INTEGER FACTORISATION ALGORITHMS AND THE RSA PROBLEM

A DISSERTATION SUBMITTED TO BIRKBECK, UNIVERSITY OF LONDON
FOR THE DEGREE OF MSC IN MATHEMATICS.

By
James Stearn
Supervisor: Dr. Maura Paterson
Department of Economics, Mathematics and Statistics

31st August 2021

Contents

List of Tables

Abstract

BIRKBECK, UNIVERSITY OF LONDON

ABSTRACT OF DISSERTATION submitted by **James Stearn** and entitled **Integer Factorisation algorithms and the RSA problem**.

Date of Submission: 31st August 2021

Factorisation of large integers is a problem of considerable theoretical and practical interest. The difficulty of factorising large integers lies at the heart of the RSA cryptosystem, which is widely used to encrypt communications data. In this paper, we outline a number of factorisation algorithms, from simple methods such as trial division, to the current fastest known algorithm, the General Number Field Sieve.

In §1, we introduce the number theoretic foundations of the RSA cryptosystem and discuss the RSA problem. We show that factorising represents the best general method for attempting to break the RSA cryptosystem.

§2 discusses a variety of factorising algorithms. We first highlight the distinction between special-purpose and general-purpose algorithms. The algorithms are presented roughly in historical order, to show how ideas developed and improved over time. Example factorisations of small numbers are given for each algorithm in order to illustrate their workings. The General Number Field Sieve is discussed in detail in §3, including proofs of some of the more important results used in the algorithm.

§4 outlines the time complexities of each algorithm, introducing notation used in the analysis of algorithms. This chapter gives a quantifiable comparison of the theoretical running times of the algorithms introduced in §2 & §3. We conclude with some consideration of future prospects for factorising in §5, given possible theoretical and technological developments.

Declaration

Chapter 1

Public-key cryptography

1.1 Fundamentals of public-key cryptography

To send a message securely in a classical (symmetric) cryptosystem, two parties must first exchange some shared secret (the key) to enable encryption and decryption of the message. Key exchange represents a critical vulnerability of the classical cryptosystem: an adversary may intercept the exchange or retrieve the key *post facto*, thereby rendering the system insecure. Continually changing keys presents further practical issues of secure distribution and storage.

Public-key (asymmetric) cryptography resolves these issues by obviating the need to distribute a shared secret amongst the cryptosystem's users. In a symmetric cryptosystem, the decryption rule d_k is identical to (or easily derived from) the encryption rule e_k. [41] In a public-key cryptosystem, each user has both a *public key* e_k, used for encryption and published publicly, and a distinct *private key* d_k, used for decryption and kept secret. This allows a user of the cryptosystem to send and receive messages securely without the vulnerability inherent in an initial key exchange.

In 1976, Diffie and Hellman [14] published a viable method of public key exchange that allowed two users to generate a shared secret securely over a public channel. In contrast to the wartime cryptographers, who utilised mechanical methods to increase the complexity of their ciphers, Diffie-Hellman key exchange exploits mathematical structures, namely the multiplicative group of integers modulo a prime. In 1977, Rivest, Shamir, and Adleman published the RSA cryptosystem, [37] building on ideas in Diffie and Hellman's work.

1.2 RSA

In this section, we outline the mechanics of the RSA cryptosystem. We discuss its security and the tractability of attacking the cryptosystem via number theoretic and computational approaches.

1.2.1 The RSA Cryptosystem

Suppose Alice wishes to send a secure message to Bob. Bob first generates two large primes p and q using one of a number of primality testing algorithms.

Bob then generates a large semiprime $n = pq$ and calculates $\varphi(n)$. [37]

Definition 1.1. A **semiprime** $n \in \mathbb{N}$ is the product of exactly two (not necessarily unique) primes.

Definition 1.2. $\varphi(n)$ is **Euler's totient function**: the number of positive integers up to n coprime to n.

As n is a semiprime, $\varphi(n) = (p-1)(q-1)$, as $\varphi(p) = p-1$ for p prime and φ is a multiplicative function.

Definition 1.3. Let (a, b) represent the **greatest common divisor** of $a, b \in \mathbb{Z}$.

Definition 1.4. An arithmetic function $f(n)$ is **multiplicative** if $f(ab) = f(a)f(b)$ for all a, b with $(a, b) = 1$.

Bob then chooses an $e \in \mathbb{N}$ such that (e,$\varphi(n)$) = 1 and calculates its inverse d modulo $\varphi(n)$ using the Extended Euclidean algorithm.

Definition 1.5. The pair (n, e) is an RSA **public key**. The integer n is an **RSA modulus**.

Definition 1.6. A corresponding RSA **private key** d is calculated $d \equiv e^{-1} \pmod{\varphi(n)}$

To encrypt a message, Alice looks up Bob's public key pair and encrypts the plaintext using a one-way trapdoor function: a function that is easy to compute in one direction but difficult to invert without some additional information (the private key). [16]

Definition 1.7. RSA encryption

To generate a ciphertext C from a plaintext M, Alice computes:

$$C = M^e \text{ (mod n)}$$

Definition 1.8. RSA decryption

Bob may now decrypt C using his knowledge of the private key d:

$$M = C^d \text{ (mod n)}.$$

Given a ciphertext, we require that the function is indeed invertible given knowledge of the private key d, and hence that the plaintext may be retrieved by the recipient.

Proposition 1.9. *RSA decryption of C using private key d returns the original plaintext M.*

Proof.

$$\begin{aligned} C^d \pmod{n} &\equiv (M^e)^d \pmod{n} \\ &\equiv M^{ed} \pmod{n} \\ &\equiv M^{k\varphi(n)+1} \pmod{n} && \text{[as ed} \equiv 1 \text{ (mod } \varphi(n)\text{)]} \\ &\equiv (M^{\varphi(n)})^k \times M \pmod{n} \\ &\equiv (1)^k \times M \pmod{n} && \textit{[By Euler's Theorem]} \text{ [20]} \\ &\equiv M \pmod{n} \end{aligned}$$

□

Having demonstrated the correctness of the RSA algorithm, we now consider its security.

1.2.2 The RSA Problem

The difficulty of inverting the encryption function, i.e. the difficulty of finding e^{th} roots modulo n underlies the security of the cryptosystem.

Definition 1.10. The **RSA problem** is defined as: [16]

$$\text{Given y} \in (\mathbb{Z}/n\mathbb{Z}^*), \text{compute } x \text{ such that } x^e \equiv y \pmod{n} \tag{1.1}$$

We now outline some different approaches to this problem.

1.2.3 Attacks on RSA

Although there is no publicly known general attack on RSA, da Costa Boucinha describes a number of specific attacks requiring implementation errors during the encryption process. [11] These include repeated usage of the same modulus n and various known plaintext attacks which require some partial knowledge of the plaintext. Careful selection of primes when generating the modulus n and the use of random padding schemes defend against these types of attack. Thus, as [11] makes clear, the most promising general attack is to attempt to factorise the modulus n. Throughout this paper, we use n to represent an RSA modulus to be factored.

