainder Theorem (CRT)-RSA is a variation that enhances the efficiency of RSA decryption. In CRT-RSA, the private key d is broken into its components using the primes p and q. This involves calculating $d_p \equiv d \pmod{p-1}$ and $d_q \equiv d \pmod{q-1}$, allowing for parallelized computations during decryption. The final result is obtained through the Chinese Remainder Theorem. This variation significantly improves the speed of RSA decryption.

Breaking RSA

The security of RSA relies on the difficulty of factoring the product of two large primes. If an efficient algorithm for factoring large numbers were discovered, it would compromise the security of RSA. Breaking RSA has far-reaching consequences, potentially leading to the unauthorized access of sensitive information, financial fraud, and the compromise of secure communication channels.

In conclusion, the RSA cryptosystem exemplifies the ingenious application of the factorization problem in cryptography, providing a robust framework for secure communication in the digital age. Understanding its mathematical foundations, variations, and potential vulnerabilities is essential for both cryptographic practitioners and researchers aiming to strengthen the security of information exchange systems.

1.3 Side-channel Attacks

Side-channel attacks represent a class of sophisticated techniques aimed at extracting sensitive information from a cryptographic system by exploiting unintentional leakage of information through various physical channels. These attacks leverage indirect information, such as power consumption, electromagnetic emanations, or timing variations, rather than directly attacking the cryptographic algorithm itself. While some side-channel attacks necessitate a deep understanding of a system's internal workings, others, like differential power analysis, can be executed as black-box attacks, requiring minimal technical knowledge.

Several types of side-channel attacks exist, each targeting different aspects of a cryptographic system:

1. **Cache Attack:**
   Cache attacks exploit the shared cache in modern computer systems to infer sensitive information [73]. By monitoring cache accesses made by a victim process, an attacker can deduce patterns in memory access and potentially

extract cryptographic keys or other sensitive data.

2. **Timing Attack:**

   Timing attacks exploit variations in the time taken to execute cryptographic operations [55]. By measuring the execution time of certain operations, attackers can infer information about secret keys or plaintexts, especially in scenarios where cryptographic algorithms exhibit different behaviors depending on input values.

3. **Power-Monitoring Attack:**

   Power-monitoring attacks leverage variations in power consumption by a device during cryptographic operations. By monitoring power consumption patterns, attackers can deduce information about the internal operations of the device, potentially revealing sensitive data such as encryption keys. Simple power analysis and differential power analysis are the two types of power-monitoring attacks [56].

4. **Electromagnetic Attack:**

   Electromagnetic attacks exploit the electromagnetic radiation emitted by electronic devices during operation [67]. By capturing and analyzing these emissions, attackers can extract information about the internal state of the device, including cryptographic keys or data.

5. **Acoustic Cryptanalysis:**

   Acoustic cryptanalysis exploits sound produced by a device during cryptographic operations [37]. By analyzing the acoustic emanations, attackers can infer information about the internal workings of the device and potentially recover sensitive data. Researchers from the University of California performed an experiment [92] that reinforced the viability of these attacks.

6. **Differential Fault Analysis:**

   Differential fault analysis involves inducing faults or errors in a cryptographic device during operation [5]. By observing how the device reacts to these faults,

attackers can gain insights into its internal state and potentially extract sensitive information such as encryption keys.

7. **Data Remanence:**

   Data remanence exploits residual data that remains in memory even after it has been supposedly erased or overwritten. Attackers can recover this residual data through techniques such as the cold boot attack, potentially revealing sensitive information [40].

8. **Allowlist:**

   Allowlist-based side-channel attacks exploit differences in behavior between allowlisted and non-allowlisted devices [97]. By observing how a device responds to allowlisted and non-allowlisted requests, attackers can infer information about its internal state or cryptographic keys.

9. **Optical:**

   Optical side-channel attacks involve capturing sensitive information using optical techniques such as visual recording with high-resolution cameras. By analyzing optical data, attackers can extract information about cryptographic keys or sensitive data stored on a device [82].

In particular, the *Data Remanence Attack* is of special interest to us in this thesis. Data remanence phenomena have been documented in both static random-access memory (SRAM) and dynamic random-access memory (DRAM), despite their contrasting volatility characteristics [18]. Traditionally, SRAM is deemed volatile, meaning its contents degrade upon power loss. Surprisingly, studies have shown data retention in SRAM even at ambient temperatures [86], challenging conventional assumptions about its volatility. Similarly, DRAM, equipped with self-refresh modules, requires periodic refreshing to maintain data integrity. While typically volatile, DRAM exhibits data remanence with retention times ranging from seconds to minutes at room temperature and up to a full week without refresh when subjected to extreme cooling with liquid nitrogen [40].

One specific type of data remanence attack that is well-known is the *Cold boot attack*. This attack method capitalizes on the data remanence property inherent in both DRAM and SRAM, enabling the retrieval of memory contents that remain accessible for seconds to minutes following a power switch-off.

In the realm of computer security, side-channel attacks pose a significant threat and thus, understanding side-channel attacks is crucial for assessing the security of cryptographic implementations and designing robust countermeasures against such threats.

1.4 Lattices and the LLL Algorithm

In the realm of number theory and cryptography, lattices play a fundamental role, particularly in the context of lattice-based cryptography. Before delving into lattice-based cryptography, let us first understand what a lattice is.

Definition 1. *A lattice $\mathcal{L}$ in $\mathbb{R}^n$ is a discrete additive subgroup of $\mathbb{R}^n$ that spans the entire space $\mathbb{R}^n$.*

In simpler terms, a lattice is a set of points in n-dimensional space that are arranged in a periodic pattern with respect to each other. Formally, a lattice $\mathcal{L}$ can be defined as $\mathcal{L} = \{\sum_{i=1}^{n} a_i \mathbf{v}_i : a_i \in \mathbb{Z}\}$ and each $\mathbf{v}_i$ is a linearly independent vector in $\mathbb{R}^n$. It can be visualized as an infinite grid-like structure where each point is an integer linear combination of a set of linearly independent basis vectors (see Figure 1.4.1).

In a lattice $\mathcal{L}$, the basis is a set of linearly independent vectors that span the lattice. Each basis vector defines a direction in the lattice space. If $\mathcal{L}$ is defined as before, then $\mathbf{v}_1, \mathbf{v}_2, \ldots, \mathbf{v}_n$ are the basis vectors of the lattice.

Definition 2. *Let $\mathcal{L} \in \mathbb{Z}^n$ be a lattice with basis $B = \{\mathbf{b}_1, \ldots, \mathbf{b}_n\}$. The determinant of the lattice $\mathcal{L}$, denoted by $\det(\mathcal{L})$, is the n-dimensional volume of the parallelepiped spanned by the basis vectors.*
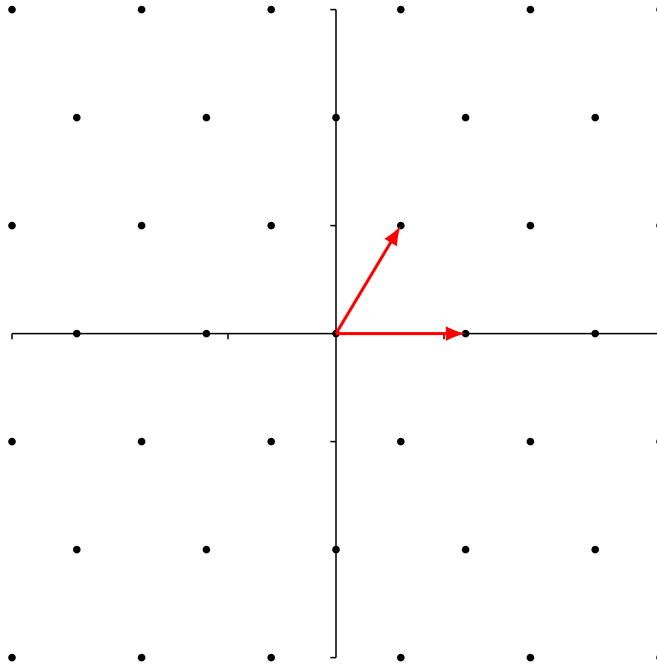
Fig. 1.4.1: A two dimensional lattice generated by two vectors.

In mathematical terms, it can be expressed as:

$$\det(\mathcal{L}) = |\det(B)|$$

where $\det(B)$ is the determinant of the matrix formed by stacking the basis vectors of $\mathcal{L}$ as rows. It is a well-defined notion, meaning that the determinant does not depend on the specific choice of basis used to represent the lattice.

Uses of Lattices

Lattices have diverse applications across various fields, including mathematics, computer science, and cryptography. Some common uses of lattices include:

- Error-correcting codes: Lattices are utilized in the design and analysis of error-correcting codes, particularly in coding theory [34].

- Signal processing: Lattices play a crucial role in signal processing applications such as digital communications and data compression [14].

- Cryptography: Lattices are extensively used in cryptography for various cryptographic primitives, including encryption, digital signatures, and key exchange protocols [34].

Lattices offer several important functionalities and serve as a cornerstone of lattice-based cryptography. In particular, lattices are an essential component of Coppersmith's method that we rely on in our hybrid SAT and computer algebra factorization approach. Some notable broad use cases of lattices in cryptography include:

- Lattice Reduction: Lattice reduction algorithms are utilized to discover short or "good" basis vectors within a lattice. It's essential to note that a lattice can have many possible bases, and the goal of basis reduction is to find vectors that are both short and approximately orthogonal. These algorithms play a crucial role in cryptographic protocols reliant on lattice problems.

- Shortest Vector Problem (SVP): The SVP involves finding the shortest nonzero vector in a lattice. It is a fundamental problem in lattice-based cryptography and serves as the basis for various cryptographic constructions.

- Closest Vector Problem (CVP): The CVP entails finding the lattice point closest to a given target point in space. This problem has applications in lattice-based cryptosystems, particularly in key exchange and digital signature schemes.

For comprehensive discussions on lattices and its use in cryptography, see [69] and [24].

The LLL Algorithm

The LLL (Lenstra–Lenstra–Lovász) algorithm is a powerful tool for lattice basis reduction, named after its inventors Arjen Lenstra, Hendrik Lenstra, and László Lovász [58].

At its core, the LLL algorithm is primarily used to find a "nice" basis for a given lattice, where "nice" refers to a basis that is both short and nearly orthogonal.
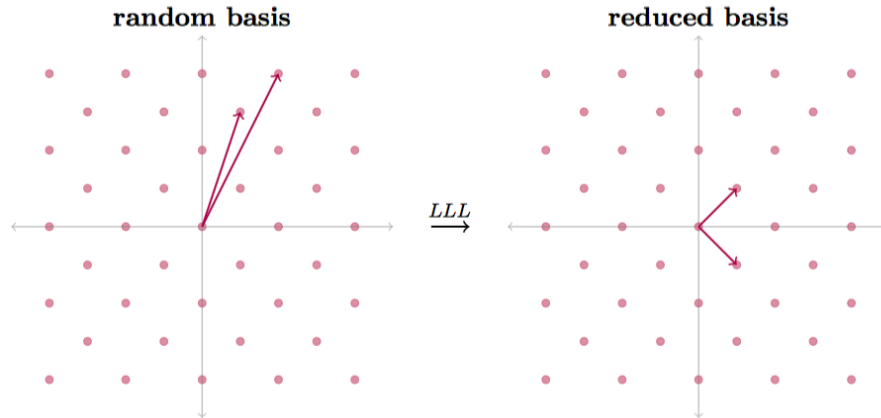
Fig. 1.4.2: Visual representation of lattice basis vectors before and after LLL reduction.

These properties of the reduced lattice basis simplify lattice problems, such as finding short vectors within the lattice, and enable efficient solutions for those problems. The algorithm achieves this reduction through a series of basis transformations, iteratively orthogonalizing vectors while simultaneously reducing their lengths. Figure 1.4.2 gives a very high-level visualization of the effect of the LLL algorithm on the lattice basis vectors.

In the LLL algorithm, one of the key components is the orthogonalization process, which is often implemented using the Gram–Schmidt orthogonalization method (for more details see [60]). This technique transforms a set of basis vectors into an orthogonal set while preserving the span of the original vectors. In the context of lattice basis reduction, Gram–Schmidt orthogonalization plays a crucial role in iteratively orthogonalizing the vectors of the lattice basis during the reduction process.

However, the direct application of Gram–Schmidt orthogonalization to lattice basis vectors may lead to vectors that are not necessarily contained within the lattice. Additionally, it can result in vectors with significantly different lengths, which impacts the quality of the lattice basis. To address this issue, the LLL algorithm introduces modifications to the Gram–Schmidt orthogonalization process to ensure that the resulting basis vectors are nearly orthogonal and of similar lengths.

We now introduce a lemma about LLL-reduced bases that plays a core part in

Coppersmith's method discussed in Section 1.5.

Lemma 3. *Let $\mathcal{L} \subset \mathbb{Z}^n$ be a lattice spanned by $\mathbf{b}_1, \ldots, \mathbf{b}_n$. The first vector $\mathbf{v}$ in an LLL-reduced basis of $\mathcal{L}$ satisfies*

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

Proof. See Lenstra et al.'s paper [58]. □

The LLL algorithm finds extensive application in various areas of cryptography, particularly in Coppersmith's algorithm for solving polynomial equations modulo a number of unknown factorization. In this context, Coppersmith's method leverages the lattice basis reduction capabilities of LLL to efficiently find small roots of polynomials modulo a composite integer. By transforming the integer factorization problem into a lattice-based form and applying the LLL algorithm, Coppersmith's method can factorize semiprime integers efficiently when partial information about the factors is known.

Overall, the LLL algorithm stands as a fundamental tool in lattice-based cryptography, enabling the development of efficient algorithms for challenging computational problems. Its applications extend beyond factorization, encompassing areas such as cryptanalysis, digital signatures, and cryptographic protocols, where lattice-based techniques offer robust security guarantees [14]. The LLL algorithm plays a major role as part of Coppersmith's method in our implementation.

1.5 Coppersmith's Method

Coppersmith's method is a powerful algorithm for finding small integer roots of a polynomial modulo a given integer N [22]. It operates efficiently even when the factorization of N is unknown, making it valuable for various cryptographic applications. The algorithm exploits a fundamental connection between short vectors and polynomials with small coefficients.

The basic idea of Coppersmith's method, involves transforming a polynomial $F(x)$ with integer coefficients into a new polynomial $G(x)$ that preserves its roots modulo N while ensuring that its coefficients are small. This property implies the small roots of $F \mod N$ will be small roots of G *over the integers*. Suppose there exists an integer x_0 such that $F(x_0) \equiv 0 \pmod{N}$, where N is the modulus, and $|x_0| < N^{1/d}$, where d is the degree of the polynomial $F(x)$. If the coefficients of $F(x)$ are sufficiently small, then $F(x_0) = 0$ over the integers (see [22], Theorem 1).

The challenge arises when $F(x)$ has a small solution $x_0 \pmod{N}$ but coefficients that are not small. Coppersmith's insight, as formulated by Howgrave-Graham, is to construct a new polynomial $G(x)$ from $F(x)$ such that $G(x_0) = 0$ over the integers, not just mod N. This is achieved by ensuring that the coefficients of $G(x)$ are small enough and that F and G share the same roots mod N.

Coppersmith's method can handle both modular univariate and multivariate polynomials. For univariate polynomials, it constructs a lattice such that each lattice vector corresponds to a polynomial with x_0 as a root modulo N. By using lattice reduction techniques like the LLL algorithm, Coppersmith's algorithm finds an integer polynomial that has integer roots matching all "small" roots of the input polynomial modulo N. Since the integer roots of a polynomial can be computed in polynomial time [90] this reduces the problem of finding the root $x_0 \pmod{N}$ to the problem of finding a short vector in Coppersmith's lattice.

In the case of multivariate polynomials, Coppersmith's method extends the lattice construction to higher dimensions, where each lattice vector corresponds to a multivariate polynomial. This approach enables the efficient detection of solutions to systems of polynomial equations modulo N, making it invaluable for various cryptographic tasks, including integer factorization and solving polynomial congruences.

Coppersmith's method is instrumental in analyzing and potentially breaking fixed padding schemes used in RSA encryption. By exploiting the algebraic structure of RSA ciphertexts and employing Coppersmith's algorithm, researchers can uncover vulnerabilities in cryptographic systems that rely on fixed padding. In scenarios where the top or bottom 50% of the bits of one prime factor p of the RSA modulus

N is available, Coppersmith's method can efficiently factorize N. It is unknown if there is an efficient method that works with less than 50% of the bits, but in general Coppersmith's method won't work with less than 50% known bits (see [22] for additional details). This capability is crucial for cryptanalysts attempting to break RSA-based encryption systems when partial information about the primes is leaked or obtained through side-channel attacks. Moreover, Coppersmith's algorithm extends beyond the standard RSA modulus $N = pq$ and can also be applied to factorize numbers of the form $p^r q$, where p and q are distinct primes. This flexibility enables the algorithm to address a broader range of factorization challenges encountered in cryptographic protocols and security analyses. Coppersmith's method also offers a solution to the Chinese Remainder Theorem (CRT) list decoding problem, which arises in various cryptographic contexts, including error correcting codes and lattice-based cryptography. A detailed explaination of all these applications and more can be found in [68] and [35].

Overall, Coppersmith's method plays a crucial role in modern cryptography by providing efficient algorithms for tackling complex computational tasks, ranging from integer factorization to error correction and decoding in cryptographic protocols. Its broad applicability and effectiveness make it a valuable tool for cryptanalysts and researchers in the field.

1.6 Our Contributions

In this thesis, we introduce a new programmatic SAT method that dramatically improves the performance of SAT solvers on leaked-bit integer factorization problems by exploiting algebraic structure of the problem that would otherwise be hidden from the solver. More precisely, we employ Coppersmith's method [22] for finding small roots of polynomials modulo a number N using lattice basis reduction.

Coppersmith's method can factorize a semiprime N in polynomial time when either the top half or the bottom half of the bits of one of its prime factors is known [70]. We exploit the algebraic structure revealed by Coppersmith's method in program-

matic SAT solvers MapleSAT [61] and CaDiCaL [11] by querying a computer algebra system supporting the necessary lattice basis reduction routines. The information provided by Coppersmith's method is translated into logical facts that the solver uses to backtrack much earlier than it otherwise would, dramatically improving the performance of the solver.

It should be stressed that our approach is not directly competitive with the best algebraic methods for the integer factorization problem. However, due to the practical importance of the factoring problem it has long been of interest to study weakenings of the factorization problem where some information about the prime factors are assumed to be known in advance. In practice, such information may be leaked through side-channel attacks (see Section 1.3). In our work, we consider random leaked-bit factorization problems—i.e., where random bits of the prime factors of the number to factor are known, but the attacker *has no control over which bits are leaked.*

Although Coppersmith's method requires only half of the bits of the prime factors to be known (see Section 1.5), the method requires the known bits to be *consecutive*— ideally either the high bits or low bits of one of the prime factors. Coppersmith's method can be adapted to work with multiple chunks of known bits, but it scales poorly as the number of chunks increases [70]. Thus, in general Coppersmith's method does not apply when the known bit positions are distributed uniformly at random.