A successful factorisation of n allows an attacker to calculate $\varphi(n)$ and thereby decrypt all messages encrypted using the corresponding public key. As the original RSA paper outlines, *'the security of the [RSA] system rests in part on the difficulty of factoring the published divisor, n.'* [37] As the difficulty of factorising generally increases with the size of the input, by choosing larger primes to generate n, the security of the system may be (indefinitely) increased.

The computational complexity of factorising n is at least as hard as the RSA problem, as recovering p and q allows us to compute x directly using the private key d. [37] argues *'...that all the obvious approaches for breaking [the] system are at least as difficult as factoring n'*. Factorisation of large integers is therefore a problem of significant theoretical, practical, and commercial importance.

1.2.4 Security of RSA & parameter choices

In 1977, Rivest was quoted in Martin Gardner's column Mathematical Games as estimating that a 126-digit semiprime would take around *'... 40 quadrillion years'* to factorise with the technology and algorithms of the day. [17] Subsequent increases in computing power and theoretical development of faster algorithms have increased the size of numbers that may be factorised.

In 1991, RSA Laboratories issued the RSA Factoring Challenge - a list of large semiprimes intended to stimulate research into the factorisation problem. These days, factorisation of large numbers is distributed across a network with many researchers contributing optimisations and processing power to large-scale factorisation projects. Optimisation of algorithm implementations is at the forefront of modern factorisation research. The problem of factorising thus lies at the cutting-edge intersection of number theory and computer science.

Table 1.1 [36] shows some record factorisations to outline advances made in the field. Estimates for the capabilities of factorising algorithms between 1970-90 are based on [35]. These records are for general numbers; larger numbers with particular special forms have been factorised but are not presented here.[1]

Number	**Decimal digits**	**Date factorised**
	20	$\sim$1970
	50	$\sim$1980
	107	1989
RSA-120	120	1/6/1993
RSA-130	130	10/4/1996
RSA-155	155	22/8/1999
RSA-576	174	3/12/2003
RSA-768	232	12/12/2009
RSA-240	240	2/12/2019
RSA-250	250	28/2/2020

Table 1.1: Integer factorisation records

RSA-250 is a semiprime with 250 decimal digits (or 829 bits). In the original RSA paper, the authors recommended *'...using 100-digit prime numbers p and q, so that n has 200 [decimal] digits'* ($\sim$512 bits). [37] The National Institute of Standards and Technology (NIST) currently recommends the use of at least 2048-bit keys to ensure security for personal use. [2] Developments in factorising technology have been fairly consistent since the early nineties, enabling institutions to accurately assess when security standards require updating.

To summarise, provided that our chosen generating primes satisfy particular conditions (to be detailed in later chapters), the size of the modulus n is the primary determinant of the security of an RSA encryption. We now detail the development of a range of integer factorisation algorithms, highlighting the improvements and optimisations of later algorithms on earlier work.

[1]RSA-576 & RSA-768 are labelled according to their number of binary, not decimal, digits

Chapter 2

Integer factorisation algorithms

2.1 Introduction

Definition 2.1. The **Integer Factorisation Problem** is defined as: [45]

$$\text{Given } n \in \mathbb{Z}^+, \text{find a non-trivial } a \text{ such that } a|n \tag{2.1}$$

Note that a need not be prime, in which case there are a number of primality testing algorithms that may be used in combination with a factorisation algorithm to determine the unique prime factorisation of n. There are two main classes of factorisation algorithm:

Definition 2.2. A **special-purpose algorithm** has runtime depending on some special property of n or the factor a (e.g. the size of a or its proximity to $\sqrt{n}$).

Definition 2.3. A **general-purpose algorithm** has runtime depending on the size of n and independent of the size of the factors.

Although general-purpose algorithms may appear more powerful, as we can input any number and reasonably estimate the runtime based on its size, the special-purpose algorithms remain useful. For example, a large number with a small prime factor may be factorised quickly and efficiently by simply attempting division by small values for a. On the other hand, a general-purpose algorithm will take as long to factor a 100-digit number with 1-digit and 99-digit prime factors as it would to factor one with two 50-digit factors. [45] It is therefore instructive to review a range of approaches to factorising and utilise methods suitable for the given situation.

In practice, many factorisation attempts will combine a combination of special-purpose and general-purpose algorithms. This is somewhat analogous to the brute-force use of lists of common passwords when attempting to crack a password, before the application of more sophisticated techniques.

We begin with a review of selected special-purpose algorithms in §2.2, then proceed to general-purpose algorithms in §2.3.

2.2 Special-purpose algorithms

2.2.1 Trial division

A naive approach to factorising involves trial division of n by all primes up to $\sqrt{n}$. As division is a computationally intensive operation, this approach is obviously inefficient as many unnecessary divisions will be performed. However, if n happens to have a small prime factor, trial division will find it in a reasonable amount of time. ([45] suggests it should only be used for $n < 10^8$). In practice, trial division is still employed as an element in even the most sophisticated algorithms, such as the Number Field Sieve.

2.2.2 Fermat's method

Fermat introduced a method of factorisation that has directly influenced the development of modern methods. We thus review it here for historical context.

Henceforth, we may assume the integer to be factored is odd, as even integers have an obvious prime factor. We begin with a theorem of elementary number theory.

Theorem 2.4. *Every odd integer may be written as the difference of two squares.*

Proof. Let n be an odd integer. Write $n = 2k+1$ for $n, k \in \mathbb{Z}$. Squaring consecutive integers $k, k+1$ and subtracting gives: $(k+1)^2 - k^2 = 2k+1$. □

Fermat's method entails searching for $x, y \in \mathbb{Z}$ such that $n = x^2 - y^2$. Once such x & y are found, we may write $n = (x+y)(x-y)$ and a factorisation has been found. It is possible that this gives a trivial factorisation, in which case a new x, y pair must be found.

To find x & y, we calculate values of $x^2 - n$ until a square is found. Then $x^2 - n = y^2$ and n may be factored as above.

We begin with $x = \lfloor\sqrt{n}\rfloor + 1$, incrementing by 1 until the value of $x^2 - n$ is a square. [24]

Example 2.5. We shall factorise $n = 1633$ using Fermat's method.

We initialise with $x = \lfloor\sqrt{1633}\rfloor + 1 = 41$ and increment by 1.

x_i	$y_i^2 = x_i^2 - n$	y_i
41	48	6.928...
42	131	11.446...
43	216	14.697...
44	303	17.407...
45	392	19.799...
46	483	21.977...
47	576	24

Table 2.1: Values calculated using Fermat's method

Having found a square, we may write $n = 47^2 - 24^2$, then $n = (47 + 24)(47 - 24)$.
Thus, $1633 = 71 \times 23$.

Note that if the two factors are 'close' together, Fermat's method will find them quickly. This needs to be taken into account when generating an RSA modulus. In the worst case however, Fermat's method is slower than trial division.