Conversely, our method takes advantage of known bits from arbitrary positions but also takes advantage of the algebraic relationships revealed by Coppersmith's method. Besides that, we also provide the option to use a custom branching heuristic that uses the problem's inherent algebraic structure (see Section 5.2.1 for details) to branch on variables that help apply Coppersmith's method earlier as compared to the solver's default branching heuristic. Our results, discussed in Section 5.3, show that the augmented SAT solver is orders of magnitude faster than an off-the-shelf SAT solver. It also outperforms a brute-force approach of trial division by all factors consistent with the known bits, even if Coppersmith's method is used to speed up the brute-force guessing (see Section 5.3). With enough known bits our method even outperforms the fastest general-purpose factoring algorithms such as the number

field sieve, though we admit this is not really a fair comparison since the number field sieve cannot exploit known bits and hence is at a disadvantage for the leaked-bit factorization problem we consider in this paper.

In summary, the significance of our method is that when enough bits are known it outperforms algebraic methods, pure SAT methods, and a brute-force + Coppersmith method. All scripts and code used for the experiments performed in this thesis can be found in a public GitHub repository at `https://github.com/yameenajani/SAT-Factoring`.

Our research has yielded significant contributions, resulting in two publications. The first publication (see [6]) was an extended abstract presented at the 8th SC-Square Workshop co-located with the 48th International Symposium on Symbolic and Algebraic Computation (ISSAC 2023). Additionally, our work titled "SAT and Lattice Reduction for Integer Factorization" has been accepted for presentation at the International Symposium on Symbolic and Algebraic Computation 2024 (ISSAC 2024).

CHAPTER 2

Boolean Satisfiability

Boolean Satisfiability (SAT) is a fundamental problem in computer science and mathematics, particularly within the field of computational complexity theory. At its core, SAT involves determining whether a given Boolean formula can be satisfied by assigning Boolean values (true or false) to its variables in such a way that the entire formula evaluates to true.

Significance

The significance of the SAT problem lies in its broad applicability and its status as the first problem proven to be NP-complete. In computational complexity theory, NP stands for "nondeterministic polynomial time", which refers to the class of problems for which a solution can be verified in polynomial time. A problem is NP-hard if every problem in NP can be reduced to it in polynomial time, indicating that it is at least as hard as the hardest problems in NP. A problem is NP-complete if it is both NP-hard and NP. This classification means that SAT is among the hardest problems in the class of NP problems because every problem in NP reduces to it. Therefore, if we could solve the SAT problem efficiently, we could solve all NP problems efficiently. This has profound implications across various fields, including factoring.

It's important to note that while factoring is an NP problem, it is not expected to be NP-hard. This means that although a solution to the factoring problem can be verified in polynomial time, it is not believed that every problem in NP can be reduced to factoring in polynomial time.

Applications

- **Hardware and Software Verification:** SAT solvers are widely used in the verification of digital circuits, software systems, and protocols, ensuring their correctness and reliability [39].

- **Automated Reasoning:** Many automated reasoning tasks, such as theorem proving and model checking, can be reduced to SAT, enabling efficient solutions to complex logical problems [10].

- **Planning and Scheduling:** SAT solvers find applications in scheduling tasks, resource allocation, and planning problems, optimizing processes in various domains including manufacturing and logistics [52].

- **Cryptography:** SAT solvers are used in cryptanalysis for breaking cryptographic primitives based on Boolean operations, aiding in the evaluation of cryptographic security [80].

- **Artificial Intelligence:** SAT solvers are employed in various AI applications, including constraint satisfaction problems, optimization, and decision-making [52].

Formal Definition

Formally, given a Boolean formula, the Boolean Satisfiability (SAT) problem asks whether there exists an assignment of Boolean values to the variables such that the entire formula evaluates to true.

Now that we have formally defined the SAT problem a natural question arises: why can't we simply resort to using something as simple as truth tables? After all, truth tables are a concept familiar to many from their school days.

However, the simplicity of truth tables belies a significant limitation when it comes to tackling complex Boolean formulas. Imagine a Boolean formula with just a handful of variables. Even then, the truth table would need to enumerate every possible

combination of true and false values for those variables. As the number of variables increases, the truth table quickly balloons in size, growing exponentially with each additional variable.

This exponential explosion in complexity makes truth tables impractical for larger formulas. For instance, a formula with just 20 variables would require over a million rows in its truth table. Imagine dealing with formulas involving hundreds or thousands of variables – the truth table approach becomes entirely unfeasible due to the sheer volume of possibilities to consider.

To address these challenges, SAT solvers offer a more efficient alternative. Rather than exhaustively enumerating all possible assignments, SAT solvers employ sophisticated algorithms to systematically explore the solution space, guided by efficient data structures and search techniques.

By representing Boolean formulas in a compact form, typically as clauses in conjunctive normal form (CNF), SAT solvers can leverage various optimization strategies and heuristics to efficiently search for satisfying assignments. This approach, known as a search-based algorithm, enables SAT solvers to navigate the solution space with remarkable efficiency, even for formulas with thousands or millions of variables.

What is a Clause?

In Boolean logic, a clause is a fundamental building block used to construct Boolean formulas. A clause is essentially a disjunction (logical OR) of literals, where each literal represents either a variable or its negation.

Formally, a clause can be expressed as follows:

$$C = l_1 \vee l_2 \vee \ldots \vee l_n$$

where $l_1, l_2, \ldots, l_n$ are literals.

For example, consider the following clause:

$$C = x_1 \vee \neg x_2 \vee x_3$$

This clause consists of three literals: x_1, $\neg x_2$, and x_3, where $\neg$ represents negation.

Clauses are often used in conjunction with each other to form Boolean formulas, particularly in CNF.

Conjunctive Normal Form

In Conjunctive Normal Form (CNF), a Boolean formula is expressed as a conjunction (logical AND) of clauses.

For instance, the CNF representation of a Boolean formula may look like this:

$$F = (x_1 \vee \neg x_2) \wedge (\neg x_1 \vee x_3) \wedge (x_2 \vee x_3 \vee x_4)$$

Each term inside the parentheses is a clause, and the entire formula is a conjunction of these clauses.

Clauses are fundamental to various applications of Boolean logic, including SAT problems. Most modern SAT solvers require the input expression to be in CNF.

2.1 Tseitin Transformation

In Boolean Satisfiability, transforming an arbitrary logical expression into CNF in a reasonable amount of time is of utmost importance. The Tseitin transformation [88] offers a solution to this challenge by converting formulas into CNF in polynomial time.

The Tseitin transformation, named after Grigoriy Tseitin who introduced it in 1968, is a method used to convert a given logical formula into an equisatisfiable CNF form. While the transformed formula is not logically equivalent to the original one due to the introduction of new variables, it preserves satisfiability. This property makes it a valuable tool in automated theorem proving, SAT solving, and various other applications.

The main idea behind the Tseitin transformation is to introduce auxiliary variables to represent subexpressions of the original formula. By doing so, the formula can be

rewritten as a conjunction of clauses, each of which corresponds to a gate in the Boolean circuit representation of the formula.

How Tseitin Transformation Works

Consider the logical expression $(p \lor q) \land \neg(p \land q)$ represented as a Boolean circuit:
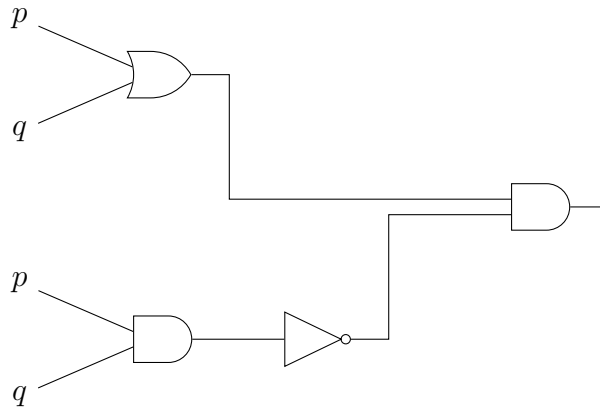
Fig. 2.1.1: Example Boolean Circuit - Tseitin Transformation

To apply the Tseitin transformation, new variables are introduced for the outputs of each gate. Let x, y, and z represent the outputs of the OR gate, AND gate, and NOT gate, respectively. These new variables are defined as follows:

$$x \leftrightarrow (p \lor q)$$
$$y \leftrightarrow (p \land q)$$
$$z \leftrightarrow \neg y$$

Now, the transformed expression can be written as $x \land z$ along with CNF representations of the definitions of x, y, and z:

$$(\neg x \vee p \vee q) \wedge (\neg p \vee x) \wedge (\neg q \vee x)$$

$$(y \vee \neg p \vee \neg q) \wedge (\neg y \vee p) \wedge (\neg y \vee q)$$

$$(y \vee z) \wedge (\neg y \vee \neg z)$$

This resulting CNF expression is equisatisfiable to the original logical expression.

The Tseitin transformation operates in polynomial time, making it efficient for practical use. Suppose n is the number of operators in the input expression. For each operator, the transformation introduces new variables and generates CNF clauses in constant time. Thus, the overall time complexity of the transformation is $O(n)$. Due to this, the Tseitin transformation finds applications in various fields such as formal verification, hardware design, and computer-aided reasoning. It is a fundamental technique in automated theorem proving and SAT solving, enabling efficient manipulation and analysis of logical formulas.

For more in-depth discussions on the Tseitin transformation and its applications, refer to [12, 66].

2.2 The Backbone of SAT - DPLL Algorithm

The DPLL algorithm [25] (Davis–Putnam–Logemann–Loveland) is a cornerstone technique in SAT solving, devised to efficiently determine the satisfiability of Boolean formulas. It builds upon the principles of the Davis–Putnam procedure [26] while incorporating additional strategies to enhance efficiency and scalability.

At its essence, the DPLL operates by exhaustively exploring the space of possible truth assignments to the variables in a given formula. It begins by converting the input formula into CNF. This conversion simplifies the structure of the formula and facilitates the application of various reduction techniques.

One significant enhancement introduced by the DPLL algorithm is Boolean Constraint Propagation (BCP) [95]. BCP involves iteratively propagating the effects of

variable assignments throughout the formula, updating the state of other variables based on the implications of these assignments. This propagation process helps identify additional variable assignments that can be made with certainty, further reducing the search space. Let's consider a simple example using the following clause set:

$$\{\{a, b\}, \{c\}, \{\neg c, \neg b, \neg a\}, \{a, c\}, \{b, c\}\}$$

The positive unit clause $\{c\}$ appears in the clauses above. Thus we can assign c to true and this allows us to simplify many other clauses—any clause that contains c is now automatically satisfied and can be removed. Also, $\neg c$ can be dropped from any clause that contains it, since $\neg c$ is now always false. After applying these BCP simplifications, the original clause set becomes

$$\{\{a, b\}, \{\neg b, \neg a\}\}.$$

The DPLL algorithm operates recursively, employing a depth-first search strategy to explore the solution space. At each step, it makes decisions regarding variable assignments and then applies BCP to propagate these assignments and simplify the formula. If a conflict arises during BCP, indicating that the current partial assignment cannot lead to a satisfying solution, the algorithm backtracks to the most recent decision point and explores alternative assignments.

Throughout the search process, the DPLL algorithm dynamically adjusts its strategy, employing heuristics to guide decision-making and variable selection. These heuristics aim to prioritize variables or assignments that are likely to lead to a solution or quickly identify infeasible branches, thereby improving the efficiency of the search process.

By iteratively applying decision-making, propagation, and backtracking, the DPLL algorithm effectively navigates the solution space, gradually narrowing down the possibilities until a satisfying assignment is found or it is determined that no such assignment exists. Its versatility, combined with its ability to handle large and complex problem instances, has established it as a fundamental component of modern SAT

solvers, underpinning their effectiveness in solving a wide range of practical problems.

2.3   Conflict Driven Clause Learning

The introduction of conflict-driven clause learning (CDCL) in SAT solving, pioneered by Marques-Silva and Sakallah in 1996 [65], marked a significant advancement in the field. This technique, combined with non-chronological backtracking, revolutionized SAT solvers and propelled them into a wide array of applications.

At the heart of CDCL lies the idea that when a conflict arises during the search for a satisfying assignment, instead of simply backtracking and trying a different assignment, the solver learns from the conflict. By analyzing the reason behind the conflict, the solver derives a new clause that encodes this information, preventing similar conflicts in the future. These learned clauses not only guide the solver away from unproductive search paths but also contribute to the overall efficiency of the solving process.

Non-chronological backtracking, another key feature of CDCL, provides flexibility in exploring the search space. Unlike traditional chronological backtracking, which revisits decisions in a linear fashion, non-chronological backtracking allows the solver to jump back multiple levels in the search tree, potentially bypassing unfruitful branches and reaching more promising areas of the solution space.

When learning a conflict clause, the CDCL method focuses on identifying the decision variables that led to the conflict, excluding those derived from BCP. By prioritizing these decision variables, CDCL generates conflict clauses that are concise yet effective, enhancing the solver's ability to detect and resolve conflicts efficiently.

The implication graph serves as a foundational tool in CDCL, representing the sequence of variable assignments and propagations made during BCP. This directed acyclic graph captures the causal relationships between assignments, facilitating the generation of optimized conflict clauses based on the structure of the search space.

Moreover, CDCL continually seeks to improve conflict clause generation, exploring various strategies to derive clauses that are both informative and concise. By

leveraging the information encoded in the implication graph, CDCL generates conflict clauses tailored to the specific characteristics of the problem instance, leading to more effective conflict resolution and overall performance gains.

Determining the optimal backtracking strategy is crucial in CDCL, balancing the need to backtrack sufficiently to avoid conflicts with the goal of minimizing unnecessary overhead. The solver aims to backtrack to the furthest possible point while ensuring that the learned conflict clause remains a unit clause under the current partial assignment, maximizing its utility in subsequent propagation steps.

Branching heuristics play a crucial role in the efficiency of SAT solvers, guiding the selection of decision variables to explore the search space effectively. Two prominent branching heuristics in the context of CDCL are the Learning Rate Based Branching Heuristic and the VSIDS (Variable State Independent Decaying Sum) Branching Heuristic. The Learning Rate Based Branching Heuristic prioritizes variables that frequently appear in learned clauses, leveraging the intuition that such variables are critical in resolving conflicts and thus promising candidates for decision making [62]. This approach dynamically adjusts the importance of variables based on their involvement in conflicts, aiming to accelerate the solver's progress. On the other hand, the VSIDS Heuristic assigns scores to variables based on their recent activity in conflicts, with these scores decaying over time to emphasize more recent conflicts [72]. This decay mechanism helps the solver adapt to changing problem characteristics, maintaining a focus on variables that are currently most influential in the search process. Both heuristics are instrumental in enhancing the performance of CDCL solvers by efficiently navigating the search space and resolving conflicts swiftly.

CHAPTER 3

Related Works

This chapter explores various works done on topics related to this thesis. Some of these works have helped lay the foundation for the work done in this thesis. The objective of this chapter is to make the reader aware about the works done in different aspects related to the project so as to develop a better understanding about the background and history related to the topic of this thesis.

3.1   Integer Factorization & Algebraic Methods

The quest to develop efficient methods for integer factorization has been a focal point in the realms of mathematics and cryptography for decades. At the heart of this endeavor lies the critical role of prime factorization in various cryptographic protocols and security systems. Given the pivotal role of integer factorization in cryptography, numerous researchers have dedicated their efforts to devising innovative algorithms capable of efficiently decomposing large composite numbers into their prime factors. These efforts have led to the exploration of diverse approaches ranging from classical methods to more advanced algebraic techniques. Understanding and evaluating these methods is crucial not only for enhancing our theoretical understanding of factorization but also for developing practical solutions to ensure the security of modern cryptographic systems. In this section, we delve into the realm of factoring and algebraic methods, exploring a plethora of research endeavors aimed at advancing the state-of-the-art in integer factorization.

As the pursuit of efficient factorization algorithms continued, the cryptographic

community faced a compelling challenge that galvanized research efforts and spurred innovation: the RSA Decomposition Challenge. Initiated in the 1990s by RSA Security, this challenge presented a formidable task: to factorize specific RSA modulus values into their prime factors. The challenge served as a litmus test for the efficacy of factorization methods and provided a real-world benchmark to assess the progress in the field. The promise of substantial financial rewards for successfully factoring RSA moduli fueled intense competition and inspired researchers to explore novel avenues in factorization research.

We find a rich tapestry of methodologies and techniques that have evolved over time to tackle the challenge of decomposing large integers into their prime factors. These algorithms can be broadly categorized into three main categories:

Special Attribute Algorithms

These algorithms leverage specific properties or attributes of the numbers being factored to expedite the factorization process. For instance, Trial Division is one of the simplest methods for factoring integers. It involves systematically dividing the target number by smaller primes to check for divisibility. While straightforward, it becomes increasingly inefficient for larger numbers due to the sheer number of potential divisors that need to be tested. Pollard's rho algorithm [77] is a probabilistic algorithm used for integer factorization that also falls into this category. It is based on the principle of cycle detection in randomly generated sequences. By iteratively applying a function to generate values and detecting cycles in the sequence, the algorithm can identify factors of composite numbers. Another contribution by Pollard, Pollard's $p-1$ algorithm [76] is particularly effective for factoring number N where there exists a prime p which divides N such that $p-1$ is 'smooth', meaning it has only small prime factors. It exploits the properties of the multiplicative order of integers modulo a prime to find factors. Another algorithm belonging to this category is the Lenstra Elliptic Curve Decomposition algorithm [59] which leverages the properties of elliptic curves over finite fields to factorize integers. It involves finding points on an elliptic curve modulo the target integer and utilizing properties of group operations to deduce

factors. We also have the Fermat Factorization Method [27] exploiting the difference of squares to express composite integers as the difference between two squares. It is particularly efficient when the factors of the target number are close to each other.

General Decomposition Algorithms

In contrast to the special attribute algorithms, general decomposition algorithms aim to factorize large integers without relying on specific properties of the numbers. Prominent examples in this category include the Dixon algorithm, Quadratic sieve, Rational sieve, and General Number Field Sieve. Most of these methods rely heavily on Sieve Theory and more information about it can be found in [19].

The Dixon algorithm [28] is a general-purpose integer factorization method that seeks to express a composite number as the product of two relatively small numbers. It relies on a combination of sieving techniques and linear algebra to identify pairs of integers whose product yields the target number. The Quadratic Sieve [78] is a powerful integer factorization algorithm that works by sieving for smooth numbers and then combining them to find congruences modulo the target number. These congruences are then used to solve linear equations, ultimately leading to the factorization of the composite integer. The Rational Sieve is a variant of the Quadratic Sieve that incorporates rational arithmetic to improve efficiency. It aims to find rational approximations to square roots that facilitate the detection of congruences modulo the target number. This method can be particularly effective for certain classes of integers. The General Number Field Sieve (GNFS) [57] is the most efficient known algorithm for factoring large integers. It is a sophisticated sieving algorithm that operates in a number field, utilizing algebraic number theory and advanced mathematical techniques. GNFS is capable of factoring numbers with hundreds of digits, making it indispensable for cryptographic applications. These algorithms represent some of the most advanced and efficient methods for integer factorization.