2.2.3 Euler's method

Euler employed a similar method to Fermat to factor larger numbers. As the proof of the method outlines the derivation of values used in a factorisation, we show it here for completeness. [9]

Proposition 2.6. *Let n be an odd integer, expressible as the sum of two squares in two distinct ways: $n = a^2 + b^2 = c^2 + d^2$ for some $a, b, c, d \in \mathbb{Z}$ with $b < d$; a, c odd, and b, d even.*

Let $r = (a - c, d - b)$; $u = (a + c, d + b)$; $s = a - c/r$; $t = d - b/r$

Then n is factorised by:

$$n = \left((\frac{r}{2})^2 + (\frac{u}{2})^2\right)\left(s^2 + t^2\right) \tag{2.2}$$

Proof. Suppose n is expressible as the sum of two squares in two distinct ways.
Factorising gives: $(a - c)(a + c) = (d - b)(d + b)$
As $r = (a - c, d - b)$, then: $a - c = rs$ and $d - b = rt$ for some $s, t \in \mathbb{Z}$ with $(s, t) = 1$.

This gives $s(a+c) = t(d+b)$, so $t|(a+c)$ and $a+c = tu$ for some $u \in \mathbb{Z}$.

Then $d+b = su \implies (a+c, d+b) = u$ with u even.

Therefore:

$$\begin{aligned}\left((\frac{r}{2})^2 + (\frac{u}{2})^2\right)\left(s^2+t^2\right) &= \frac{1}{4}\left((rs)^2 + (rt)^2 + (su)^2 + (tu)^2\right)\\ &= \frac{1}{4}\left((a-c)^2 + (d-b)^2 + (d+b)^2 + (a+c)^2\right)\\ &= \frac{1}{2}\left(a^2+b^2+c^2+d^2\right)\\ &= n\end{aligned}$$

□

Example 2.7. We shall factorise $n = 4453$ using Euler's method.

By a similar process to Fermat's method, we have found that we may write 4453 as $4453 = 63^2+22^2$ and $4453 = 33^2+58^2$. We can calculate greatest common divisors efficiently using the Extended Euclidean algorithm.

$a = 63$	$a - c = 30$	$r = (30, 36) = 6$
$b = 22$	$a + c = 96$	$u = (96, 80) = 16$
$c = 33$	$d - b = 36$	$s = a - c/r = 5$
$d = 58$	$d + b = 80$	$t = d - b/r = 6$

Table 2.2: Values required for Euler's method

Taking the values from the last column and substituting into (2.2), gives:

$$n = \left((\tfrac{6}{2})^2 + (\tfrac{16}{2})^2\right)\left(5^2+6^2\right)$$

Thus, we obtain the factorisation $4453 = (3^2+8^2)(5^2+6^2) = 73 \times 61$.

Euler's method relies on being able to express n as the sum of two squares in more than one way. Moreover, to be of any use, we must be able to find these sums in an efficient manner. Unfortunately, its utility is severely limited by the following theorem:

Theorem 2.8. ***Sum of two squares theorem:*** *For $n \in \mathbb{Z}$, n is the sum of two squares if and only if all primes p of the form $p \equiv 3 (mod\ 4)$ in the prime factorisation of n have even powers .*

Proof. A proof is given in Theorem 3.13 of [3]. □

2.2.4 Pollard's rho algorithm

In 1975, John Pollard introduced a novel factorisation method in his paper *'A Monte Carlo Method for Factorization'*. Pollard's method utilises a pseudo-randomly generated sequence, hence the paper's title.

To factorise n, the method proceeds by generating a sequence with some polynomial $f \in \mathbb{Z}[x]$. [33] Pollard originally used $f(x) = x^2 - 1$.[1] An initial value $x_0 \in \mathbb{N}$ is chosen randomly such that $x < n$. The sequence is now generated recursively: [45]

$$x_i \equiv f(x_{i-1}) \pmod{n}$$

As n is finite, there are a finite number of congruence classes modulo n, so the sequence must eventually repeat and cycle.

Now consider some non-trivial factor d of n. Both the sequences x_i (mod d) and x_i (mod n) must eventually cycle, but it is likely that x_i (mod d) will cycle sooner, as $d < n$, so there are fewer congruence classes modulo d. It is therefore likely there exists some pair x_i, x_j such that:

$$\begin{aligned} x_i &\equiv x_j (\text{mod d}) \\ x_i &\not\equiv x_j (\text{mod } n) \end{aligned} \tag{2.3}$$

In this case, $(x_i - x_j, n)$ is a non-trivial factor of n as $d \mid (x_i - x_j)$ but $n \nmid (x_i - x_j)$. [45]

Of course, we don't know d in advance, so to find it, we could check the gcd of each x_i with all previous x_j values until a non-trivial factor is found $\big($if (2.3) does not hold, $(x_i - x_j, n)$ will be a trivial factor$\big)$. This becomes quite inefficient as the number of x_is grows, so Pollard used Floyd's cycle-finding algorithm to detect a cycle. This algorithm works by incrementally checking x_i and x_{2i} for a repeated value in order to identify a cycle. [45]

Floyd's cycle-finding algorithm greatly increases efficiency as it requires only that we check $(x_{2i} - x_i, n)$ for each i, until a non-trivial factor is found. When plotted as a graph, the eventual cycling of the x_i (mod d) sequence resembles the Greek letter ρ, hence the algorithm's name.

Example 2.9. We shall factorise $n = 6497$ using Pollard's rho algorithm.

We initialise the algorithm using $x_0 = 2$ and $f(x) = x^2 - 1 \pmod{6497}$.
We calculate the values of x_i for $i = 1, 2, 3...$, and calculate $(x_{2i} - x_i, 6497)$ until a non-trivial

[1][33] conjectures that all polynomials x^2±a with $a \neq 0, 2$ work equally well.

factor is found. Intermediate calculations of the algorithm are shown in the table on the next page.

One particular advantage of Pollard's rho algorithm is that it does not require much storage, as only a few x_i values need to be stored at any one time.

i	x_i	$x_i^2 - 1 \pmod{6497}$	$(x_{2i} - x_i, 6497)$
0	2	3	N/A
1	3	8	1
2	8	63	1
3	63	3968	1
4	3968	2792	73
5	2792	5360	1
6	5360	6362	1
7	6362	5230	1
8	5230	529	73

Table 2.3: Values calculated using Pollard's rho algorithm

So $(x_8 - x_4, 6497) = 73$ and the algorithm has found a non-trivial factor of 6497 (we could have stopped the calculations in the final column here).
We therefore find the factorisation $6497 = 73 \times 89$.

It is possible that the sequence modulo d and modulo n will repeat at the same x_i value, in which case the gcd will be n and the algorithm will fail to find a non-trivial factor. [45] In this case, we initialise with a different x_0 value or change our polynomial $f(x)$.

The greatest success of Pollard's rho algorithm was its use (in a slightly adapted form) in the factorisation of the 8th Fermat number $F_8 = 2^{2^8} + 1$. [4] F_8 has 78 decimal digits and was factored in 1980 by Pollard and Brent, who would go on to factor larger Fermat numbers such as F_{10}. [6] Brent's optimisation consists of a cycle-finding algorithm ~36% faster than Floyd's algorithm, leading to an overall speed up of ~24% for the entire algorithm. [5] Besides this optimisation, the overall approach is the same, so we do not describe the details here.