Beyond the conventional methods, there exist innovative approaches that push the boundaries of factorization research. One notable example is Shor's algorithm [85],

which exploits the principles of quantum computation to achieve an exponential speedup in factorizing large integers.

3.2   RSA Key Reconstruction

The motivation behind the research into RSA key reconstruction is two-fold. On one hand, the surge in computational resources has led to intensified efforts to enhance the efficiency of key factorization attacks. These efforts have driven researchers to explore novel strategies, including utilizing partial information about RSA keys to expedite the factorization process. On the other hand, the advent of side-channel attacks—attacks that exploit unintended information leakage from the physical implementation of a cryptographic system—has unveiled vulnerabilities that allow attackers to gain insights into key components. This section aims to elucidate the landscape of RSA key reconstruction, exploring seminal works that delve into the vulnerabilities brought to light by partial key knowledge.

The paper by Heninger and Shacham [43] presents a groundbreaking algorithm for reconstructing RSA private keys using a fraction of randomly known bits. This work has been pivotal in the field of RSA cryptanalysis, serving as a foundation for subsequent research and optimizations including this thesis. Motivated by the threat of cold boot attacks, the authors address the practical implications of key reconstruction. Leveraging the redundancy in key data and the public modulus N, they propose a method to reconstruct partially known factors p and q. Focusing on CRT-RSA, the algorithm targets a special case where the public exponent e is small ($e = 3$), a common scenario observed in real-world TLS data [93]. The key claim of the paper is the efficient reconstruction of RSA keys in polynomial time, with specified fractions of known bits of p, q, d, d_p, and d_q. For instance, the algorithm can factor RSA modulus in an expected polynomial time if 42% bits of each of p, q and d are known. The reconstruction algorithm used is defined as a "branching and pruning" approach which is essentially just a "smart" brute-force approach.

As an extention of this work, a lattice-based approach to key reconstruction was

introduced [64]. Unlike the previous work, which primarily focused on brute-force methods, this paper delved into the combinatorial aspects of key reconstruction algorithms. The authors scrutinized the search tree expansion of the method proposed in the previous work. In contrast to the exhaustive search approach of the original algorithm, the authors advocate for a more efficient strategy that employs Coppersmith's algorithm [22] to recover the remaining bits after assigning 50% of the low bits. Furthermore, the paper introduces a novel reconstruction algorithm that operates from the most significant bit (MSB) side, diverging from previous methods that start from the least significant bit (LSB) side. This new algorithm exhibits polynomial time complexity relative to $\log N$.

Besides these works, a survey [70] published in 2024 is an exceptional source that contains a compilation of all possible variations of RSA key reconstruction with partial information. Not just that, the survey also explains each case with examples for better understanding. It covers straightforward scenarios as well as scenarios involving more complex implementations like lattice reductions. The survey highlights the cases in which a particular method can be used and also points out any shortcomings of the method.

3.3 The Magic of SAT

There have been various research endeavours aimed at leveraging SAT solvers for the challenging task of factoring semi-prime numbers. Initially explored by Schoenmackers and Cavender, early efforts focused on optimizing SAT formulations for prime factorization, proposing constraints to narrow the search space and enhance solver efficiency [84]. However, scalability issues prompted further investigation, leading to Forsblom and Lundén's exploration of parallel SAT solvers to improve performance and efficiency [33]. Their study compared sequential and parallel approaches, assessing effectiveness, speedup, and efficiency metrics, shedding light on the strengths and limitations of each method. Building upon this foundation, Mosca and Verschoor delved into quantum SAT solvers' potential for factoring, analyzing classical and

quantum algorithms' runtime and efficiency [71]. Despite promising advancements, scalability challenges and limitations of quantum computing were highlighted.

Patsakis' work stands out for its innovative approach to cryptanalysis using SAT solvers [75]. He focused on reconstructing RSA private keys from partial knowledge, leveraging SAT solvers to model RSA problems efficiently. By formulating equations relating known and unknown key bits and converting them into SAT instances, Patsakis demonstrated the feasibility of breaking RSA keys with limited information. He utilized the MiniSat solver to test various scenarios, showing that even with partial knowledge of key bits, RSA keys could be recovered efficiently. While acknowledging the study's specialization within the broader context of RSA factorization, the work showed the potential of SAT solvers in cryptanalysis and proved to be one of the primary motivations for the work presented in this thesis.

The fusion of SAT solvers with computer algebra systems (CASs) marks a pivotal convergence between two traditionally distinct fields: satisfiability checking and symbolic computation. This interdisciplinary synergy, spearheaded by Ábrahám [1] and Zulkoski et al. [100], has led to transformative research endeavours, epitomized by initiatives like the SC-Square project [4]. Prior to these groundbreaking developments, the domains of satisfiability checking and symbolic computation operated largely in silos, each with its own set of methodologies and objectives.

The integration of SAT solvers into CAS environments has opened up a rich tapestry of applications across a spectrum of disciplines. For instance, in the realm of hardware verification, researchers have utilized SAT+CAS techniques to rigorously verify the correctness of complex multiplier circuits [49]. In computational algebra, novel algorithms for matrix multiplication have been devised, leveraging the combined strengths of SAT solving and symbolic manipulation [45]. Moreover, in theoretical mathematics, SAT+CAS approaches have contributed to the exploration of conjectures in geometric group theory, shedding new light on fundamental mathematical problems [83].

Beyond theoretical realms, the impact of SAT+CAS extends to practical domains such as digital circuit design. By integrating SAT solving capabilities into CAS frame-

works, researchers have developed sophisticated tools for debugging digital circuits, streamlining the identification and resolution of design flaws [63]. Additionally, in combinatorics, SAT+CAS methodologies have enabled the generation of combinatorial objects in an isomorph-free manner, facilitating efficient exploration of complex combinatorial spaces [53].

The collaboration between the SAT and CAS communities not only enhances the computational capabilities of symbolic computation but also fosters interdisciplinary collaborations that transcend traditional boundaries. By harnessing the complementary strengths of SAT solvers and computer algebra systems, researchers are poised to tackle increasingly complex challenges across a wide range of scientific and engineering domains. This symbiotic relationship between SAT and CAS heralds a new era of interdisciplinary research, driving innovation and advancing knowledge at the intersection of computer science, mathematics, and beyond.

CHAPTER 4

Methodology

4.1 The Hybrid Approach

This section introduces the core concept of our thesis: the hybrid SAT + Computer Algebra System (CAS) approach. We present a novel programmatic SAT method designed to significantly enhance the performance of SAT solvers when tackling integer factorization problems. Our approach capitalizes on the inherent algebraic structure of these problems, which is often obscured from traditional solvers.

Specifically, we leverage Coppersmith's method, a renowned technique for identifying small roots of polynomials modulo a number N using lattice basis reduction. This method demonstrates the ability to factorize a semiprime N in polynomial time when partial knowledge about one of its prime factors is available [70].

By integrating Coppersmith's method into programmatic SAT solvers such as MapleSAT [61] and CaDiCaL [11], we harness the insights gleaned from algebraic computations to inform the SAT solving process. This involves translating the information provided by Coppersmith's method into logical constraints that guide the solver's decision-making, enabling it to backtrack more efficiently and thus drastically improving overall performance.

It's worth noting that our approach does not directly compete with the most advanced algebraic methods for integer factorization. However, the practical significance of the factorization problem lies in its susceptibility to weakenings where partial information about prime factors is assumed. Such information leakage can occur through side-channel attacks.

In our research, we focus on random leaked-bit factorization problems, where bits of the prime factors are known to the attacker but are randomly leaked without the attacker's control. Our method shines particularly in scenarios where a sufficient number of known bits are available, surpassing the performance of pure algebraic methods, traditional SAT approaches, and even brute-force techniques combined with Coppersmith's method.

Through our hybrid approach, we aim to address the practical challenges associated with integer factorization in real-world scenarios, demonstrating the efficacy of combining algebraic insights with SAT solving techniques to tackle challenging computational problems.

4.1.1 Factoring with Coppersmith

This section provides an overview of Coppersmith's method as applied in the context of factorization. For a general overview of this method, please refer to Chapter 1, Section 1.5.

Assume we have a semiprime $N = p \cdot q$, such as $N = 16803551 = 2837 \cdot 5923$. Coppersmith's method proves effective when at least 50% of the most significant bits (MSBs) of p are known. In such cases, p can be expressed as $p = \tilde{p} + x_0$, where $\tilde{p}$ shares at least 50% of its MSBs with p, and x_0 represents the unknown low bits of p. For example, if $p = 2837$ and $\tilde{p} = 2830$, then $x_0 = 7$.[1]

As detailed in Chapter 1, Section 1.5, Coppersmith's method seeks small roots x_0 of a polynomial $f(x)$ modulo an integer. In the context of factorization, the modulus is the prime p. Although p remains unknown, our knowledge of N (a multiple of p) suffices. Coppersmith's method identifies a small x_0 such that $f(x_0) \equiv 0 \pmod{p}$, thereby ensuring that $f(x_0) \bmod N$ is divisible by p. Thus, we extract p by computing the greatest common divisor with N. We select $f(x) = \tilde{p} + x$, ensuring that $f(x_0) = p \equiv 0 \pmod{p}$.

Next, consider the polynomials $f(x)$, $xf(x)$, $x^2 f(x)$, and the constant polynomial N (note that x_0 serves as a root for each of these polynomials modulo p). Lattice ba-

---

[1]This example uses demical numbers for ease of explanation.

$$\begin{pmatrix} 16803551 & 0 & 0 & 0 \\ 2830 & 10 & 0 & 0 \\ 0 & 28300 & 100 & 0 \\ 0 & 0 & 283000 & 1000 \end{pmatrix} \xrightarrow{\text{LLL Reduction}} \begin{pmatrix} 105 & -1200 & 800 & 1000 \\ 336 & -1040 & 1500 & -1000 \\ 1094 & 1580 & 600 & 1000 \\ -1400 & 530 & 2100 & 0 \end{pmatrix}$$

Convert first row to polynomial

$$x_0 = 7 \xleftarrow{\text{Solve}} x^3 + 8x^2 - 120x + 105$$
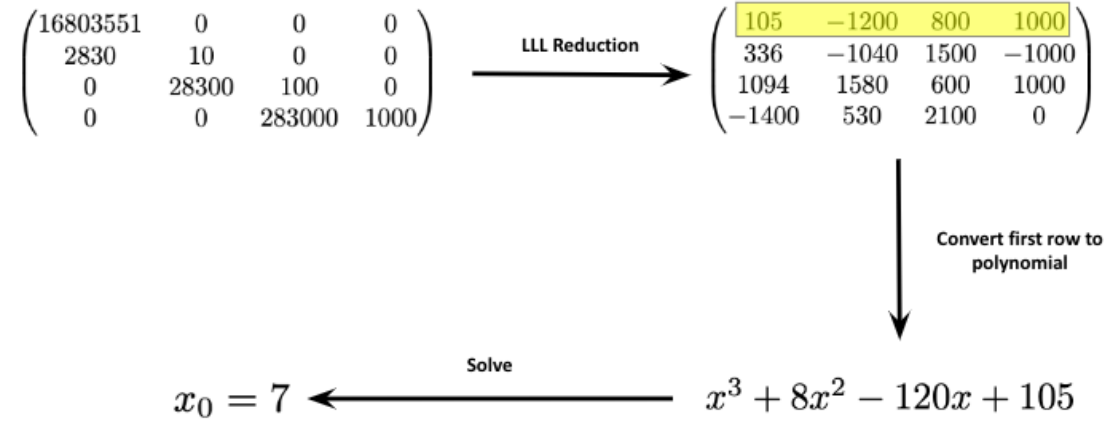
Fig. 4.1.1: Example demonstrating the working of Coppersmith's method, where $N = 16803551$ and $f(x) = 2830 + x$. After applying lattice basis reduction, the short polynomial $x^3 + 8x^2 - 120x + 105$ is discovered, with the integer root $x_0 = 7$. Subsequently, $f(x_0) = 2837$ is identified as a factor of N.

sis reduction is applied to the lattice basis generated by $\{N, f(x), xf(x), x^2 f(x)\}$, where a polynomial $a_0 + a_1 x + a_2 x^2 + a_3 x^3$ is represented by the lattice vector $(a_0, 10a_1, 100a_2, 1000a_3)$ or, more generally, $(a_0, Xa_1, X^2 a_2, X^3 a_3)$, with X serving as an upper bound on the size of x_0. Upon uncovering a short vector of the lattice, integer root detection reveals the small root $x_0 < 10$, from which $p = \tilde{p} + x_0$ is obtained. Refer to Figure 4.1.1 for a diagram illustrating the working of Coppersmith's method.

Factoring with LSBs

Coppersmith's method can also factorize N if at least 50% of the least significant bits (LSBs) of one of the primes are known. Let $\tilde{p}$ represent the integer value corresponding to the m least significant bits of p. We seek a small root $x_0 \pmod{p}$ of $2^m \cdot x + \tilde{p}$. If N has $2k$ bits and p has k bits, the root $x_0 \bmod p$ should be an integer at most 2^{k-m}.

To ensure that $f(x)$ is monic (having a leading coefficient of 1), we multiply the polynomial by $2^{-m} \bmod N$. Thus, we define

$$f(x) = x + (2^{-m} \bmod N) \cdot \tilde{p} \tag{1}$$

and apply Coppersmith's method to find the small root x_0 of $f(x)$ (mod p). Finally, we compute $\gcd(f(x_0), N)$ which gives us the value of p.

Customizing Coppersmith

The intuition behind using Coppersmith's method is that it can be used to test when a partial assignment can be extended to a complete assignment without requiring the SAT solver to actually search for the extension itself. We call Coppersmith's method from within the SAT solver whenever the solver's current partial assignment has assigned values to more than 60% of the high or low bits of the first prime factor. Even though Coppersmith's method can be used when $50\% + \epsilon$ of the high or low bits of p are known, as ϵ decreases the required lattice dimension increases which slows down lattice reduction. Thus, we use Coppersmith's method when more than 60% bits are known in order to limit the overhead from the lattice reduction step.

In our implementation, the primary tests were conducted using a lattice with a dimension of 5. To evaluate the impact of lattice dimension on performance, additional tests were performed on lattices of varying sizes. The results of these tests are detailed in Section 5.3.5. Henceforth, unless explicitly stated otherwise, all explanations and discussions will assume a lattice dimension of 5.

The polynomials used to form the lattice are

$$N^2, N f(x), f(x)^2, x f(x)^2 \text{ and } x^2 f(x)^2 \tag{2}$$

where $f(x) = x + (2^{-m} \bmod N) \cdot \tilde{p}$ as explained in (1). The coefficients of the above polynomials form the lattice basis

$$B = \begin{bmatrix} N^2 & & & & \\ Np_1 & NX & & & \\ p_1^2 & 2p_1 X & X^2 & & \\ & p_1^2 X & 2p_1 X^2 & X^3 & \\ & & p_1^2 X^2 & 2p_1 X^3 & X^4 \end{bmatrix}$$

where $p_1 = (2^{-m} \mod N) \cdot \tilde{p}$. Note it is easy to calculate the determinant of this lattice, since the basis matrix is a lower triangular matrix.

Following [68, 70], a lattice of dimension 5 is sufficient to recover the unknown bits when slightly more than 60% of the high or low bits are known of one of the factors. Below we state the theorem for recovering unknown high bits from known low bits, as that was the case we found most effective in our experiments. Before presenting the theorem, we introduce some notation and a lemma of Howgrave-Graham. Let $G(x) = \sum_{i=0}^{d} a_i x^i$ be a univariate polynomial with coefficient vector $(a_0, a_1, \ldots, a_d)$. When the polynomial is evaluated at xX, its coefficient vector becomes $v = (a_0, a_1 X, \ldots, a_d X^d)$, and we denote the Euclidean norm of v as $\|G(xX)\|$.

Lemma 4. *Let X and M be positive integers. Suppose that:*

1. *$G(x_0) \equiv 0 \pmod{M}$ where $|x_0| \leq X$*

2. *$\|G(xX)\| < \frac{M}{\sqrt{d}}$*

Then $G(x_0) = 0$ holds over the integers.

Proof. Refer to the result by Howgrave-Graham [47]. $\qquad \square$

Theorem 5. *Let $N = p \cdot q$ where p and q are k bits each. Suppose $\tilde{p} \in \mathbb{N}$ represents the m low bits of p where $m \geq k - (\frac{1}{5} \log_2 N - 2)$. Then the lattice of dimension of 5 formed by the polynomials in (2) using $X = N^{1/5}/4$ is sufficient to factor N in time polynomial in $(\log N)$ given N and $\tilde{p}$.*

Proof. We can write $f(x)$ as defined in Equation 1 so that x_0, the integer denoting the $k - m$ high bits of p, is a root of f modulo p. First, we need to show that x_0 satisfies $|x_0| < X$. Given that $\tilde{p}$ represents the m low bits of p, we have

$$p = \tilde{p} + 2^m \cdot x_0.$$

Since $p - \tilde{p}$ is a k-bit integer, it follows that

$$0 \leq p - \tilde{p} < 2^k.$$

Dividing by 2^m, we get

$$x_0 = \frac{p - \tilde{p}}{2^m} < 2^{k-m}.$$

Given $m \geq k - \left(\frac{1}{5} \log_2 N - 2\right)$, it follows that

$$k - m \leq \frac{1}{5} \log_2 N - 2.$$

Exponentiating both sides, we get

$$2^{k-m} \leq 2^{\frac{1}{5} \log_2 N - 2} = \frac{N^{1/5}}{4},$$

and thus

$$x_0 = \frac{p - \tilde{p}}{2^m} < \frac{N^{1/5}}{4} = X.$$

Next, we form the lattice $\mathcal{L}$ using the polynomials defined in Equation 2 with $X = \frac{N^{1/5}}{4}$. For a lattice of dimension 5, the determinant is

$$\det(\mathcal{L}) = N^3 \cdot X^{10}.$$

We perform lattice reduction using the LLL algorithm. From Lemma 3, if v is the first vector in the LLL reduced basis, then

$$\|v\| \leq 2 \det(\mathcal{L})^{1/5} = 2N^{3/5} X^2 = \frac{N}{8}.$$

We now apply Lemma 4 with $M = p^2$. By construction, every vector in the lattice (including v) corresponds to a polynomial $G(x)$ with $G(x_0) \equiv 0 \pmod{p^2}$, so to apply Lemma 4, we need

$$\|v\| < \frac{p^2}{\sqrt{5}}.$$

Since p has k bits and N has at most $2k$ bits, we know $N < 2p^2$, and it follows that

$$\|v\| \leq \frac{1}{8} N < \frac{p^2}{4} < \frac{p^2}{\sqrt{5}},$$

and Lemma 4 implies that x_0 is a root of the polynomial associated to v (over the integers, not just modulo p^2). Thus, x_0 can be recovered by finding the integer roots of the polynomial associated to v. □

A more generalized version of this theorem that works for different lattices sizes is given below. It implies that an RSA modulus N can be factored in polynomial time when slightly more than 50% of the low bits of one of the prime factors of N are known. As the lattice size grows towards infinity, the percentage of known bits required to factor N asymptotically approaches 50%.