To summarise, we have reviewed some historically important special-purpose factorisation algorithms, and outlined the development of factorising capabilities. This review is by no means exhaustive. In particular, we have not discussed Lenstra's elliptic-curve method, which is a fast special-purpose algorithm often used to factorise n known to have small prime factors. For further details, see [29]. We now proceed to the general-purpose algorithms.

2.3 General-purpose algorithms

2.3.1 Legendre's congruence

Most general-purpose algorithms exploit an idea based upon Fermat's method, perhaps first introduced by Maurice Kraitchik. [34] Recall from §2.2.2 that in order to factorise n, we attempt to find some values x, y such that $x^2 - y^2 = n$. The closely related congruence $x^2 \equiv y^2 \pmod{n}$ is fundamental to the general-purpose algorithms. Kraitchik observed:

Proposition 2.10. *If x and y are two integers less than n, such that $x \neq y$ and $x + y \neq n$, and $x^2 \equiv y^2$ (mod n), then $(x - y, n)$ and $(x + y, n)$ are possibly non-trivial factors of n.*

The congruence above is sometimes known as Legendre's congruence. [45] To see why Proposition 2.10 holds, we tabulate the possibilities for a semiprime $n = pq$ with p, q prime, as used in RSA: [8]

p|x+y?	**p|x-y?**	**q|x+y?**	**q|x-y?**	**(x+y,n)**	**(x-y,n)**	**Non-trivial factor?**
✓	✓	✓	✓	n	n	×
×	✓	✓	✓	q	n	✓
✓	×	✓	✓	n	q	✓
✓	✓	×	✓	p	n	✓
✓	✓	✓	×	n	p	✓
✓	×	✓	×	n	1	×
×	✓	×	✓	1	n	×
✓	×	×	✓	p	q	✓
×	✓	✓	×	q	p	✓

Table 2.4: Possible factors when $x^2 \equiv y^2$ (mod n)

Thus, if we are able to find x, y such that Legendre's congruence holds, there is a probability of $\frac{2}{3}$ that we will find a non-trivial factor. Each algorithm varies in the manner in which the congruence of squares is found.

We begin ahistorically with Dixon's algorithm, which uses a number of concepts employed in more complex algorithms. In §2.3.3 , we outline one of the earliest general-purpose algorithms, the Continued Fraction (or CFRAC) method. We build upon these concepts in discussion of the Quadratic Sieve in §2.3.4, leading naturally into a review of the General Number Field Sieve in §3.

2.3.2 Dixon's algorithm

Rather than attempting to find x and y such that $x^2 \equiv y^2 \pmod{n}$, Dixon's method requires only that we find a number of *relations*:

$$x^2 \equiv y \pmod{n} \text{ where y is B-smooth.} \tag{2.4}$$

Definition 2.11. A number is **B-smooth** if all of its prime factors p_i satisfy $p_i \leq B$ for some $B \in \mathbb{N}$. [45]

Example 2.12. 10780 is 11-smooth as $10780 = 2^2 \times 5 \times 7^2 \times 11$.

11466 is not 11-smooth as $11466 = 2 \times 3^2 \times 7^2 \times 13$.

Definition 2.13. A **factor base** P_B is the set of primes $p_i \leq B$.

Dixon's method is an improvement on Fermat's, as there are more smooth numbers satisfying (2.4) than the stricter requirement $x^2 \equiv y^2 \pmod{n}$. To see how we generate a congruence of squares using these relations, consider the following example:

Example 2.14. We will factorise $n = 1537$ using the factor base $P_B = \{2, 3, 5, 7, 11\}$

The required relations are of the form: [15]

$$x_i^2 \equiv y_i \pmod{1537} \text{ where the } y_i \text{ are 11-smooth (i.e. } y_i = \prod_{p_i \in P_B} p_i^{a_i})$$

By calculating $x^2 \pmod{1537}$ for random x values and searching for 11-smooth numbers, we find the relations in Table 2.5.[2]

x	x^2 **(mod 1537)**	**Factorisation of** y_i	**Vector**
47	672	$2^5 \times 3 \times 7$	(1 1 0 1 0)
49	864	$2^5 \times 3^3$	(1 1 0 0 0)
57	175	$5^2 \times 7$	(0 0 0 1 0)
62	770	$2 \times 5 \times 7 \times 11$	(1 0 1 1 1)
69	150	$2 \times 3 \times 5^2$	(1 1 0 0 0)
80	252	$2^2 \times 3^2 \times 7$	(0 0 0 1 0)

Table 2.5: Relations found with smooth y_i

[2]There are more relations than this, those shown are examples only.

The prime factorisations of each y_i are also represented as vectors $v_i \in \mathbb{F}_2^B$, with $\mathbb{F}_2^B$ the B-dimensional vector space over the finite field $\mathbb{Z}/2\mathbb{Z}$. [8]
The vector entries thus show the parity of each prime's exponent in the factorisation.

Example 2.15. $756 = 2^2 \times 3^3 \times 5^0 \times 7^1 \times 11^0$ would be represented (0 1 0 1 0).

We may now generate a square using linear algebra. Provided we have B+1 or more vectors $v_i \in \mathbb{F}_2^B$, there must be a non-trivial linear dependence between them. Gaussian elimination or another algorithm for solving systems of linear equations may now be used to find a combination of vectors which sum to the zero vector.

In this case, we see by inspection that adding the first 3 vectors $v_1 + v_2 + v_3$ gives:

$$(1\ 1\ 0\ 1\ 0) + (1\ 1\ 0\ 0\ 0) + (0\ 0\ 0\ 1\ 0) = (0\ 0\ 0\ 0\ 0)$$

This is equivalent to multiplying x_1, x_2, x_3 and ensures that all prime powers are even. This gives the congruence of squares:

$$47^2 \times 49^2 \times 57^2 \equiv 2^{10} \times 3^4 \times 5^2 \times 7^2 \pmod{1537}$$

$$\text{or}$$

$$(47 \times 49 \times 57)^2 \equiv (2^5 \times 3^2 \times 5 \times 7)^2 \pmod{1537}$$

Reducing modulo 1537 gives:

$$626^2 \equiv 858^2 \pmod{1537}$$

We now calculate $(858 + 626, 1537) = 53$ and $(828 - 626, 1537) = 29$ to find the factorisation $1537 = 29 \times 53$.

It's possible that this method gives a trivial factorisation of n, in which case we try again with a different combination of relations or an expanded factor base. It is clear that the speed of this method (and others based upon it) relies on finding enough relations efficiently. In addition, we require some efficient way of determining smoothness. With a smaller factor base, finding a linear dependency is faster but we risk not being able to find enough smooth values to guarantee a dependency. [34] Thus, our choice of factor base is critical to the algorithm's runtime.

We now consider the Continued Fraction method, which uses continued fractions to find relations of the form (2.4).

2.3.3 Continued Fraction method

Lehmer and Powers introduced the Continued Fraction (CFRAC) method in a 1931 paper, but it was not until the 1970's that computers were powerful enough to fully exploit their ideas.

Definition 2.16. A **continued fraction** is an expression of the form:

$$a_0 + \cfrac{b_0}{a_1 + \cfrac{b_1}{a_2 + \cfrac{b_2}{\cdots + \cfrac{b_n}{a_n}}}}$$

$a_i, b_i \in \mathbb{Z}$ and all a_i positive for $i > 0$.
If all $b_i = 1$, the continued fraction is **regular**.

Earlier factorisation methods utilising continued fractions depended on finding squares in the denominators of an expansion, which happens only rarely. As [25] observes, more often we may find a square as the *'...product of two more denominators.'*

To generate relations, we consider the continued fraction expansion of $\sqrt{kn}$. To see why, consider the congruence $x^2 \equiv W \pmod{n}$ for some small W. If W is small enough, it has a reasonable chance of being smooth. We can express this congruence as $x^2 = W + knd^2$ for some $k, d \in \mathbb{Z}$. Then $(\frac{x}{d})^2 - kn = W/d^2$ and W/d^2 is small. So $\frac{x}{d} \in \mathbb{Q}$ is an approximation of $\sqrt{kn}$. [45]

We first generate a sequence of convergents $\frac{A_i}{B_i}$ to $\sqrt{kn}$, and set $W_i = A_i^2 - B_i^2 kn$.

Definition 2.17. The **convergents** of a continued fraction are rational approximations to a real number. The first convergent is given by $\dfrac{a_0}{1}$. The n_{th} convergent is given by the following recursive formula, with the a_i from the continued fraction expansion: [21, §1.2]

$$\frac{h_n}{d_n} = \frac{a_n h_{n-1} + h_{n-2}}{a_n d_{n-1} + d_{n-2}}$$

For details on finding the continued fraction representation of $\sqrt{n}$ and generating convergents, see §2.5 and §2.6 of [21].

Example 2.18. We shall factorise $n = 1711$ using the Continued Fraction method.

We set $k = 1$ in this expansion and will use the factor base $P_B = \{-1, 2, 3, 5\}$.

Note that we include -1 in the factor base. This allows us to consider negative congruences, which may be of smaller absolute value, and thus more likely to be smooth. As we require that the power of -1 in our final congruence be even, the linear algebra step is unchanged.

The continued fraction expansion of $\sqrt{1711}$ is:

$$\sqrt{1711} = 41 + \cfrac{1}{2 + \cfrac{1}{1 + \cfrac{1}{2 + \cfrac{1}{1 + \cfrac{1}{13 + \cdots}}}}}$$

We show the convergents in Table 2.6. Note that only convergents with W_i smooth are shown for clarity.

$\frac{A_i}{B_i}$ **(mod 1711)**	$W_i = A_i^2 - 1711 \times B_i^2$ **(mod 1711)**	**Factorisation of** W_i	**Vector**
$\frac{41}{1}$	-30	$-1 \times 2 \times 3 \times 5$	(1 1 1 1)
$\frac{83}{2}$	45	$3^2 \times 5$	(0 0 0 1)
$\frac{455}{11}$	-6	$-1 \times 2 \times 3$	(1 1 1 0)
$\frac{1113}{151}$	5	1×5	(0 0 0 1)
$\frac{1129}{1403}$	-54	$-1 \times 2 \times 3^3$	(1 1 1 0)

Table 2.6: Smooth values found using the CFRAC method

We observe that summing the third and fifth vectors generates the zero vector (0 0 0 0). By the definition of W_i, we have $W_i \equiv A_i^2 \pmod{n}$, giving the congruence:

$$455^2 \times 1129^2 \equiv -1^2 \times 2^2 \times 3^4 \pmod{1711}$$

Which simplifies and reduces to:

$$395^2 \equiv 18^2 \pmod{1711}$$

Finally, we calculate $(395 + 18, 1711) = 59$ and $(395 - 18, 1711) = 29$ and obtain the factorisation $1711 = 29 \times 59$.

In 1970, Morrison and Brillhart developed a computer implementation of CFRAC and used it to factorise the seventh Fermat number, $2^{2^7}+1$, a number with 39 decimal digits. [31] Although this represented a significant milestone in factorising technology, the CFRAC algorithm was soon superseded by a forerunner to today's methods, the Quadratic Sieve of Carl Pomerance.

2.3.4 The Quadratic Sieve

A primary innovation of the Quadratic Sieve, attributed by Pomerance in [34] to unpublished work by Richard Schroeppel in the late 1970's, was the introduction of a sieving process to drastically speed up the identification of smooth values. Although the primes in our factor base may be small relative to the modulus, much time is wasted by trial division on non-smooth values, which cannot be used to form a congruence.

The sieve inverts this process: rather than dividing every value by each prime in our factor base to determine if they are smooth, we instead fix a prime $p_i \in P_B$ and determine all values which are divisible by p_i. Pomerance's choice of polynomial facilitates this step. Recall from §2.3.2 that Dixon's method seeks $x^2 \equiv y \pmod{n}$ where y is B-smooth.

Pomerance instead proposed finding smooth values of the polynomial:

$$f(x_i) = x_i^2 - n \tag{2.5}$$

Once B+1 x_i with $f(x_i)$ smooth are found, we use linear algebra to find a subset S with: [8]

$$\prod_{x_i \in S} f(x_i) = \prod_{p_i \in P_B} (p_i^{q_i})^2$$

Setting $a = \prod_{x_i \in S} x_i$ and $b = \prod_{p_i \in P_B} (p_i^{q_i})$, gives the required congruence of squares:

$$a^2 = \prod_{x_i \in S} (x_i^2) \equiv \prod_{x_i \in S} (x_i^2 - n) \equiv \prod_{x_i \in S} f(x_i) = b^2 \pmod{n} \tag{2.6}$$

Proposition 2.19. *If $p|f(x_i)$ for $p \in P_B$, then $p|f(x_i + kp)$ for $k \in \mathbb{Z}$.*

Proof. As $p|f(x_i)$, $f(x_i) \equiv 0$ (mod p). So $x_i^2 - n \equiv 0$ (mod p).
Now, $f(x_i + kp) = (x_i + kp)^2 - n = (x_i^2 + 2x_ikp + k^2p^2) - n) \equiv x_i^2 - n \equiv 0$ (mod p). [8]
So $p|f(x_i + kp)$. □

To sieve by p_j, we first need to solve the congruence:

$$x_i^2 - n \equiv 0 \pmod{p_j}. \tag{2.7}$$

The Tonelli-Shanks algorithm is an efficient method for doing this. [44] We sieve over an interval $[-u \leq x_i \leq u]$ that is heuristically expected to generate enough smooth values of $f(x_i)$. To do this, a $2u \times B$ array is initialised with the $2u$ values of $f(x_i)$ in the first column. For x_i solving (2.7) we divide $f(x_i)$ by p_j and put the result in the j_{th} column. We can then divide all further values $f(x_i + kp_j)$ in the interval by p_j by Proposition 2.19.