Theorem 6. *Let $N = p \cdot q$ where p and q are k bits each. Suppose $\tilde{p} \in \mathbb{N}$ represents the m low bits of p where $m \geq k - (\frac{h}{4h+2} \log_2 N - 2)$ for some constant $h \in \mathbb{N}$. Then a lattice of dimension of $2h + 1$ and $X = N^{h/(4h+2)}/4$ is sufficient to factor N in time polynomial in $(\log N)$ given N and $\tilde{p}$.*

Proof. The proof follows the same line of thought as that of Theorem 5 with some generalizations.

Similar to the previous proof, first we need to establish that x_0, the integer denoting the $k - m$ high bits of p, satisfies $|x_0| < X$. Following the same steps as before, we get

$$2^{k-m} \leq 2^{\frac{h}{4h+2} \log_2 N - 2} = \frac{N^{h/(4h+2)}}{4},$$

and thus

$$x_0 = \frac{p - \tilde{p}}{2^m} < \frac{N^{h/(4h+2)}}{4} = X.$$

Next, we form the lattice $\mathcal{L}$ using the polynomials

$$N^h, N^{h-1}f(x), N^{h-2}f(x)^2, \ldots, Nf(x)^{h-1}, f(x)^h, xf(x)^h, \ldots, x^h f(x)^h.$$

For a lattice of dimension $2h + 1$, the determinant is

$$\det(\mathcal{L}) = N^{h(h+1)/2} \cdot X^{h(2h+1)}.$$

We perform lattice reduction using the LLL algorithm. From Lemma 3, if v is the

first vector in the LLL reduced basis, then

$$\|v\| \leq 2^{h/2} \det(\mathcal{L})^{1/(2h+1)}$$
$$= 2^{h/2}(N^{(h+1)/(4h+2)}X)^h$$
$$= 2^{h/2-2h}N^{h/2}$$
$$= \frac{(N/2)^{h/2}}{2^h}.$$

We now apply Lemma 4 with $M = p^h$. By construction, every vector in the lattice (including v) corresponds to a polynomial $G(x)$ with $G(x_0) \equiv 0 \pmod{p^h}$. To apply Lemma 4, we need

$$\|v\| < \frac{p^h}{\sqrt{2h+1}}.$$

Since p has k bits and N has at most $2k$ bits, we know $N/2 < p^2$, and it follows that

$$\|v\| \leq \frac{(N/2)^{h/2}}{2^h} < \frac{p^h}{2^h} < \frac{p^h}{\sqrt{2h+1}},$$

since $2^h > \sqrt{2h+1}$ for all $h \geq 1$. Therefore, Lemma 4 implies that x_0 is a root of the polynomial associated with v over the integers, not just modulo p^h. Thus, x_0 can be recovered by finding the integer roots of the polynomial associated with v. $\qquad\square$

For each small integer root x_0 returned by Coppersmith's method, a validation step is executed. If $\gcd(f(x_0), N)$ is nontrivial, then the procedure concludes successfully with a factorization of N. However, in cases where no roots provide a factor of N, a "blocking clause" is added to the SAT solver's learned clause database.

4.1.2 Blocking Clauses

In our implementation, the blocking clause encodes that the combination of the low bits passed to Coppersmith's method was erroneous. This is done by stating that at least one of the bits must change from its current assigned value. For example,
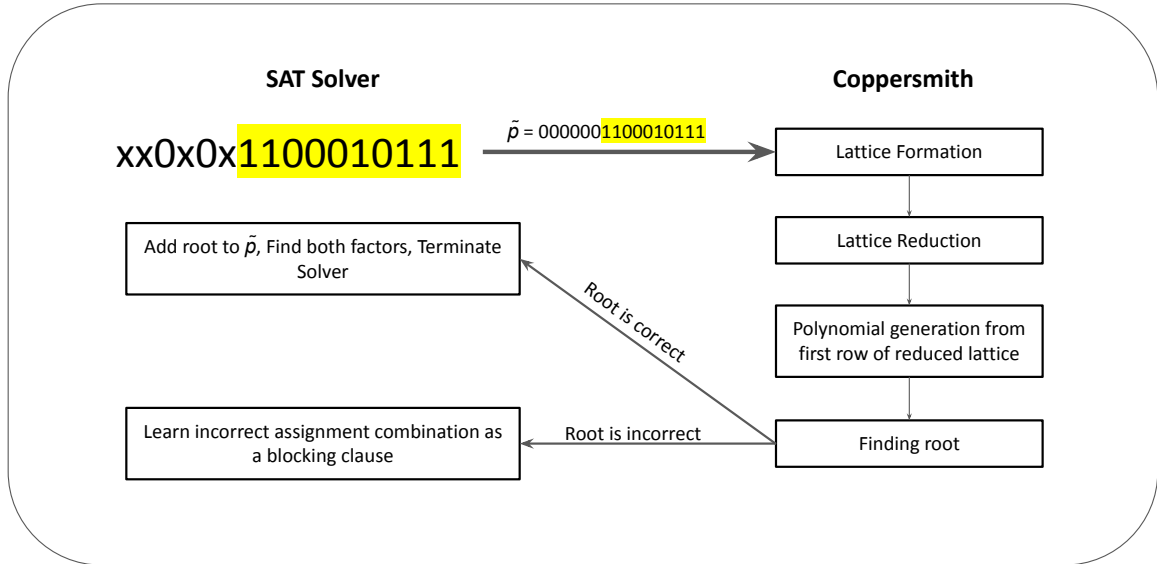
Fig. 4.1.2: A diagram outlining our SAT+CAS method for the factorization problem. Coppersmith's method is invoked whenever at least 60% of the low bits of p are assigned. If the low bits of p were set correctly, then Coppersmith's method reveals the high bits of p and the solver terminates. If the low bits were set incorrectly, then Coppersmith's method fails and a "blocking clause" is learned telling the solver to backtrack and try a new bit assignment.

suppose Coppersmith's method is applied to an 8-bit prime with the assignment

$$p = ???10011$$

and fails. Then the blocking clause passed to the solver will be

$$\neg p_4 \lor p_3 \lor p_2 \lor \neg p_1 \lor \neg p_0$$

where the bits of p (from low to high) are represented by the variables $p_0, \ldots, p_7$. The solver incorporates this knowledge as a learnt clause and immediately backtracks to explore alternative bit combinations. Figure 4.1.2 visually depicts how the technique works.

CHAPTER 5

Experiments and Results

5.1   The Encoding

Converting an instance of the factorization problem to a SAT instance is straight-forward, as multiplication circuits can be converted to SAT formulae by operating directly on the bit-representation of the integers. For example, say we are forming the instance of encoding $N = p \cdot q$ where p and q are known to be two integers of bitlength k. We represent p and q as bitvectors $[p_0, \ldots, p_{k-1}]$ and $[q_0, \ldots, q_{k-1}]$ and generate a multiplier circuit (constructed from an array of full and half adders) to compute the product of p and q. Each bit of the product can be obtained through binary addition operations. The circuit is converted into CNF by using the Tseitin Transformation described in Section 2.1.

To create our SAT instances, we employ the *CNF Generator for Factoring Problems* by Purdom and Sabry [79] using the "N-bit" adder type and the "Karatsuba" multiplier type, as we found those to be the most effective. The code for this encoder
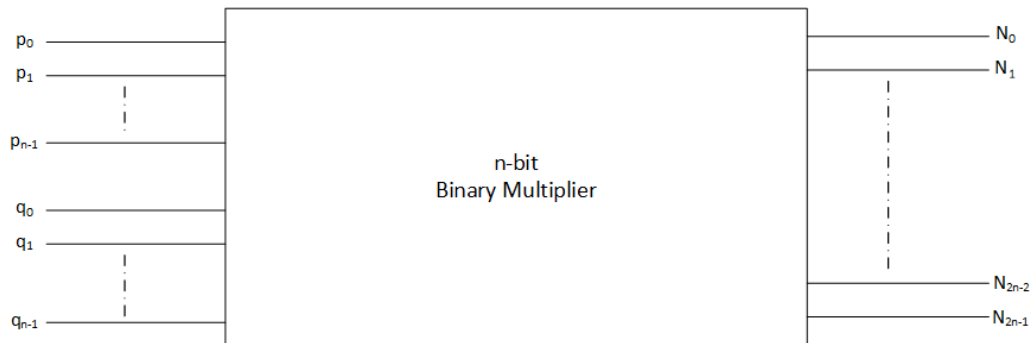

Fig. 5.1.1: A n-bit binary multiplier circuit

is written in Haskell and originally represents p and q using $2k - 1$ and k variables respectively. However, since our implementation deals with only those cases where p and q are of equal bitlength we modified the code such that p and q both have k bits. This modified version is the "balanced" encoding whereas the original code generates the "unbalanced" encoding. It is important to note that in the unbalanced encoding, the extra high bits are set to 0 using unit clauses. We have used both encodings for the purpose of comparing performance and analyzing the effect of this on the solver. The output bits of the circuit are set to match the $2k$ bits of N using $2k$ unit clauses (or in some cases N has $2k - 1$ bits). Similarly, the random known bits of p and q are also added to the SAT instance as unit clauses.

Some simple optimizations are also encoded. For example, p and q must be odd or the problem is trivial, so we fix the low bits p_0 and q_0 to true with unit clauses. Similarly, since both p and q are assumed to be of bitlength k, we fix also both high bits p_{k-1} and q_{k-1} to true. In case of the unbalanced encoding, we assign the high $k - 1$ bits of p to false since we only encoded factorization problems with p and q of equal bitlength.

5.1.1 Including Inferable Information

Based on the way the problem is set up, there are some details that can be inferred which decreases the number of unknown variables.

Apart from the simple modifications mentioned above, we can derive some extra bits using some basic number theory [43]. However, some constraints need to be satisfied in order to derive this extra information. The idea is that a bit i in one of the primes, lets say p, can be derived if –

1. Bit i in the second prime (q in this case) is known.

2. Bits 0 to $i - 1$ are known in both primes p and q.

The idea is to derive p_i from the congruence

$$p_i + q_i \equiv (N - p'q')[i] \pmod 2$$

where $p' = p_{i-1} \ldots p_0$, $q' = q_{i-1} \ldots q_0$ and $(N - p'q')[i]$ denotes the ith bit of $N - p'q'$ as described in [43]. To get a clearer idea of how this would work, let's consider an example. Let

$$p = \text{1x0xxx011}$$

$$q = \text{xx1x01011}$$

$$N = \text{100001110111000001}$$

Now, we can derive p_3 since q_3 is known and bits $p_0 \ldots p_2$ and $q_0 \ldots q_2$ are known. Following the method described above we get $p' = (011)_2 = (3)_{10}$ and $q' = (011)_2 = (3)_{10}$ giving us $p'q' = (1001)_2 = (9)_{10}$. Thus, $N - p'q' = (100001110110111000)_2$. Since we are trying to find the bit at position $i = 3$ the congruence becomes

$$p_3 + q_3 \equiv (N - p'q')[3] \pmod 2$$

$$p_3 + 1 \equiv 1 \pmod 2.$$

For the congruence to hold, p_3 must be 0. Similarly, after setting $p_3 = 0$, we can also derive p_4 as all the constraints for derivation of p_4 are satisfied. Follow the same steps, we can derive $p_4 = 0$. No other bits in either p or q can derived as the constraints will not be satisfied.

5.1.2 Incorporating the RSA Private Exponent

In addition to the encoding already described, we also considered a more constrained case of the factorization problem, namely, the problem of factoring an RSA modulus N with a public exponent of $e = 3$ (implying that both $p - 1$ and $q - 1$ are not divisible by 3). The basics of RSA tells us that

$$ed \equiv 1 \mod \phi(N), \text{ or}$$

$$ed = 1 + k\phi(N)$$

where d is the decryption exponent. We try to extend the work in [75] to our new method. This allows us to set the value of $k = 2$ and $e = 3$ in the above equation. Thus, we get:

$$3d = 1 + 2\phi(N)$$

Using the definitions $\phi(N) = (p-1)(q-1)$ and $N = pq$, the above equation can be rewritten as:

$$3d + 2(p+q) = 2N + 3 \tag{1}$$

Moreover, we can approximate d by $\tilde{d} = \lfloor (2N+3)/3 \rfloor$ because $p + q$ is relatively small compared to N. Indeed, if $p \geq q$ and both factors have k bits then $q \leq \sqrt{N}$ and $p < 2\sqrt{N}$, so $p + q < 3\sqrt{N}$. In fact, as pointed out by Boneh et al. [13], one can derive

$$0 \leq \tilde{d} - d < 3\sqrt{N}, \tag{2}$$

and they remark

"It follows that $\tilde{d}$ matches d on the $n/2$ most significant bits of d."

Similarly, Heninger and Shacham [43] remark that $\tilde{d}$ *"agrees with d on their $\lfloor n/2 \rfloor - 2$ most significant bits"*.[1] Surprisingly, both claims are false as adding even a small difference $\tilde{d} - d < 3\sqrt{N}$ to d can in rare cases cause a cascade of carries changing some bits well into in the upper-half of d. For example, when $N = 827 \cdot 953$, one has $d = 2^{19} - 53$ and $\tilde{d} = 2^{19} + 1133$ which share *no* high bits (as bitstrings of length 20). We noticed this oversight when we attempted to set the high bits of d to match the high bits of $\tilde{d}$ (computed directly from N) and in some cases the resulting instances were shown to be unsatisfiable by the SAT solver. We resolved this oversight using Lemma 7 below. First, we give a slightly improved version of (2).

Lemma 7. *Let $N = pq$ be an n-bit RSA modulus where p and q have the same bitlength, suppose d is the decryption exponent for encryption exponent $e = 3$, and set $\tilde{d} = \lfloor 2N/3 + 1 \rfloor$. Then*

---

[1]In both of these quotes n denotes the bitlength of N.

(a) $0 \leq \tilde{d} - d < \sqrt{2N}$.

(b) *Write $\tilde{d}$ and $\tilde{d} - \lfloor \sqrt{2N} \rfloor$ as bitstrings of length n, and suppose the upper l bits of the bitstrings match. Then the upper l bits of d's bitstring of length n match those of $\tilde{d}$.*

Proof. Without loss of generality suppose $q \leq p < 2q$, so that $pq < 2q^2$ (i.e., $q > \sqrt{N/2}$) and $q^2 \leq pq$ (i.e., $q \leq \sqrt{N}$). Then $p + q = N/q + q$ and $f(q) := N/q + q$ is monotonically decreasing over $q \in (\sqrt{N/2}, \sqrt{N}]$, so $p + q < f(\sqrt{N/2}) = 3\sqrt{2N}/2$. Using (1) we have

$$0 \leq \tilde{d} - d \leq 2(p+q)/3 < 2f(\sqrt{N/2})/3 = \sqrt{2N}$$

which is the inequality in (a).

The inequality in (a) is equivalent to $d \in (\tilde{d} - \sqrt{2N}, \tilde{d}]$. By assumption, the bitstrings of the lowest and highest integers in this range have n bits and share the same l high bits. The only way this can happen is if *all* bitstrings of integers in this range all share the same l high bits, including d. Otherwise, if we want the high bit (i.e., the bit of index $n-1$) to match in the lowest and highest integers but *not* with some integer in the range we would need the range to contain at least 2^n integers which it does not. $\square$

Equation (1) can be encoded in SAT using a binary adder on the terms of the left-hand side, reusing the variables for the bits of p and q and introducing new variables for the bits of d. The output bits of the binary adder are then set to the binary representation of $2N + 3$. The upper bits of d are fixed to those of $\tilde{d}$ using unit clauses (with the number of bits fixed determined by Lemma 7).

5.2   Solving Method

Each experimental iteration commences with the generation of an appropriately sized modulus N using a SageMath script. The modulus is then passed as input to the CNF

Generator, which in turn generates the requisite CNF and delivers it in the DIMACS SAT file format. To this file, we append the unit clauses that specify known bits of p and q. The percentage of known bits is fixed and given as an argument to the script. However, which specific bits of the primes are set is selected uniformly at random. There is also an option to encode the information of d assuming N is a low public exponent modulus (i.e., its prime factors are not congruent to 1 mod 3) using the encoding described in Section 5.1.2. When this option is selected, random bits of d are also leaked in the same proportion as that of p and q.

The instances were solved using a programmatic version of MapleSAT [61] available as a part of the MathCheck project [16]. We also used the very recent programmatic version of CaDiCaL solver [11, 32]. The version of Coppersmith's algorithm used is a custom implementation in C++. The GMP library [38] was used to form the lattice and it was reduced using the fplll library [87]. The formation of the polynomial from the reduced basis and its factorization is done using FLINT [42]. The modification process of the CNF instance is carried out by Python scripts.

We tested our method on random semiprime factorization problems where $p - 1$ and $q - 1$ are not divisible by 3 both with and without the inclusion of the private exponent d described in Section 5.1.2. We generated 15 random keys of the appropriate size for each problem type and ran the solvers on the SAT instance produced from each key.

5.2.1 Branching Heuristics

The implementation employs the default branching strategies integrated into MapleSAT and CaDiCaL. These strategies dictate the selection of variables upon which to branch during the search for a satisfying assignment. MapleSAT and CaDiCaL utilize sophisticated heuristics to guide the branching process efficiently.

Additionally, there is an option to use a custom branching heuristic where the branching decisions are influenced by constraints proposed by Heninger and Shacham [43], as elucidated in the Section 5.1.1. These constraints play a crucial role in shaping the search space and directing the solver towards potential solutions by branching on

variables such that sufficient number of consecutive variables have been assigned for Coppersmith's method to be employed. By integrating default branching heuristics with Heninger and Shacham's constraints, the custom branching heuristic implementation aims to strike a balance between exploration and exploitation, facilitating the efficient recovery of RSA keys from partial information. Again, it is important to note that this is merely an option that can be selected by the user and not all experiments have been run using the custom branching heuristic.

Besides that, when using the MapleSAT solver, we also have an option to enable the "variable activity" heuristic. Essentially, the solver by default assigns some activity value to each of the variables before solving starts. The solver uses these activity values to determine which variable to branch on. When this heuristic is enabled, we assign high activity values to the variables corresponding to low bits of the first prime. This makes sure that the solver prioritizes branching on these variables eventually helping us apply Coppersmith's method sooner.

5.2.2 System Requirements & Configuration

All experimentation took place on Compute Canada's Cedar and Graham clusters with each instance solved on a single CPU core allocated 4 GiB of RAM. The CPUs used are Intel Xeon E5-2683 v4 Broadwell processors.

5.3 Summary of Results

The first set of experiments fixes the number of known bits of p and q, and increases the bitlength of N until the instances become too hard to solve. The tests use the balanced encoding and have branching heuristics enabled. In case of the MapleSAT solver, the variable activity heuristic is also enabled. As indicated in Figure 5.3.1(a), a 112-bit N with 25% leaked bits takes a pure SAT approach a median of 17,969 seconds to factor (for both MapleSAT and CaDiCaL), while the SAT+CAS approach factors it in a median of 243 and 90 seconds using MapleSAT and CaDiCaL respectively. In these instances, Coppersmith was called a median of 185k times in MapleSAT and 89k
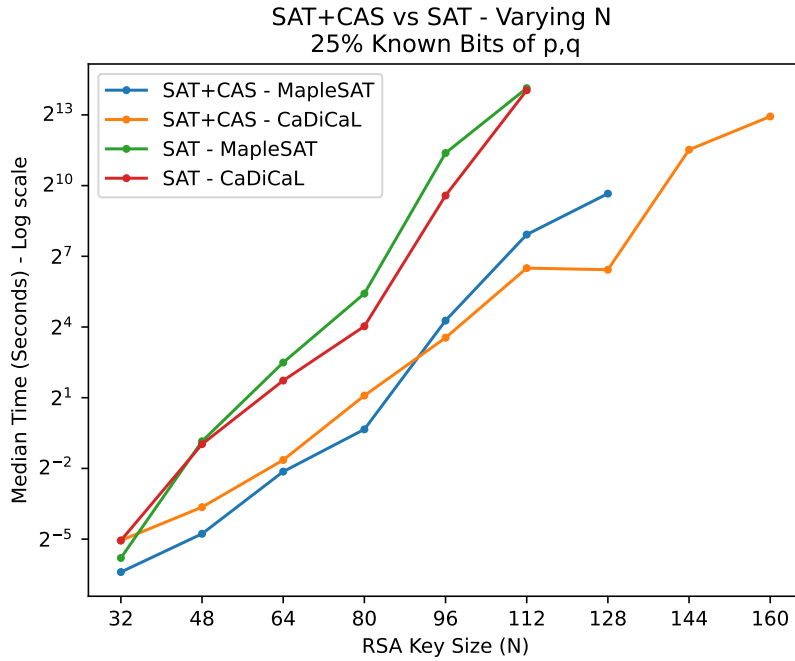
(a)
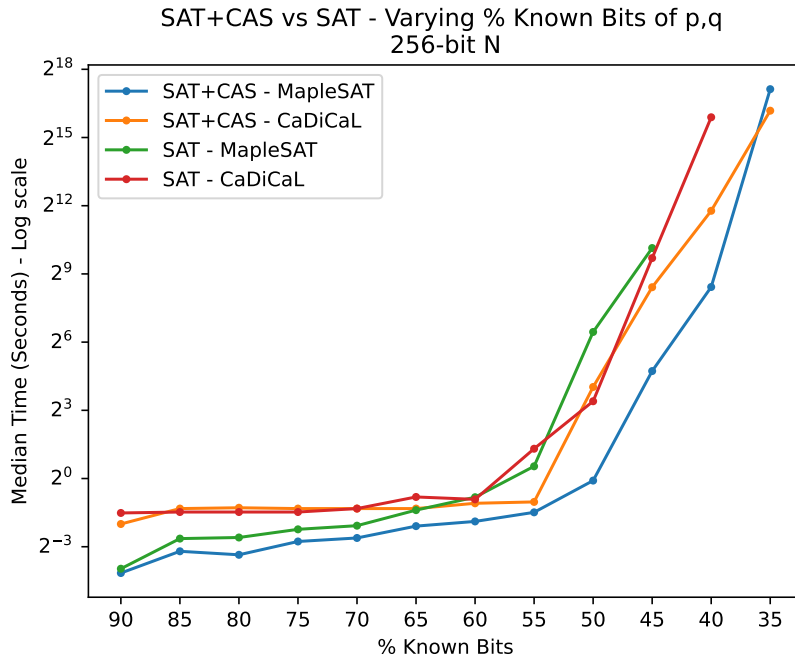
(b)

Fig. 5.3.1: The two plots shown above compare the median running time across instances with random known bits of p and q. In plot (a) the size of N varies, while the percentage of known bits varies in plot (b). All instances were run with a timeout of 2 days, so the lack of a point on the graph indicates the median time was over 2 days. All plots are given on a logarithmic scale with base 2.

49

times in CaDiCaL. Each call to the CAS interface took around 0.003 seconds. While the SAT method alone timed out for all bitsizes greater than 112, the SAT+CAS method could factor up to 160-bit N in the allotted time using the CaDiCaL solver.

The next set of experiments fixes the size of N to 256 bits and varies the percentage of known bits of p and q. Similar to the previous set of test, the branching heuristics (and variable activity heuristic in case of MapleSAT) are used along with the balanced encoding. When a large number of bits are known (at least 50%) both the SAT and SAT+CAS approaches perform relatively well. In fact, when the percentage of known bits is higher than 60%, the simpler pure SAT approach can even outperform the more involved SAT+CAS approach. However, the SAT+CAS approach clearly scales better. For example, in Figure 5.3.1(b), with 45% leaked bits, the pure SAT solver MapleSAT finds the factors in a median of 1126 seconds, while the SAT+CAS method using the MapleSAT solver factors N in a median of 26 seconds.

For a detailed breakdown of the experimental results, please refer to the Appendix A, which includes comprehensive tables summarizing the solver performance for all testing conditions considered in this section.

5.3.1 Calling Coppersmith using High Bits

In this section, we examine the performance of our proposed method when Coppersmith's method is invoked with a sufficient number of high bits of one of the primes, in contrast to our primary approach which relies on the low bits. The experimental setup involves fixing the bit size of N to 256 bits and varying the percentage of known bits of p and q. We compare the performance of the SAT method, SAT+CAS (Low), and SAT+CAS (High) methods using the CaDiCaL solver, with a timeout set to 2 days. The results are shown in Figure 5.3.2.

The results indicate that the SAT+CAS (High) method performs significantly worse compared to both the SAT and SAT+CAS (Low) methods. This poor performance was consistently observed across various tests, leading us to favor the SAT+CAS (Low) method in our experiments. The plots illustrate the differences in performance, highlighting the inefficiency of calling Coppersmith's method with
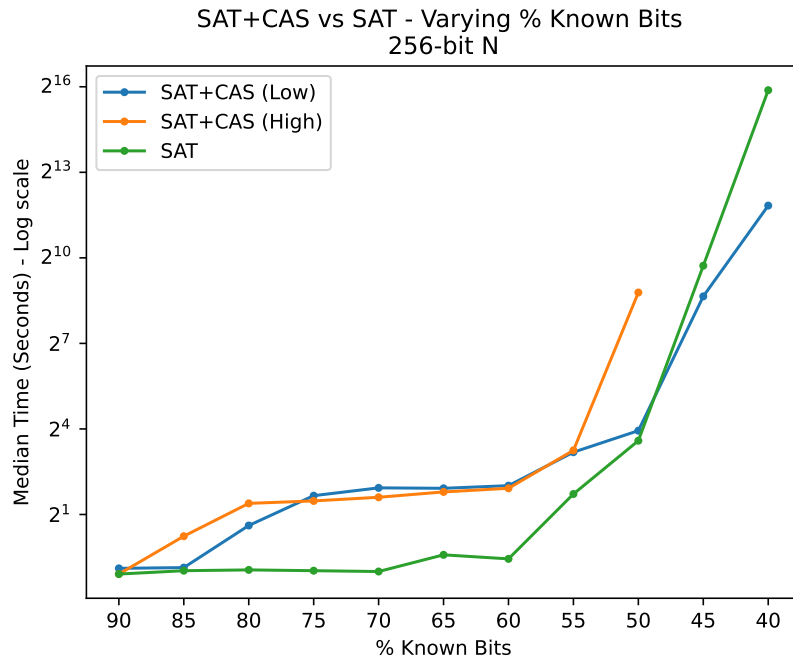
Fig. 5.3.2: The plot compares the median running time for a 256-bit N using a varying percentage of known bits of p and q. The methods compared are the pure SAT approach, SAT+CAS (Low), and SAT+CAS (High). All instances were run with a timeout of 2 days. The plots are on a logarithmic scale with base 2.

high bits of the prime.

Upon further investigation, we found that the SAT+CAS (Low) method outperforms the SAT+CAS (High) method because of the solver's ability to exploit the mathematical structure inherent in the problem. Specifically, we performed experiments to observe the variable assignments by the solver and discovered that the solver consistently adhered to the constraints corresponding to the method proposed by Heninger and Shacham [43] (see Section 5.1.1 for details). This observation suggests that the solver effectively internalizes these number theory-based constraints, which significantly enhances its performance. We suspect that this ability to leverage the inherent mathematical structure of the low bits is a major factor contributing to the performance difference between the SAT+CAS (Low) and SAT+CAS (High) methods.

5.3.2 Incorporating the Private Exponent

The encoding with partial information about d performed significantly better than the standard encoding. The results are indicated in Figure 5.3.3. Again, it uses the balanced encoding and the branching heuristics. For example, in Figure 5.3.3(a), a 192-bit N with 25% leaked bits takes a pure SAT approach a median of 40,897 seconds to factor (both MapleSAT and CaDiCaL had similar performance), while the fastest SAT+CAS approach (using CaDiCaL) factors it in a median of 51 seconds. This shows a 99.8% reduction in the running time using our SAT+CAS method.

If we include information about d for the case with 256-bit N and varying the percentage of known bits, the best-performing SAT and SAT+CAS methods take 2493 seconds and 20 seconds, respectively, with 30% known bits of p, q and d shown in Figure 5.3.3(b).

5.3.3 Effect of Branching Heuristic

In this subsection, we analyze the impact of the branching heuristic on the performance of the SAT and SAT+CAS approaches using the MapleSAT solver, as shown
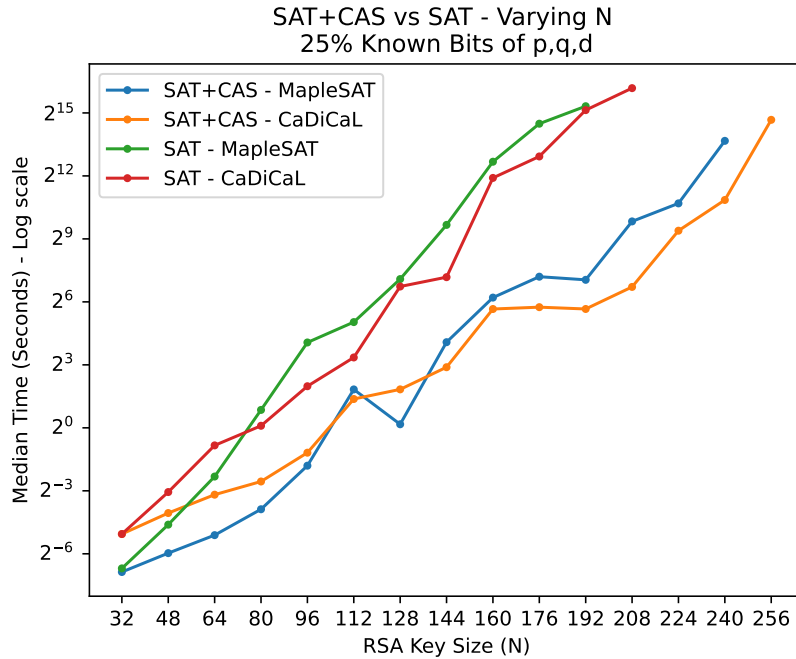
(a)
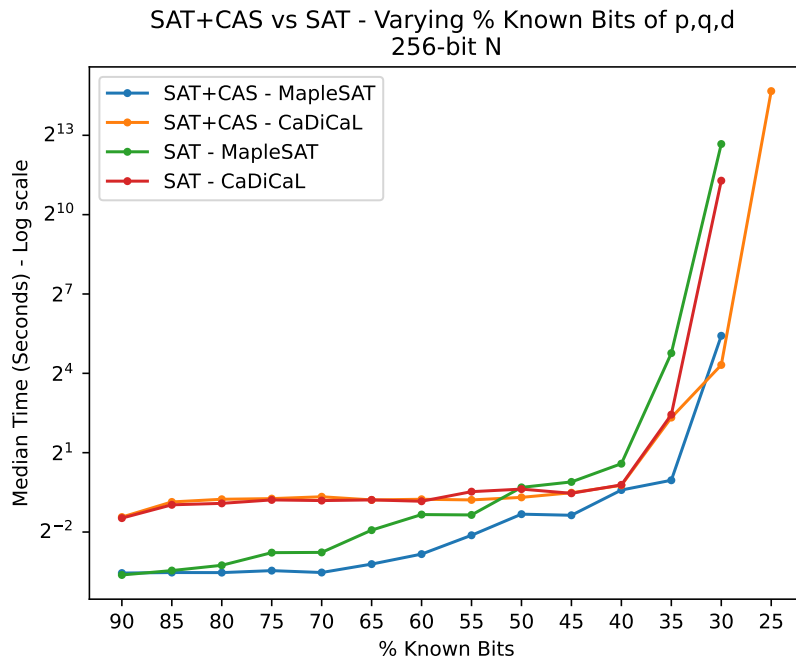
(b)

Fig. 5.3.3: These plots correspond to the low exponent encoding with $e = 3$ and also incorporate randomly known bits of d. Plot (a) compares the median running time for varying bitsizes of N using 25% of known bits of p, q and d. Plot (b), however, varies the percentage of known bits. All instances were run with a timeout of 2 days, so the lack of a point on the graph indicates the median time was over 2 days. All plots are given on a logarithmic scale with base 2.
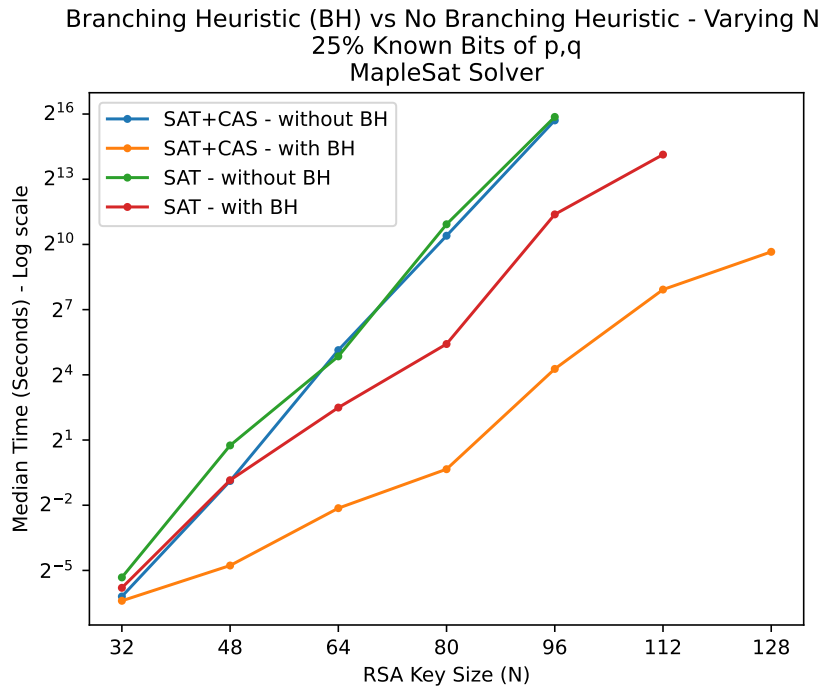
Fig. 5.3.4: This plot compares the median running time for factoring RSA keys of varying sizes using the MapleSAT solver with 25% of the bits of p and q known. The comparison is made between the SAT and SAT+CAS approaches with and without the use of a branching heuristic. The timings are presented on a logarithmic scale with base 2. The timeout was set to 2 days.

in Figure 5.3.4. The balanced encoding was used and variable activity heuristic was turned on. The experiments were conducted with 25% of the bits of p and q known, and the RSA key size N varied to observe the performance trends.

The results indicate that the inclusion of a branching heuristic significantly improves the median time to factorize RSA keys across various key sizes. For the SAT-only approach, using the branching heuristic reduces the median solving time by orders of magnitude, especially noticeable for larger key sizes. Without the branching heuristic, the SAT method alone times out for key sizes greater than 96 bits, whereas with the heuristic, it manages to solve up to 128-bit keys within the allotted time.

Similarly, the SAT+CAS approach also benefits from the branching heuristic. For example, the median time for factoring a 112-bit key is reduced from approximately 2^{14} seconds without the heuristic to 2^8 seconds with it. For larger key sizes, the heuristic allows the SAT+CAS method to remain effective, solving 128-bit keys significantly faster than without the heuristic.

The improvements are evident not only in the reduced solving times but also in the extended range of key sizes that can be feasibly factorized within practical time limits.

5.3.4   Effect of Different Encodings

The experiment investigates the effect of different encodings (balanced vs unbalanced) on the performance of the SAT+CAS and SAT methods. The MapleSAT solver was used for the comparison with the branching heuristics and variable activity heuristic turned on. Figure 5.3.5 shows the median time (seconds) to factor an RSA key with 25% known bits of p and q for varying key sizes using the MapleSAT solver.