This process only works if n is a quadratic residue modulo p (otherwise (2.7) has no solutions), so we must modify our factor base P_B to contain only primes satisfying this condition. The process is then repeated for each $p_j \in P_B$. Finally the array is scanned for values of ± 1, which indicate that the $f(x_i)$ factors completely over the factor base. As Pomerance mentions, $f(x_i)$ values divisible by p_j^a for $a \geqslant 2$ will not equal 1 unless we sieve further by higher powers of p_j. For example, 60 would be reduced to 2 after sieving only by $2^1, 3^1,$ and 5^1. [35]

Example 2.20. We will sieve the interval [0,60] for smooth values of $f(x_i) = x_i^2 - n$ and use this to factorise $n = 2291$ with $P_B = \{-1, 2, 3, 5, 7, 11\}$.

We first need to solve (2.7) modulo the $p_j \in P_B$. We find that 2291 is a quadratic non-residue modulo 3, so we remove 3 from our factor base.

Besides $p = 2$, for any p for which n is a quadratic modulo, there will be 2 solutions. We find $x_i^2 \equiv 2291 \pmod 5$ has solutions $x \equiv 1 \pmod 5$ & $x \equiv 4 \pmod 5$.

We thus go to the 1st and 4th rows in the column $p = 5$ and divide $f(x_i)$ by 5. We continue down the column, dividing by 5 in the $(1+5k)_{th}$ and $(4+5k)_{th}$ positions. To sieve by higher powers, we next solve $x_i^2 \equiv 2291 \pmod{5^2}$ and repeat this process.

Table 2.7 shows the results of sieving with $P_B = \{-1, 2, 5, 7, 11\}$ and some small values of p_j^a. We only tabulate a few selected values with small factors for clarity, including any x_i that are found to generate smooth $f(x_i)$ values, shown in bold.

$\mathbf{x}_i$	$\mathbf{x}_i^2 - 2291$	**2**	**5**	$\mathbf{5^2}$	$\mathbf{5^3}$	**7**	$\mathbf{7^2}$	$\mathbf{7^3}$	**11**
1	-2290	-1145	-229	-	-	-	-	-	-
4	-2275	-	-455	-91	-	-13	-	-	-
17	-2002	-1001	-	-	-	-143	-	-	-13
24	-1715	-	-343	-	-	-49	-7	**-1**	-
31	-1330	-665	-133	-	-	-19	-	-	-
39	-770	-385	-77	-	-	-11	-	-	**-1**
46	-175	-	-35	-7	-	**-1**	-	-	-
49	110	55	11	-	-	-	-	-	**1**
51	310	155	31	-	-	-	-	-	-
60	1309	-	-	-	-	187	-	-	17

Table 2.7: Selected sieving results

After scanning the sieve, we find the following smooth values and use them to generate the exponent matrix:

$\mathbf{x_i}$	$\mathbf{f(x_i)}$	**Factorisation**
24	-1715	$-1 \times 5 \times 7^3$
39	-770	$-1 \times 2 \times 5 \times 7 \times 11$
46	-175	$-1 \times 5^2 \times 7$
49	110	$2 \times 5 \times 11$

$$\longrightarrow \begin{bmatrix} 1 & 0 & 1 & 1 & 0 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 0 & 0 & 1 & 0 \\ 0 & 1 & 1 & 0 & 1 \end{bmatrix}$$

Although we have only collected 4 smooth values and a dependency is not guaranteed, in this case we are fortunate that $v_2 + v_3 + v_4 = (0\ 0\ 0\ 0\ 0)$.

We thus generate the congruence by (2.6):

$$(39 \times 46 \times 49)^2 \equiv (2 \times 5^2 \times 7 \times 11)^2 \pmod{2291}$$

or

$$87906^2 \equiv 1559^2 \pmod{2291}$$

Calculating as usual $(87906 + 1559, 2291) = 29$ and $(87906 - 1559, 2291) = 79$ gives the factorisation $2291 = 29 \times 79$.

As there are fewer primes allowed in our factor base due to the quadratic residue condition, we may miss some $f(x_i)$ values with only small prime factors, but we save time by only dividing by a p_j once we already know that $p_j | f(x_i)$. This represents a huge gain in efficiency over naive trial division methods of finding smooth values.

2.3.5 Optimisation and the Multiple Polynomial Quadratic Sieve

Pomerance introduced an optimisation in [34] that further speeds up the sieving process. Rather than dividing by the p_j's directly, we instead initialise the array with the values of $\log f(x_i)$ and subtract $\log p_j$, which is equivalent to dividing by p_j by the Second Log Law. As the logarithms are computed only roughly, this is much quicker than division by (potentially) large numbers. The array is then scanned for values close to 0, which correspond to values *likely* to be smooth. Only at this point is time-consuming trial division performed, in order to verify that the $f(x_i)$ actually is smooth. In this way, we only perform trial division on values highly likely to be smooth.

Depending on the initial accuracy of the logarithms, there will be some small percentage of misidentified non-smooth values tested, but the prior time savings more than make up for these unnecessary divisions.

Example 2.21. We show the sieving process using logarithms only for $x_i = 24, 31$ as examples. Recall from Table 2.7 that $x_i = 24$ gave a smooth $f(x_i)$.

$\mathbf{x}_i$	$\mathbf{log_{10}}\ \lvert(x_i^2 - 2291)\rvert$	**2**	**5**	$\mathbf{5^2}$	$\mathbf{5^3}$	**7**	$\mathbf{7^2}$	$\mathbf{7^3}$	**11**
24	3.234	-	2.535	-	-	1.690	0.845	**0**	-
31	3.124	2.823	2.124	-	-	1.279	-	-	-

Table 2.8: Sieving results using logarithms

A further improvement on the algorithm was developed by Peter Montgomery. [34] As the values of $f(x_i)$ grow larger, it becomes increasingly more unlikely that they will be smooth over the factor base. Montgomery proposed employing multiple quadratic polynomials to increase the number of smooth values found. These days, it is this form of the algorithm that is generally used, and that has made the factorisation of 60-80 digit numbers feasible on a home computer.

Montgomery's optimisation also allows the distribution of computation across a network, as each polynomial may be sieved in parallel, with the results then combined for the linear algebra processing. For further details on the Multiple Polynomial Quadratic Sieve (MPQS) see [38].

During the mid-90's Lenstra and Manasse exploited the parallelisation of the sieve to distribute factorising projects over the internet. [35] Sample factorisation data from this project is shown below to give a more realistic example of the size of factor bases and primes required for a successful factorisation with MPQS. [28] Table 2.9 also indicates the rapid growth in computation time as the size of the modulus increases.[3]

Parameter	**116-digit n**	**120-digit n**	**129-digit n**
Size of factor base	120,000	245,810	524,339
Prime bound B	10^8	2^{30}	2^{30}
Data generated (GB)	0.25	1.1	2
Sieving time (MY)	400	825	5000
Matrix processing time (Hours)	0.5	4	45

Table 2.9: Factorisation data using MPQS

The Quadratic Sieve set many factoring records throughout the 1980's, before it too, was superseded by the current fastest-known algorithm, the General Number Field Sieve (GNFS). For n with less than 120 digits or so, [19] MPQS remains the fastest algorithm due to the complexity of setting up the GNFS. Above this threshold, the GNFS begins to be faster.

In this section, we have described some important general-purpose algorithms and laid the mathematical foundations for an outline of the General Number Field Sieve in §3.

[3]Sieving time is given in units of MIPS-years; at the time, 1 MY equalled 1 year of processing on a VAX 11/780. [28]

Chapter 3

The General Number Field Sieve

3.1 Foundations

3.1.1 History

The foundations of the General Number Field Sieve can be traced to ideas circulated in a letter by John Pollard in 1988 for factorising a special class of numbers: those of the form $r^e + s$ with small positive r, s. In his letter, Pollard proposed F_9, with 155 decimal digits, as a potential candidate for factorisation using what is now called the Special Number Field Sieve (SNFS). This was successfully achieved in 1990, by A Lenstra, H Lenstra, and Manasse. [26] The General Number Field Sieve employs similar ideas but removes the input conditions and works with integers of any form, at the expense of some speed. We thus concentrate on the GNFS for the remainder of this chapter. For further information on the SNFS, see [27] and §4 & 5 of [26].

3.1.2 Key ideas

The key innovation of Pollard's idea was to expand the search for smooth values beyond the ring $\mathbb{Z}$, using algebraic number fields. If we can find rings that generate more smooth values than $\mathbb{Z}$, we should in theory be able to find relations faster. Such an idea is not obvious - smoothness is easily understood when dealing with integers, but how do we apply it to elements not in $\mathbb{Z}$? To answer this, we will need to extend our concept of smoothness.

A second, more intuitive, innovation is to use polynomials other than quadratics as our generating polynomial $f(x_i)$. If some higher-power polynomials generate more smooth values than quadratics, again, we may find relations faster. The expansion of our scope to other rings requires some knowledge of algebraic number theory; we outline the necessary mathematical background in §3.2. In §3.3, we present the theory required to extend our concept of smoothness, and discuss sieving for the GNFS in §3.4. The final step of calculating a square root in an algebraic number field is discussed in §3.5.

3.2 Rings of algebraic integers

3.2.1 Ring homomorphisms

Recall that the quadratic sieve maps squares in $\mathbb{Z}$ to squares in $\mathbb{Z}/n\mathbb{Z}$.

Definition 3.1. A **ring homomorphism** is a function from a ring R to a ring S, $\phi : R \to S$ such that: [40]

- $\phi(1_R) = (1_S)$
- $\phi(a+b) = \phi(a) + \phi(b)$ for all $a, b \in R$
- $\phi(ab) = \phi(a)\phi(b)$ for all $a, b \in R$

To generate a congruence of squares using rings other than $\mathbb{Z}$, we construct a ring homomorphism. We first choose a monic, irreducible polynomial $f(x)$ of degree d with rational coefficients. Typically for numbers in the 120-150 digit range, we set $d = 5$.

Then $f(x)$ factorises: $f(x) = (x - \theta_1)(x - \theta_2)\ldots(x - \theta_d)$ with $\theta_i \in \mathbb{C}$.

Each root θ may be used to form a ring:

Proposition 3.2. *Let θ be the root of a monic, irreducible polynomial $f(x)$ of degree d with rational coefficients. The set of all polynomials in θ with rational coefficients is a ring, denoted $\mathbb{Q}(\theta)$.*

By Theorem 2.2.2 of [8], $\mathbb{Q}(\theta)$ is a field and the elements of $\mathbb{Q}(\theta)$ may be represented as $\mathbb{Q}$-linear combinations of elements of the set $R = \{1, \theta, \theta^2, \ldots, \theta^{d-1}\}$, i.e. R forms a basis for $\mathbb{Q}(\theta)$.

3.2.2 The ring $\mathbb{Z}[\theta]$

We now consider the ring of integers of the field $\mathbb{Q}(\theta)$.

Definition 3.3. An $\alpha \in \mathbb{C}$ is an **algebraic integer** if it is the root of some monic polynomial with integer coefficients. [40]

That is, for an equation of the form: $f(x) = x^n + \cdots + a_2x^2 + a_1x + a_0$ with $a_i \in \mathbb{Z}$, we have $f(\alpha) = 0$.

Definition 3.4. A **number field** K is an extension of the field $\mathbb{Q}$ that is a finite dimensional vector space. We call the dimension of the vector space the **degree** of K.

The set of all algebraic integers in a field $F \subseteq \mathbb{C}$ is called the ring of integers.

Definition 3.5. The **ring of integers** O_K of the number field $K \subseteq \mathbb{C}$ is defined: $O_K = K \cap \mathbb{A}$ where $\mathbb{A}$ is the ring of algebraic integers.

As $\mathbb{Q}(\theta)$ is a field, we now form a subring of $O_{\mathbb{Q}(\theta)}$ by Proposition 2.3.2 of [8], as follows:

Proposition 3.6. *Given a root θ of a monic, irreducible polynomial f(x) such that:*

$$f(x) = x^n + \cdots + a_2x^2 + a_1x + a_0 \text{ with } a_i \in \mathbb{Z},$$

the set of $\mathbb{Z}$-linear combinations of the elements $\{1, \theta, \theta^2, \ldots, \theta^{d-1}\}$, forms a subring of $O_{\mathbb{Q}(\theta)}$, denoted $\mathbb{Z}[\theta]$.

It is this ring $\mathbb{Z}[\theta]$ which we will use to find smooth values. By using $\mathbb{Z}$-linear combinations of the elements, we can ignore denominators in our representations of the elements. This simplification does have some important consequences; our ring is now possibly a *proper subring* of $O_{\mathbb{Q}(\theta)}$, which causes some complications, which we shall resolve once we have found smooth values in $\mathbb{Z}[\theta]$.

3.2.3 Generating a congruence of squares

How then do we use the ring $\mathbb{Z}[\theta]$ and a ring homomorphism to generate a congruence of squares? Consider the following mapping: [35]

Proposition 3.7. *Given $f(x)$ satisfying the conditions of Proposition 3.6, a root $\theta \in \mathbb{C}$, and $m \in \mathbb{Z}/n\mathbb{Z}$ such that $f(m) \equiv 0$ (mod n), the function $\phi : \mathbb{Z}[\theta] \to \mathbb{Z}/n\mathbb{Z}$ defined by:*

$$\phi(1) = 1 \text{ (mod n)}$$
$$\phi(\theta) = m \text{ (mod n)}$$

is a ring homomorphism.

We will use this homomorphism to construct the required congruence using smooth values in $\mathbb{Z}[\theta]$ and $\mathbb{Z}$.