As can be seen from the plot, the SAT+CAS solver with a balanced encoding outperforms the SAT+CAS solver with an unbalanced encoding for all key sizes. For example, with a balanced encoding the solver could factor up to a 128-bit N, whereas it was only able to go up to 112-bit N with an unbalanced encoding. The SAT solver also performs better with a balanced encoding compared to an unbalanced encoding following a trend similar to that of SAT+CAS.
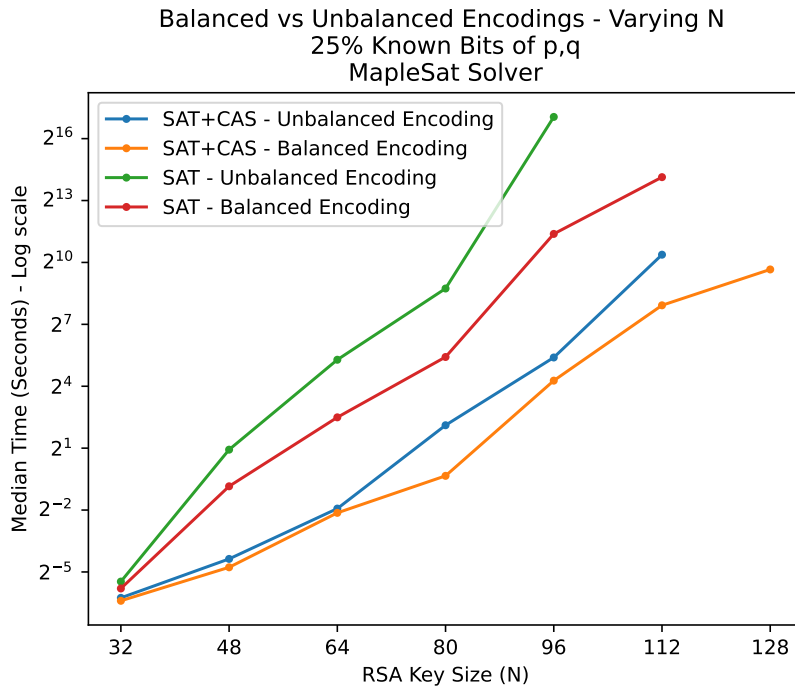
Fig. 5.3.5: This plot compares the median time (seconds) to factor an RSA key with 25% known bits of p and q for varying key sizes using the MapleSAT solver. It investigates the impact of encoding (balanced vs unbalanced) on the performance of both the SAT and SAT+CAS solvers. The timings are presented on a logarithmic scale (base 2). The timeout was set to 2 days.
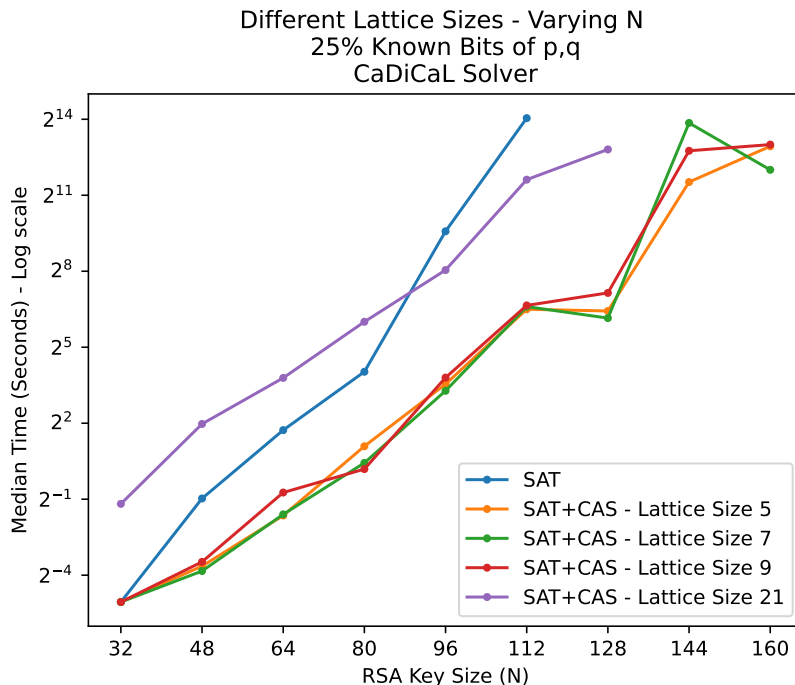
Fig. 5.3.6: This plot compares the median time (seconds) to factor an RSA key with 25% known bits of p and q for varying key sizes using the CaDiCaL solver. It investigates the impact of different lattice sizes on the performance of the SAT and SAT+CAS approaches. The timings are presented on a logarithmic scale. The timeout was set to 1 day.

These results suggest that using a balanced encoding can improve the performance of both the SAT+CAS and SAT solvers for factoring RSA keys with known bits.

5.3.5 Effect of Changing Lattice Size

This section explores the impact of varying the lattice size used within the SAT+CAS solver. We experimented with four different lattice sizes: 5, 7, 9, and 21. The CaDiCaL solver was used with the balanced encoding and branching heuristic turned on for all lattice sizes. The timings are shown in Figure 5.3.6. The results reveal an interesting interplay between lattice size, the number of known bits, and Coppersmith's method calls.

Our findings indicate that as the lattice size grows, the SAT+CAS solver utilizes Coppersmith's method more frequently. This is because the solver can leverage the additional lattice structure to find solutions with fewer known bits. However, it's

important to note that LLL-reduction, a crucial step within Coppersmith's method, becomes computationally more expensive with larger lattices. This translates to each call to Coppersmith's method taking more time as the lattice size increases. Despite the increased complexity of LLL, there's a possibility that larger lattice sizes might offer better scalability for factoring larger RSA keys (N). This warrants further investigation, but intuitively, a larger lattice might provide more efficient ways to exploit the structure of the problem as the key size grows.

In conclusion, the choice of lattice size involves a trade-off. While larger sizes enable the solver to utilize Coppersmith's method more often with fewer known bits, they also introduce computational overhead due to the complexity of LLL-reduction. The optimal size may depend on the specific key size (N) and the desired balance between the number of known bits and overall runtime.

5.3.6 Effect of Known Bits in One Prime Only

The experiment investigates the effect of having known bits in only one prime factor (p) on the difficulty of factoring an RSA key. The plot shown in Figure 5.3.7 compares the median time to factor an RSA key with 25% known bits in only one prime factor for varying key sizes (N) using the CaDiCaL solver. The balanced encoding was used along with the branching heuristic.

It is evident that the SAT+CAS method performs orders of magnitude better than the SAT approach. However, it seems that having randomly known bits in both primes rather than one makes it easier for the solver to find a solution. This can be seen by the fact that the SAT+CAS approach with partial information about both primes was able to factor up to 160-bit N whereas it was only able to factor up to 112-bit N when given partial information about only one prime.

5.3.7 Comparison with Other Works

We compared the memory usage of our approach with the method of Heninger–Shacham [43]. We ran their publicly available code and compared the results with
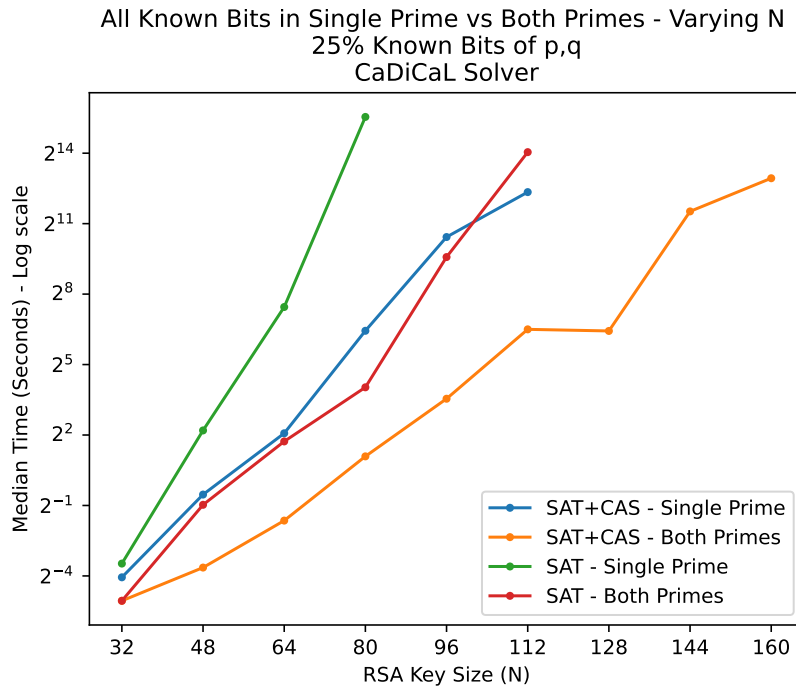
Fig. 5.3.7: This plot compares the median time (seconds) to factor an RSA key with 25% known bits of only p vs 25% known bits of p and q for varying key sizes using the CaDiCaL solver. It investigates the impact of having known bits only in one prime compared to both. The timings are presented on a logarithmic scale. The timeout was set to 1 day.
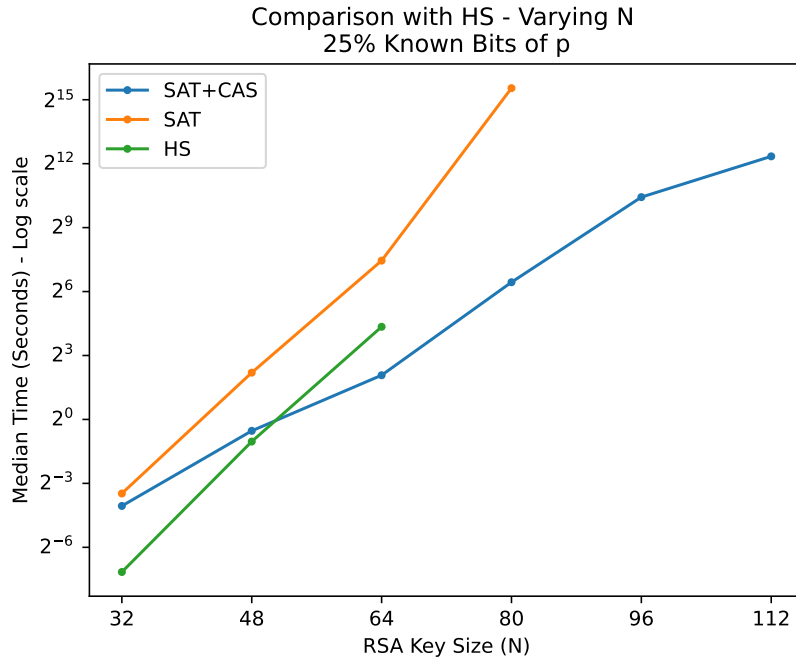
(a)
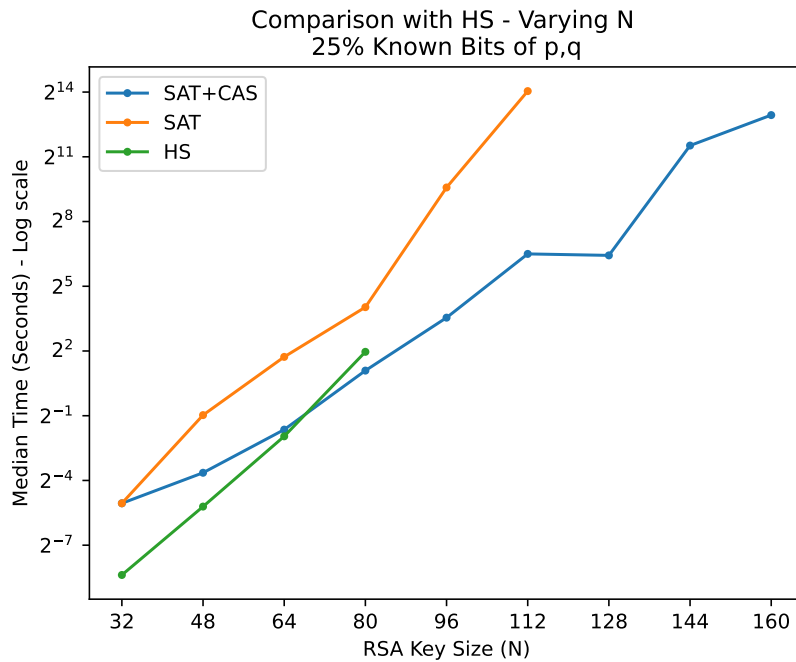
(b)

Fig. 5.3.8: This figure compares the median running time (seconds, logarithmic scale) of our method with Heninger-Shacham's (HS) method for factoring RSA keys with 25% known bits. Each subplot shows the results for a specific scenario: (a) Known bits only in p, (b) Known bits in p and q, and (c) Known bits in p, q, and d. Both methods were allocated the same resources (1 day timeout and 4GB memory). Note that a missing data point on the HS line means "Out of Memory".
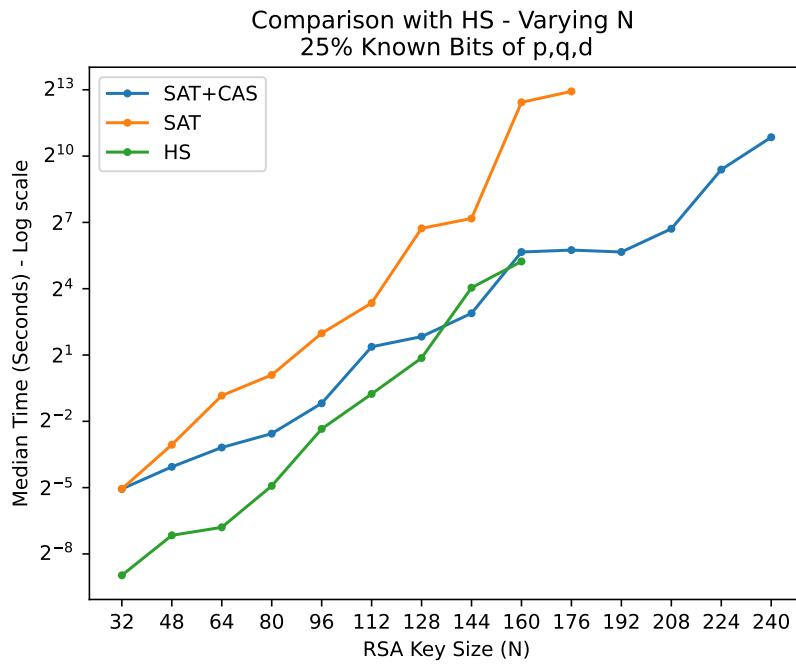
(c)

Fig. 5.3.8: This figure compares the median running time (seconds, logarithmic scale) of our method with Heninger-Shacham's (HS) method for factoring RSA keys with 25% known bits. Each subplot shows the results for a specific scenario: (a) Known bits only in p, (b) Known bits in p and q, and (c) Known bits in p, q, and d. Both methods were allocated the same resources (1 day timeout and 4GB memory). Note that a missing data point on the HS line means "Out of Memory".
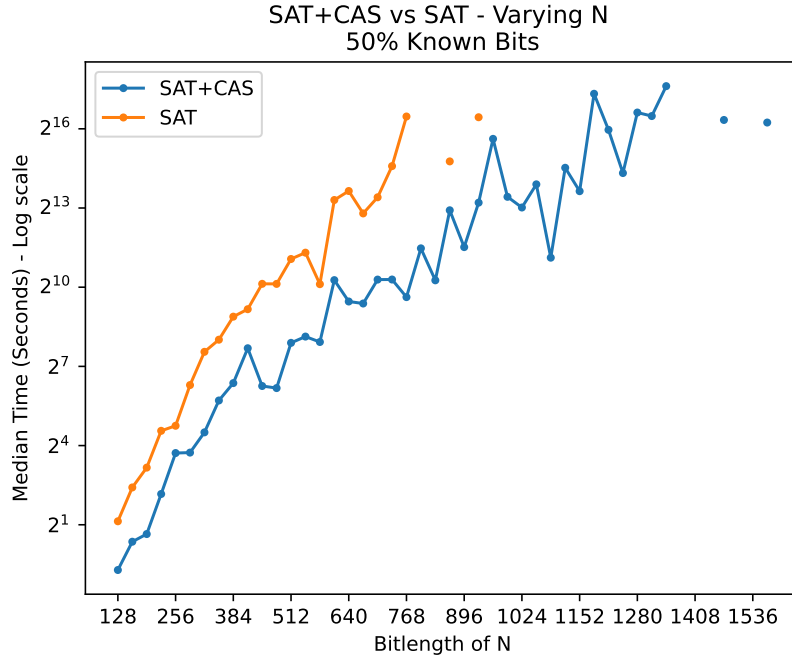
Fig. 5.3.9: This plot compares the median time (seconds) to factor an RSA key with 50% known bits of p and q for varying key sizes using the MapleSAT solver. All instances were run with a timeout of 3 days, so the lack of a point on the graph indicates the median time was over 3 days. These tests used the unbalanced encoding. The timings are presented on a logarithmic scale.

our approach on three scenarios:

- Randomly known bits only in p (see Figure 5.3.8 (a))

- Randomly known bits in both p and q (see Figure 5.3.8 (b))

- Randomly known bits in p, q, and d (see Figure 5.3.8 (c))

In all cases, we fixed the percentage of known bits at 25% and allocated the same resources (1 day timeout and 4GB memory) for both our method and the HS code.

Our experiments revealed that the HS method suffers from scalability issues in terms of memory consumption. Except for instances with smaller key sizes (N), the HS code ran out of memory for most test cases. This suggests that our approach offers a more memory-efficient solution for factoring RSA keys with known bit information.

Note that all the results in this section show that the SAT+CAS method outperforms not only a pure SAT approach, but also is also much better than using Cop-

persmith with a brute-force guessing approach. For example, with 50% leaked bits a 512-bit N can be factored by the SAT+CAS solver in a median of 237 seconds (see Figure 5.3.9), but a brute-force approach would need to determine values for around 64 unknown bits in the lower half of p before Coppersmith could be applied—much more expensive given the speed of Coppersmith. Additionally, the SAT+CAS solver will also be much more efficient than the number field sieve on the specific problem of factoring a 512-bit N with 50% leaked bits, given that factoring a 512-bit N with the number field sieve takes around 2770 CPU hours on Amazon's Elastic Compute Cloud (EC2) service [89]. Importantly, these results suggest that the SAT+CAS method has the potential to factor even larger keys, such as 1024-bit RSA keys which are commonly used in practice [8]. This makes the SAT+CAS method a highly practical tool for cryptanalysis when some key information is leaked.

These results show that the performance of off-the-shelf SAT solvers can be dramatically improved by incorporating algebraic information that—at least prior to this work—has typically only been exploited by computer algebra systems and not SAT solvers.

CHAPTER 6

Conclusion and Future Work

In this work we have demonstrated the performance of SAT solvers on integer factorization problems can be dramatically improved by calling a computer algebra system (CAS) during the solving process in order to reveal algebraic structure that is unknown to the solver. Specifically, our programmatic SAT+CAS solver calls Coppersmith's method when a significant portion of the bits of the prime factors have been assigned. Coppersmith's method is then able to efficiently (a) uncover the remaining unknown bits; or (b) tell the solver that the current bit assignment is incorrect and have the solver backtrack immediately. The latter is the typical case and our results clearly demonstrate that even with the overhead of querying a CAS the ability to backtrack early causes the solver to factor integers orders of magnitude more efficiently.

An intriguing avenue for future exploration revolves around the potential impact of parallelization on runtime efficiency. By parallelizing the algorithm, it can be possible to ascertain whether simultaneous processing can yield notable reductions in factorization time, contributing to the scalability and performance optimization of our approach.

Although there has been much recent work on adding algebraic reasoning into a SAT solver, to our knowledge this is the first work applying a SAT+CAS solver to integer factorization problems.

REFERENCES

[1] Ábrahám, E. (2015). Building Bridges between Symbolic Computation and Satisfiability Checking. In *Proceedings of the 2015 ACM on International Symposium on Symbolic and Algebraic Computation*. ACM.

[2] Ábrahám, E. (2015). Building Bridges between Symbolic Computation and Satisfiability Checking. In Yokoyama, K., Linton, S., and Robertz, D., editors, *Proceedings of the 2015 ACM on International Symposium on Symbolic and Algebraic Computation, ISSAC 2015, Bath, United Kingdom, July 6–9, 2015*, pages 1–6. ACM.

[3] Ábrahám, E. (2016). Symbolic Computation Techniques in Satisfiability Checking. In *2016 18th International Symposium on Symbolic and Numeric Algorithms for Scientific Computing (SYNASC)*, pages 3–10. IEEE.

[4] Ábrahám, E., Abbott, J., Becker, B., Bigatti, A. M., Brain, M., Buchberger, B., Cimatti, A., Davenport, J. H., England, M., Fontaine, P., Forrest, S., Griggio, A., Kroening, D., Seiler, W. M., and Sturm, T. (2017). Satisfiability checking and symbolic computation. *ACM Communications in Computer Algebra*, 50(4):145–147.

[5] Abuelyaman, E. S. and Devadoss, B. (2005). Differential Fault Analysis. In *International Conference on Internet Computing*.

[6] Ajani, Y. and Bright, C. (2023). A hybrid SAT and lattice reduction approach for integer factorization. In Ábrahám, E. and Sturm, T., editors, *Proceedings of the 8th SC-Square Workshop co-located with the 48th International Symposium on*

Symbolic and Algebraic Computation, SC-Square@ISSAC 2023, Tromsø, Norway, July 28, 2023, volume 3455 of *CEUR Workshop Proceedings*, pages 39–43. CEUR-WS.org.

[7] Bacchus, F. and Winter, J. (2004). Effective Preprocessing with Hyper-Resolution and Equality Reduction. In Giunchiglia, E. and Tacchella, A., editors, *Theory and Applications of Satisfiability Testing*, pages 341–355, Berlin, Heidelberg. Springer Berlin Heidelberg.

[8] Barker, E. (2020). Recommendation for key management: Part 1 - general.

[9] Bebel, J. and Yuen, H. (2013). Hard SAT instances based on factoring. *Proceedings of SAT Competition 2013: Solver and Benchmark Description*, page 102.

[10] Biere, A., Clarke, E. M., and Strichman, O. (2003). Bounded Model Checking. In *Bounded Model Checking*.

[11] Biere, A., Fazekas, K., Fleury, M., and Heisinger, M. (2020). CaDiCaL, Kissat, Paracooba, Plingeling and Treengeling entering the SAT Competition 2020. In Balyo, T., Froleyks, N., Heule, M., Iser, M., Järvisalo, M., and Suda, M., editors, *Proc. of SAT Competition 2020 – Solver and Benchmark Descriptions*, volume B-2020-1 of *Department of Computer Science Report Series B*, pages 51–53. University of Helsinki.

[12] Biere, A., Heule, M., van Maaren, H., and Walsh, T., editors (2021). *Handbook of Satisfiability*. IOS Press.

[13] Boneh, D., Durfee, G., and Frankel, Y. (1998). An attack on RSA given a small fraction of the private key bits. In *Advances in Cryptology — ASIACRYPT'98*, page 25–34. Springer Berlin Heidelberg.

[14] Boutros, J. J., di Pietro, N., Georghiades, C. N., and Kumar, K. P. (2016). Lattices are Good for Communication, Security, and Almost Everything.

[15] Bright, C., Djokovic, D. Z., Kotsireas, I., and Ganesh, V. (2019). A SAT+CAS approach to finding good matrices: New examples and counterexamples. *Proceedings of the AAAI Conference on Artificial Intelligence*, 33(01):1435–1442.

[16] Bright, C., Ganesh, V., Heinle, A., Kotsireas, I., Nejati, S., and Czarnecki, K. (2016). MATHCHECK2: A SAT+CAS verifier for combinatorial conjectures. In *Computer Algebra in Scientific Computing*, pages 117–133. Springer International Publishing.

[17] Bright, C., Kotsireas, I., and Ganesh, V. (2022). When satisfiability solving meets symbolic computation. *Communications of the ACM*, 65(7):64–72.

[18] Cakir, C., Bhargava, M., and Mai, K. (2012). 6T SRAM and 3T DRAM data retention and remanence characterization in 65nm bulk CMOS. In *Proceedings of the IEEE 2012 Custom Integrated Circuits Conference*, pages 1–4.

[19] Cojocaru, A. C. and Murty, M. R. (2005). *An Introduction to Sieve Methods and Their Applications*. London Mathematical Society Student Texts. Cambridge University Press.

[20] Collison, M. J. (1980). The Unique Factorization Theorem: From Euclid to Gauss. *Mathematics Magazine*, 53(2):96–100.

[21] Cook, S. and Mitchell, D. (1997). *Finding hard instances of the satisfiability problem: A survey*, page 1–17. American Mathematical Society.

[22] Coppersmith, D. (1997). Small solutions to polynomial equations, and low exponent RSA vulnerabilities. *J. Cryptology*, 10:233–260.

[23] Davenport, J. H., England, M., Griggio, A., Sturm, T., and Tinelli, C. (2020). Symbolic computation and satisfiability checking. *Journal of Symbolic Computation*, 100:1–10.

[24] Davey, B. A. and Priestley, H. A. (2002). *Introduction to Lattices and Order*. Cambridge University Press, 2 edition.

[25] Davis, M., Logemann, G., and Loveland, D. (1962). A Machine Program for Theorem-Proving. *Commun. ACM*, 5(7):394–397.

[26] Davis, M. and Putnam, H. (1960). A Computing Procedure for Quantification Theory. *J. ACM*, 7(3):201–215.

[27] de Fermat, P., Henry, C., and Tannery, P. (1894). *Oeuvres de Fermat*. Number v. 2 in Oeuvres de Fermat. Gauthier-Villars.

[28] Dixon, J. D. (1981). Asymptotically fast factorization of integers. *Mathematics of Computation*, 36:255–260.

[29] England, M. (2022). SC-Square: Overview to 2021. In Bright, C. and Davenport, J., editors, *Proceedings of the Sixth International Workshop on Satisfiability Checking and Symbolic Computation*, pages 1–6.

[30] Eriksson, J. and Höglund, J. (2014). A comparison of reductions from FACT to CNF-SAT.

[31] Eén, N. and Sörensson, N. (2004). An Extensible SAT-solver. In Giunchiglia, E. and Tacchella, A., editors, *Theory and Applications of Satisfiability Testing*, page 502–518, Berlin, Heidelberg. Springer Berlin Heidelberg.

[32] Fazekas, K., Niemetz, A., Preiner, M., Kirchweger, M., Szeider, S., and Biere, A. (2023). IPASIR-UP: User Propagators for CDCL. In *26th International Conference on Theory and Applications of Satisfiability Testing*. Schloss Dagstuhl - Leibniz-Zentrum für Informatik.

[33] Forsblom, E. and Lundén, D. (2015). Factoring integers with parallel SAT solvers.

[34] Fukshansky, L. and Hollanti, C. (2023). Euclidean lattices: theory and applications. *Communications in Mathematics*, Volume 31 (2023), Issue 2...

[35] Galbraith, S. D. (2012). *Mathematics of Public Key Cryptography*. Cambridge University Press, USA, 1st edition.

[36] Ganesh, V., O'Donnell, C. W., Soos, M., Devadas, S., Rinard, M. C., and Solar-Lezama, A. (2012). Lynx: A programmatic SAT solver for the RNA-folding problem. In *Theory and Applications of Satisfiability Testing – SAT 2012*, pages 143–156. Springer Berlin Heidelberg.

[37] Genkin, D., Shamir, A., and Tromer, E. (2013). RSA Key Extraction via Low-Bandwidth Acoustic Cryptanalysis. Cryptology ePrint Archive, Paper 2013/857. `https://eprint.iacr.org/2013/857`.

[38] Granlund, T. and the GMP development team (2012). *GNU MP: The GNU Multiple Precision Arithmetic Library*, 5.0.5 edition. `http://gmplib.org/`.

[39] Gupta, A., Ganai, M. K., and Wang, C. (2006). SAT-Based Verification Methods and Applications in Hardware Verification. In Bernardo, M. and Cimatti, A., editors, *Formal Methods for Hardware Verification*, pages 108–143, Berlin, Heidelberg. Springer Berlin Heidelberg.

[40] Halderman, J. A., Schoen, S. D., Heninger, N., Clarkson, W., Paul, W., Calandrino, J. A., Feldman, A. J., Appelbaum, J., and Felten, E. W. (2009). Lest We Remember: Cold-Boot Attacks on Encryption Keys. *Commun. ACM*, 52(5):91–98.

[41] Hamadi, Y. and Wintersteiger, C. M. (2013). Seven challenges in parallel SAT solving. *AI Magazine*, 34(2):99–106.

[42] Hart, W., Johansson, F., and Pancratz, S. (2013). FLINT: Fast Library for Number Theory. Version 2.9.0, `https://flintlib.org`.

[43] Heninger, N. and Shacham, H. (2009). Reconstructing RSA Private Keys from Random Key Bits. In Halevi, S., editor, *Advances in Cryptology - CRYPTO 2009*, pages 1–17, Berlin, Heidelberg. Springer Berlin Heidelberg.

[44] Heule, M. (2018). Schur Number Five. *Proceedings of the AAAI Conference on Artificial Intelligence*, 32(1).

[45] Heule, M. J. H., Kauers, M., and Seidl, M. (2021). New ways to multiply 3×3-matrices. *Journal of Symbolic Computation*, 104:899–916.

[46] Heule, M. J. H., Kullmann, O., and Marek, V. W. (2016). Solving and Verifying the Boolean Pythagorean Triples Problem via Cube-and-Conquer. In *Theory and Applications of Satisfiability Testing – SAT 2016*, pages 228–245. Springer International Publishing.

[47] Howgrave-Graham, N. (1997). Finding Small Roots of Univariate Modular Equations Revisited. In *IMA Conference on Cryptography and Coding*, pages 131–142.

[48] Johnson, S. (1966). Tricks for improving Kronecker's method. *Bell Laboratories Report*.

[49] Kaufmann, D. and Biere, A. (2023). Improving AMulet2 for verifying multiplier circuits using SAT solving and computer algebra. *International Journal on Software Tools for Technology Transfer*, 25(2):133–144.

[50] Kautz, H. and Neph, S. (2004). FactoringAsSat.

[51] Kautz, H. and Selman, B. (2004). Walksat version 45.

[52] Kautz, H. A. and Selman, B. (1992). Planning as Satisfiability. In *European Conference on Artificial Intelligence*.

[53] Kirchweger, M., Scheucher, M., and Szeider, S. (2023). SAT-Based Generation of Planar Graphs. In *26th International Conference on Theory and Applications of Satisfiability Testing*. Schloss Dagstuhl - Leibniz-Zentrum für Informatik.

[54] Knuth, D. E. (2022). *The Art of Computer Programming, Volume 4B, Combinatorial Algorithms, Part 2*. Addison-Wesley Professional.

[55] Kocher, P. C. (1996). Timing Attacks on Implementations of Diffie-Hellman, RSA, DSS, and Other Systems. In *Annual International Cryptology Conference*.

[56] Kocher, P. C., Jaffe, J., and Jun, B. (1999). Differential Power Analysis. In *Proceedings of the 19th Annual International Cryptology Conference on Advances in Cryptology*, CRYPTO '99, page 388–397, Berlin, Heidelberg. Springer-Verlag.

[57] Lenstra, A. K. and Lenstra, H. W., editors (1993). *The development of the number field sieve*. Springer Berlin Heidelberg.

[58] Lenstra, A. K., Lenstra, H. W., and Lovász, L. (1982). Factoring polynomials with rational coefficients. *Mathematische Annalen*, 261(4):515–534.

[59] Lenstra, H. W. (1987). Factoring integers with elliptic curves. *Annals of Mathematics*, 126:649–673.

[60] Leon, S. J., Björck, A., and Gander, W. (2013). Gram-Schmidt orthogonalization: 100 years and more. *Numerical Linear Algebra with Applications*, 20(3):492–532.

[61] Liang, J. H., Ganesh, V., Poupart, P., and Czarnecki, K. (2016a). Learning rate based branching heuristic for SAT solvers. In *Theory and Applications of Satisfiability Testing - SAT 2016 - 19th International Conference, Bordeaux, France, July 5–8, 2016, Proceedings*, pages 123–140.

[62] Liang, J. H., Ganesh, V., Poupart, P., and Czarnecki, K. (2016b). Learning rate based branching heuristic for SAT solvers. In *Proceedings of the 19th International Conference on Theory and Applications of Satisfiability Testing (SAT 2016)*, pages 123–140. Springer.

[63] Mahzoon, A., Große, D., and Drechsler, R. (2018). Combining symbolic computer algebra and boolean satisfiability for automatic debugging and fixing of complex multipliers. In *2018 IEEE Computer Society Annual Symposium on VLSI (ISVLSI)*. IEEE.

[64] Maitra, S., Sarkar, S., and Sen Gupta, S. (2010). Factoring RSA Modulus Using Prime Reconstruction from Random Known Bits. In Bernstein, D. J. and Lange,

T., editors, *Progress in Cryptology – AFRICACRYPT 2010*, page 82–99, Berlin, Heidelberg. Springer Berlin Heidelberg.

[65] Marques Silva, J. and Sakallah, K. (1996). GRASP-A new search algorithm for satisfiability. In *Proceedings of International Conference on Computer Aided Design*, pages 220–227.

[66] Marques-Silva, J. P. and Sakallah, K. A., editors (2014). *Bounded variable logics and counting: A collection of papers presented at the International Workshop on Bounded Variable Logics and Counting*. IOS Press.

[67] Martinasek, Z., Zeman, V., and Trasy, K. (2012). Simple Electromagnetic Analysis in Cryptography. *International Journal of Advances in Telecommunications, Electrotechnics, Signals and Systems*, 1.

[68] May, A. (2021). *Lattice-Based Integer Factorisation: An Introduction to Coppersmith's Method*, page 78–105. London Mathematical Society Lecture Note Series. Cambridge University Press.

[69] Micciancio, D. (2011). *Lattice-Based Cryptography*, pages 713–715. Springer US, Boston, MA.

[70] Micheli, G. D. and Heninger, N. (2024). Survey: Recovering cryptographic keys from partial information, by example. *IACR Communications in Cryptology*, 1(1).

[71] Mosca, M. and Verschoor, S. R. (2022). Factoring semi-primes with (quantum) SAT-solvers. *Scientific Reports*, 12(1).

[72] Moskewicz, M. W., Madigan, C. F., Zhao, Y., Zhang, L., and Malik, S. (2001). Chaff: Engineering an efficient SAT solver. In *Proceedings of the 38th Annual Design Automation Conference (DAC 2001)*, pages 530–535. ACM.

[73] Mushtaq, M., Mukhtar, M. A., Lapotre, V., Bhatti, M. K., and Gogniat, G. (2020). Winter is here! a decade of cache-based side-channel attacks, detection & mitigation for RSA. *Information Systems*, 92:101524.

[74] Nguyen, P. Q. and Vallée, B., editors (2010). *The LLL Algorithm*. Springer Berlin Heidelberg.

[75] Patsakis, C. (2013). RSA private key reconstruction from random bits using SAT solvers. *IACR Cryptol. ePrint Arch.*, 2013:26.

[76] Pollard, J. M. (1974). Theorems on factorization and primality testing. *Mathematical Proceedings of the Cambridge Philosophical Society*, 76(3):521–528.

[77] Pollard, J. M. (1975). A monte carlo method for factorization. *Bit Numerical Mathematics*, 15(3):331–334.

[78] Pomerance, C. (1985). The Quadratic Sieve Factoring Algorithm. In Beth, T., Cot, N., and Ingemarsson, I., editors, *Advances in Cryptology*, pages 169–182, Berlin, Heidelberg. Springer Berlin Heidelberg.

[79] Purdom, P. and Sabry, A. (2003). CNF Generator for Factoring Problems. https://cgi.luddy.indiana.edu/ sabry/cnf.html.

[80] Ramamoorthy, A. and Jayagowri, P. (2023). The state-of-the-art Boolean Satisfiability based cryptanalysis. *Materials Today: Proceedings*, 80:2539–2545. SI:5 NANO 2021.

[81] Rivest, R. L., Shamir, A., and Adleman, L. (1978). A method for obtaining digital signatures and public-key cryptosystems. *Communications of the ACM*, 21(2):120–126.

[82] Roy, S., Farheen, T., Tajik, S., and Forte, D. (2022). Self-timed Sensors for Detecting Static Optical Side Channel Attacks. In *2022 23rd International Symposium on Quality Electronic Design (ISQED)*, pages 1–6.

[83] Savela, J., Oikarinen, E., and Järvisalo, M. (2020). Finding Periodic Apartments via Boolean Satisfiability and Orderly Generation. In *EPiC Series in Computing*. EasyChair.

[84] Schoenmackers, S. and Cavender, A. (2004). Satisfy This: An Attempt at Solving Prime Factorization using Satisfiability Solvers.

[85] Shor, P. W. (1999). Polynomial-Time Algorithms for Prime Factorization and Discrete Logarithms on a Quantum Computer. *SIAM Review*, 41(2):303–332.

[86] Skorobogatov, S. P. (2002). Low temperature data remanence in static RAM.

[87] The fplll development team (2023). fplll, a lattice reduction library, Version: 5.4.4. Available at `https://github.com/fplll/fplll`.

[88] Tseytin, G. S. (1968). *On the Complexity of Derivation in Propositional Calculus*, pages 115–125. Springer. Reprinted in `https://doi.org/10.1007/978-3-642-81955-1_28`.

[89] Valenta, L., Cohney, S., Liao, A., Fried, J., Bodduluri, S., and Heninger, N. (2017). Factoring as a Service. In *Financial Cryptography and Data Security*, pages 321–338. Springer Berlin Heidelberg.

[90] von zur Gathen, J. and Gerhard, J. (2013). *Modern Computer Algebra*. Cambridge University Press.

[91] Wikipedia contributors (2024). Euler's theorem — Wikipedia, the free encyclopedia. [Online; accessed 18-April-2024].

[92] Yang, S. (2005). Researchers recover typed text using audio recording of keystrokes.

[93] Yilek, S., Rescorla, E., Shacham, H., Enright, B., and Savage, S. (2009). When private keys are public: Results from the 2008 Debian OpenSSL vulnerability. In Feldmann, A. and Mathy, L., editors, *Proceedings of IMC 2009*, pages 15–27. ACM Press.

[94] Yuen, H. and Bebel, J. (2011). ToughSAT Generation. `https://toughsat.appspot.com/`.

[95] Zabih, R. and McAllester, D. A. (1988). A rearrangement search strategy for determining propositional satisfiability. In *AAAI Conference on Artificial Intelligence*.

[96] Zassenhaus, H. (1981). Polynomial time factoring of integral polynomials. *ACM SIGSAM Bulletin*, 15:6–7.

[97] Zhang, Y. and Lin, Z. (2022). When Good Becomes Evil: Tracking Bluetooth Low Energy Devices via Allowlist-based Side Channel and Its Countermeasure. In *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security*, CCS '22, page 3181–3194, New York, NY, USA. Association for Computing Machinery.

[98] Zhaohui, F. (2004). zChaff version 2004.5.13.

[99] Zulkoski, E., Bright, C., Heinle, A., Kotsireas, I., Czarnecki, K., and Ganesh, V. (2016). Combining SAT solvers with computer algebra systems to verify combinatorial conjectures. *Journal of Automated Reasoning*, 58(3):313–339.

[100] Zulkoski, E., Ganesh, V., and Czarnecki, K. (2015). MathCheck: A Math Assistant via a Combination of Computer Algebra Systems and SAT Solvers. In *Automated Deduction - CADE-25*, pages 607–622. Springer International Publishing.

APPENDIX A

Experimental Results

The appendix provides detailed information on the experimental results for the various cases considered in Section 5.3. It includes tables that summarize the performance of the MapleSAT and CaDiCaL solvers under different configurations and also provide some extra statistics not shown in the plots.

Bit Length	Median SAT Time (s)	Median SAT Memory (MB)	Median SAT+CAS Time (s)	Median SAT+CAS Memory (MB)	Median Coppersmith Time (s)	Median Coppersmith Count
32	0.018	42	0.012	42	0.002	12
48	0.555	42	0.037	42	0.015	94
64	5.629	43	0.228	43	0.084	488
80	42.738	48	0.789	45	0.236	1011
96	2669.540	90	19.315	46	3.704	13991
112	17969.700	185	242.630	54	54.458	185308
128	T	T	808.971	68	178.066	532152

Table A.0.1: Performance of the MapleSAT solver on factoring RSA keys with 25% known bits in both primes (p and q) and varying key sizes (N). Balanced encoding with branching and variable activity heuristics were enabled. "T" denotes a timeout after 2 days of runtime.

Bit Length	Median SAT Time (s)	Median SAT Memory (MB)	Median SAT+CAS Time (s)	Median SAT+CAS Memory (MB)	Median Coppersmith Time (s)	Median Coppersmith Count
32	0.030	8.550	0.030	9.320	0.002	9
48	0.510	11.160	0.080	10.620	0.011	69
64	3.310	15.080	0.320	13.250	0.122	721
80	16.350	21.170	2.130	17.340	0.522	2200
96	763.850	90.630	11.670	21.170	4.327	15518
112	16897.230	248.720	90.460	31.450	27.236	89487
128	T	T	86.280	38.090	30.324	91591
144	T	T	2938.870	432.580	855.027	2353774
160	T	T	7821.400	493.640	3331.140	7195157

Table A.0.2: Performance of the CaDiCaL solver on factoring RSA keys with 25% known bits in both primes (p and q) and varying key sizes (N). Balanced encoding with branching heuristics was used. "T" denotes a timeout after 2 days of runtime.

Bit Length	Median SAT Time (s)	Median SAT Memory (MB)	Median SAT+CAS Time (s)	Median SAT+CAS Memory (MB)	Median Coppersmith Time (s)	Median Coppersmith Count
32	0.010	42	0.009	42	0.001	6
48	0.041	42	0.016	42	0.003	17
64	0.200	43	0.029	43	0.007	44
80	1.806	45	0.068	44	0.011	60
96	16.750	48	0.287	45	0.047	211
112	32.825	52	3.553	47	0.401	1778
128	135.839	59	1.125	49	0.073	205
144	812.712	78	16.909	53	1.284	3293
160	6542.250	172	73.714	59	7.816	17274
176	22963.300	281	146.969	68	9.905	23743
192	40897.500	311	132.508	68	4.516	10752
208	T	T	911.934	99	47.605	101196
224	T	T	1653.890	100	137.817	268593
240	T	T	13033.200	179	669.299	1130936

Table A.0.3: Performance of the MapleSAT solver on factoring RSA keys with varying key sizes (N). The experiment uses balanced encoding with branching and variable activity heuristics enabled. "T" denotes a timeout after 2 days of runtime. Here, 25% of the bits are known in both primes (p and q) as well as the decryption exponent d.

Bit Length	Median SAT Time (s)	Median SAT Memory (MB)	Median SAT+CAS Time (s)	Median SAT+CAS Memory (MB)	Median Coppersmith Time (s)	Median Coppersmith Count
32	0.030	8.650	0.030	9.580	0.001	3
48	0.120	10.740	0.060	10.930	0.003	20
64	0.560	13.470	0.110	12.910	0.009	52
80	1.070	16.570	0.170	15.370	0.025	125
96	3.950	20.920	0.440	19.070	0.058	281
112	10.210	27.100	2.590	24.950	0.406	1816
128	106.260	51.020	3.560	28.590	0.487	1474
144	144.660	53.930	7.400	33.270	1.528	4071
160	3833.700	199.500	50.560	44.380	1.810	4651
176	7785.640	278.420	53.740	55.150	5.756	13747
192	35952.590	508.680	50.610	53.910	6.472	14785
208	74295.680	669.850	104.810	62.610	13.464	27670
224	T	T	670.900	96.110	79.500	159851
240	T	T	1849.900	235.630	133.222	256135
256	T	T	26084.470	472.380	2953.360	4869580

Table A.0.4: Performance of the CaDiCaL solver on factoring RSA keys with varying key sizes (N). The experiment uses balanced encoding with branching heuristic enabled. "T" denotes a timeout after 2 days of runtime. Here, 25% of the bits are known in both primes (p and q) as well as the decryption exponent d.

Bit Length	Median SAT Time (s)	Median SAT Memory (MB)	Median SAT+CAS Time (s)	Median SAT+CAS Memory (MB)	Median Coppersmith Time (s)	Median Coppersmith Count
90	0.064	61	0.056	61	0.001	1
85	0.160	61	0.109	61	0.002	1
80	0.166	61	0.098	60	0.001	1
75	0.212	61	0.147	60	0.002	1
70	0.237	61	0.163	59	0.001	1
65	0.382	61	0.234	60	0.001	1
60	0.566	61	0.270	60	0.002	1
55	1.452	62	0.357	60	0.002	2
50	87.213	74	0.939	60	0.007	11
45	1126.820	128	26.482	72	0.123	215
40	T	T	343.552	90	9.033	16362
35	T	T	143099.000	320	8991.890	15228850

Table A.0.5: Performance of the MapleSAT solver on factoring 256-bit RSA keys with varying percentages of known bits in both primes (p and q). Balanced encoding with branching and variable activity heuristics were enabled. "T" denotes a timeout after 2 days of runtime.

Bit Length	Median SAT Time (s)	Median SAT Memory (MB)	Median SAT+CAS Time (s)	Median SAT+CAS Memory (MB)	Median Coppersmith Time (s)	Median Coppersmith Count
90	0.350	49.300	0.250	44.920	0.002	1
85	0.360	47.240	0.400	43.120	0.002	1
80	0.360	47.240	0.410	42.860	0.002	1
75	0.360	47.240	0.400	42.860	0.001	1
70	0.400	47.760	0.400	42.600	0.002	1
65	0.570	55.310	0.400	42.700	0.002	1
60	0.530	55.690	0.470	43.170	0.002	2
55	2.480	68.410	0.490	43.380	0.002	1
50	10.550	71.910	16.300	70.720	8.739	16538
45	829.670	166.300	341.300	72.540	237.596	440711
40	60731.150	740.600	3507.440	99.020	2784.730	5212033
35	T	T	74213.950	1540.860	5878.160	9655885

Table A.0.6: Performance of the CaDiCaL solver on factoring 256-bit RSA keys with varying percentages of known bits in both primes (p and q). Balanced encoding with branching heuristic was used. "T" denotes a timeout after 2 days of runtime.

Bit Length	Median SAT Time (s)	Median SAT Memory (MB)	Median SAT+CAS Time (s)	Median SAT+CAS Memory (MB)	Median Coppersmith Time (s)	Median Coppersmith Count
90	0.081	62	0.085	62	0.002	1
85	0.091	63	0.086	63	0.001	1
80	0.105	62	0.086	62	0.002	1
75	0.146	62	0.091	61	0.001	1
70	0.147	62	0.087	61	0.002	1
65	0.263	62	0.108	61	0.002	1
60	0.395	62	0.140	61	0.002	1
55	0.391	62	0.230	61	0.002	1
50	0.805	62	0.399	61	0.002	1
45	0.929	62	0.388	61	0.002	1
40	1.499	63	0.751	61	0.002	1
35	27.158	73	0.971	62	0.003	4
30	6517.400	195	42.796	74	0.204	330

Table A.0.7: Performance of the MapleSAT solver on factoring 256-bit RSA keys with varying percentages of known bits in both primes (p and q) as well as the decryption exponent d. Balanced encoding with branching and variable activity heuristics were enabled. "T" denotes a timeout after 2 days of runtime.

Bit Length	Median SAT Time (s)	Median SAT Memory (MB)	Median SAT+CAS Time (s)	Median SAT+CAS Memory (MB)	Median Coppersmith Time (s)	Median Coppersmith Count
90	0.360	48.530	0.370	48.640	0.002	1
85	0.510	48.020	0.550	45.960	0.002	1
80	0.530	45.700	0.590	45.180	0.002	1
75	0.580	45.690	0.600	45.180	0.002	1
70	0.570	45.440	0.630	44.920	0.002	1
65	0.580	45.880	0.580	44.670	0.002	1
60	0.560	46.710	0.590	44.660	0.002	1
55	0.720	53.820	0.580	44.660	0.002	1
50	0.770	53.930	0.620	44.410	0.002	2
45	0.690	55.050	0.700	44.930	0.002	1
40	0.860	56.750	0.850	53.800	0.003	3
35	5.420	69.900	5.020	71.090	0.005	6
30	2493.880	193.370	19.930	73.750	1.461	2690
25	T	T	26084.470	472.380	2953.360	4869580

Table A.0.8: Performance of the CaDiCaL solver on factoring 256-bit RSA keys with varying percentages of known bits in both primes (p and q) as well as the decryption exponent d. Balanced encoding was used with branching heuristic enabled. "T" denotes a timeout after 2 days of runtime.

Bit Length	Median SAT Time (s)	Median SAT Memory (MB)	Median SAT+CAS Time (s)	Median SAT+CAS Memory (MB)	Median Coppersmith Time (s)	Median Coppersmith Count
32	0.025	42	0.014	42	0.002	9
48	1.684	42	0.542	43	0.030	170
64	28.837	47	35.180	47	0.139	800
80	1947.140	106	1351.350	91	0.314	1283
96	59767.900	273	53513.000	254	3.965	13137

Table A.0.9: As part of an experiment to evaluate the effect of the branching heuristic on solver performance, this table shows the performance of the MapleSAT solver on factoring RSA keys with varying key sizes (N) and 25% known bits in both primes (p and q). Balanced encoding is used with the variable activity heuristic enabled, but the **branching heuristic is disabled**. "T" denotes a timeout after 2 days of runtime. This table allows comparison with the scenario with the branching heuristic (see Table A.0.1) and explores its influence on the solver's performance.

Bit Length	Median SAT Time (s)	Median SAT Memory (MB)	Median SAT+CAS Time (s)	Median SAT+CAS Memory (MB)	Median Coppersmith Time (s)	Median Coppersmith Count
32	0.023	42	0.013	42	0.002	13
48	1.901	44	0.048	43	0.013	95
64	38.965	48	0.263	45	0.070	355
80	426.490	63	4.318	48	0.577	2226
96	135852.000	404	42.042	52	4.514	16277
112	T	T	1328.510	76	251.594	820389

Table A.0.10: This table explores MapleSAT solver's performance on factoring RSA keys with varying key sizes (N). The experiment uses **unbalanced encoding**, branching heuristic, and the variable activity heuristic. All keys have 25% of the bits known in both primes (p and q). "T" indicates a timeout after 2 days of runtime. Examining this table alongside Table A.0.1's results reveals the influence of encoding on the solver's performance.

Bit Length	Median SAT Time (s)	Median SAT Memory (MB)	Median SAT+CAS Time (s)	Median SAT+CAS Memory (MB)	Median Coppersmith Time (s)	Median Coppersmith Count
32	0.090	8.810	0.060	9.580	0.009	59
48	4.580	14.250	0.690	11.830	0.391	2179
64	174.930	40.080	4.210	14.410	2.627	12395
80	47773.950	387.830	86.690	18.090	61.770	221350
96	T	T	1379.530	21.610	1138.820	3735111
112	T	T	5195.330	36.170	4301.560	13039904

Table A.0.11: Performance of the CaDiCaL solver on factoring RSA keys with varying key sizes (N) and 25% percentage of known bits in only the first prime factor (p). Balanced encoding with branching heuristic were enabled. "T" denotes a timeout after 1 day of runtime. This table explores the difficulty of factoring when only partial information about one prime factor is available.

Bit Length	32	48	64	80	96
Median SAT Time (s)	0.030	0.510	3.310	16.350	763.850
Median SAT Memory (MB)	8.550	11.160	15.080	21.170	90.630
Median SAT+CAS Time (s) Lattice Size = 5	0.030	0.080	0.320	2.130	11.670
Median SAT+CAS Memory (MB) Lattice Size = 5	9.320	10.620	13.250	17.340	21.170
Median Coppersmith Time (s) Lattice Size = 5	0.002	0.011	0.122	0.522	4.327
Median Coppersmith Count Lattice Size = 5	9	69	721	2200	15518
Median SAT+CAS Time (s) Lattice Size = 7	0.030	0.070	0.330	1.350	9.710
Median SAT+CAS Memory (MB) Lattice Size = 7	9.570	10.870	12.490	16.040	20.650
Median Coppersmith Time (s) Lattice Size = 7	0.003	0.014	0.221	0.544	5.499
Median Coppersmith Count Lattice Size = 7	9	40	514	1002	7380
Median SAT+CAS Time (s) Lattice Size = 9	0.030	0.090	0.600	1.140	13.930
Median SAT+CAS Memory (MB) Lattice Size = 9	9.580	10.860	12.480	14.900	20.660
Median Coppersmith Time (s) Lattice Size = 9	0.006	0.035	0.474	0.600	11.638
Median Coppersmith Count Lattice Size = 9	8	40	459	510	6732
Median SAT+CAS Time (s) Lattice Size = 21	0.440	3.920	13.820	64.180	264.490
Median SAT+CAS Memory (MB) Lattice Size = 21	9.840	10.950	12.710	15.040	20.650
Median Coppersmith Time (s) Lattice Size = 21	0.418	3.839	13.715	61.868	263.248
Median Coppersmith Count Lattice Size = 21	8	62	220	411	2163

Table A.0.12: Effect of Lattice Size (Part A) - Performance of the CaDiCaL solver on factoring RSA keys with balanced encoding and branching heuristic. The experiment investigates the effect of varying lattice size (5, 7, 9, and 21) on solver performance. RSA keys with varying key sizes (N) and 25% percentage of known bits of p and q were considered. "T" denotes a timeout after 1 day of runtime.

Bit Length	112	128	144	160
Median SAT Time (s)	16897.230	T	T	T
Median SAT Memory (MB)	248.720	T	T	T
Median SAT+CAS Time (s) Lattice Size = 5	90.460	86.280	2938.870	7821.400
Median SAT+CAS Memory (MB) Lattice Size = 5	31.450	38.090	432.580	493.640
Median Coppersmith Time (s) Lattice Size = 5	27.236	30.324	855.027	3331.140
Median Coppersmith Count Lattice Size = 5	89487	91591	2353774	7195157
Median SAT+CAS Time (s) Lattice Size = 7	97.160	71.040	14813.470	4105.720
Median SAT+CAS Memory (MB) Lattice Size = 7	28.520	42.750	746.570	389.990
Median Coppersmith Time (s) Lattice Size = 7	57.306	50.553	3454.150	2197.670
Median Coppersmith Count Lattice Size = 7	67588	54158	3243504	2039868
Median SAT+CAS Time (s) Lattice Size = 9	100.450	141.470	6914.550	8187.760
Median SAT+CAS Memory (MB) Lattice Size = 9	25.330	37.060	428.700	192.180
Median Coppersmith Time (s) Lattice Size = 9	89.222	105.194	4837.710	6881.380
Median Coppersmith Count Lattice Size = 9	44629	49011	1926530	2546440
Median SAT+CAS Time (s) Lattice Size = 21	3137.380	7173.130	T	T
Median SAT+CAS Memory (MB) Lattice Size = 21	25.710	30.110	T	T
Median Coppersmith Time (s) Lattice Size = 21	3120.610	7143.230	T	T
Median Coppersmith Count Lattice Size = 21	24887	55089	T	T

Table A.0.13: Effect of Lattice Size (Part B) - Performance of the CaDiCaL solver on factoring RSA keys with balanced encoding and branching heuristic. The experiment investigates the effect of varying lattice size (5, 7, 9, and 21) on solver performance. RSA keys with varying key sizes (N) and 25% of known bits of p and q were considered. "T" denotes a timeout after 1 day of runtime.

Bit Length	Median SAT Time (s)	Median SAT+CAS Time (s)	Median HS Time (s)
32	0.090	0.060	0.007
48	4.580	0.690	0.487
64	174.930	4.210	20.345
80	47773.950	86.690	OoM
96	T	1379.530	OoM
112	T	5195.330	OoM

Table A.0.14: Comparison of CaDiCaL solver's performance on factoring RSA keys with varying key sizes (N) and 25% known bits of p against results from Heninger–Shacham's (HS) work. Balanced encoding and branching heuristic were used. "T" denotes a timeout after 1 day of runtime. "OoM" denotes that the instance ran out of memory (4GB).

Bit Length	Median SAT Time (s)	Median SAT+CAS Time (s)	Median HS Time (s)
32	0.030	0.030	0.003
48	0.510	0.080	0.027
64	3.310	0.320	0.258
80	16.350	2.130	3.893
96	763.850	11.670	OoM
112	16897.230	90.460	OoM
128	T	86.280	OoM
144	T	2938.870	OoM
160	T	7821.400	OoM

Table A.0.15: Comparison of CaDiCaL solver's performance on factoring RSA keys with varying key sizes (N) and 25% known bits of p and q against results from Heninger–Shacham's (HS) work. Balanced encoding and branching heuristic were used. "T" denotes a timeout after 1 day of runtime. "OoM" denotes that the instance ran out of memory (4GB).

Bit Length	Median SAT Time (s)	Median SAT+CAS Time (s)	Median HS Time (s)
32	0.030	0.030	0.002
48	0.120	0.060	0.007
64	0.560	0.110	0.009
80	1.070	0.170	0.033
96	3.950	0.440	0.197
112	10.210	2.590	0.590
128	106.260	3.560	1.820
144	144.660	7.400	16.507
160	5532.390	50.560	37.631
176	7785.640	53.740	OoM
192	T	50.610	OoM
208	T	104.810	OoM
224	T	670.900	OoM
240	T	1849.900	OoM

Table A.0.16: Comparison of CaDiCaL solver's performance on factoring RSA keys with varying key sizes (N) and 25% known bits of p, q and d against results from Heninger–Shacham's (HS) work. Balanced encoding and branching heuristic were used. "T" denotes a timeout after 1 day of runtime. "OoM" denotes that the instance ran out of memory (4GB).

VITA AUCTORIS

NAME: Yameen Ajani

PLACE OF BIRTH: Hyderabad, India

YEAR OF BIRTH: 2000

EDUCATION: Fr. Conceicao Rodrigues College of Engineering, B.E. in Computer Engineering, Mumbai, 2022

University of Windsor, M.Sc. in Computer Science, Windsor, Ontario, 2024