Suppose we have found a set S of pairs of coprime integers (a, b) such that:

$$\prod_{(a,b)\in S} (a + b\theta) = \beta^2 \text{ for } \beta \in \mathbb{Z}[\theta], \text{ and } \prod_{(a,b)\in S} (a + bm) = y^2 \text{ for } y \in \mathbb{Z} \tag{3.1}$$

Then setting $x = \phi(\beta)$ and applying the homomorphism of Proposition 3.7 gives: [35]

$$x^2 \equiv \phi(\beta)^2 \equiv \phi(\beta^2) \equiv \phi\left(\prod_{(a,b)\in S} (a + b\theta)\right) \equiv \prod_{(a,b)\in S} \phi(a+b\theta) \equiv \prod_{(a,b)\in S} (a+bm) \equiv y^2 \pmod{\text{n}} \tag{3.2}$$

The congruence of (3.2) is the heart of the Number Field Sieve. Note that in order to find our factors using this congruence we must calculate $(x + y, n)$ and $(x - y, n)$. We therefore need to find x, or what is equivalent, calculate the square root of a product of algebraic numbers, a non-trivial procedure which we shall outline in §3.5. Conversely, finding y is straight-forward as $y^2 \in \mathbb{Z}$.

3.2.4 Choosing a suitable polynomial

To generate $\mathbb{Z}[\theta]$, we require a monic, irreducible polynomial $f(x)$ with integer coefficients. A simple way of finding one is the base-m method. [10] We choose m *first* and construct our polynomial to satisfy the condition $f(m) \equiv 0 \pmod{n}$. m is chosen such that $m \approx n^{\frac{1}{d}}$. Writing n in base-m then gives:

$$n = m^d + a_{d-1}m^{d-1} \dots a_1m^1 + a_0 \text{ with } 0 \leq a_i < m \tag{3.3}$$

The coefficients in this expansion are used to construct the polynomial:

$$f(x) = x^d + a_{d-1}x^{d-1} \dots a_1x + a_0 \tag{3.4}$$

By construction, f is monic and $f(m) \equiv 0 \pmod{n}$ as required. However, f may be reducible. In this case, there exists a polynomial-time algorithm [35] that will factor f into its irreducible factors. Then $f(x) = g(x)h(x)$ for some polynomials $g(x), h(x)$, in which case we can factorise n without needing to employ the GNFS. Otherwise, $f(x)$ is irreducible as required and we may proceed with sieving.

The base-m method was suggested in early papers on the GNFS. However, our initial choice of polynomial can greatly affect the efficiency of the algorithm, with 'tuned' polynomials yielding greater numbers of smooth values. This is a particularly active area of GNFS research, for further details see [32] and [22].

3.3 Factor bases in $\mathbb{Z}$ & $\mathbb{Z}[\theta]$

3.3.1 The rational factor base

By the congruence (3.2) and the definitions of (3.1), we need to find a set of coprime integers (a, b) such that the products $\prod_{(a,b)\in S}(a+b\theta)$ and $\prod_{(a,b)\in S}(a+bm)$ are squares in their respective rings.

To find $\prod_{(a,b)\in S}(a+bm)$ square, we use essentially the same process as in the quadratic sieve, except we now have two variables a, b rather than the single variable x of (2.5).

In practice, we fix $b \in \mathbb{Z}$ and sieve over an interval $[-u \leq a \leq u]$, seeking smooth values of $(a + bm)$. We then increment b by 1 and repeat the process.

Definition 3.8. The **rational factor base** is a set P_R of primes $p \leq B_R$ for some bound B_R.

To find the array positions for sieving, we note that p$|(a + bm) \iff a \equiv -bm \pmod{p}$. Then $a_i = -bm + kp$ for some $k \in \mathbb{Z}$ and $-u \leq a \leq u$, so we can calculate all possible values of a_i in the interval. [8]

The array is initialised with the values of $(a + bm)$ and we proceed to divide by p in the $a_i th$ positions. The entire process is then repeated for the next $p \in P_R$. We thus find smooth values in a manner similar to that of the quadratic sieve. We must now extend the concept of smoothness to the ring $\mathbb{Z}[\theta]$.

3.3.2 The algebraic factor base

We begin with some definitions of algebraic number theory.

Definition 3.9. Let R be a ring. An element $r \in R$ is a **unit** if $\exists s \in R$ with $rs = 1$.

Definition 3.10. Let R be a ring. Two elements $r, s \in R$ are **associates** if $r = us$ with u a unit of R.

Definition 3.11. A **unique factorisation domain** (UFD) is a ring R satisfying the following conditions: [40, §4.5]

- Every $r \in R$ that is not a unit is a product of irreducible elements of R.

- If $p_1, p_2, \dots p_n, q_1, q_2, \dots q_m \in R$ are irreducible and $p_1 p_2 \dots p_n = q_1 q_2 \dots q_m$ then $n = m$ and p_i, q_i are associates.

The principal issue with extending the concept of smoothness is that $\mathbb{Z}[\theta]$ is not necessarily a unique factorisation domain. So the elements of $\mathbb{Z}[\theta]$ may not be 'factored' uniquely into irreducible elements. To resolve this, we instead consider factorisation of ideals into prime ideals, which *will* be unique, provided we are working in a Dedekind domain.

Definition 3.12. A **Dedekind domain** is a ring R satisfying the following conditions:

- R is a Noetherian Integral Domain. (Every ideal of R is finitely generated.)
- Every non-zero proper ideal of R factors into prime ideals.

To find (a, b) pairs with $(a + b\theta)$ smooth, the general idea is to first choose an 'algebraic' factor base, a set of prime ideals of $\mathbb{Z}[\theta]$. We then consider the principal ideal $\langle a + b\theta \rangle$ generated by each element $(a + b\theta)$ and check to see if it factorises completely into prime ideals of our factor base. Those which do are smooth over the algebraic factor base and the (a, b) pair is useful.

We now describe the details of determining smoothness in $\mathbb{Z}[\theta]$. We begin with an important fact about the ring of integers.

Theorem 3.13. *The ring of integers O_K of a number field K is a Dedekind domain.*

Proof. See [40, §5.2, Theorem 5.3] □

This gives $O_{\mathbb{Q}(\theta)}$ the following important property:

Theorem 3.14. *Every non-zero ideal of $O_{\mathbb{Q}(\theta)}$ can be written as a product of prime ideals, uniquely up to the order of factors.*

Proof. See [40, §5.2, Theorem 5.6] □

We can use this property of the ring of integers to uniquely factorise ideals in the ring $\mathbb{Q}(\theta)$ into prime ideals and thereby extend our concept of smoothness. As the elements $\alpha \in \mathbb{Z}[\theta]$ are algebraic integers, to simplify this approach we instead work with the norm of α.

Theorem 3.15. *Let $K = \mathbb{Q}(\theta)$ be a number field of degree d. Then there are exactly d distinct monomorphisms $\sigma_i : \mathbb{Q}(\theta) \to C$ $(i = 1, \dots, d)$.*

Proof. See [40, §2.2, Theorem 2.4] □

Definition 3.16. Let $K = \mathbb{Q}(\theta)$ be a number field of degree d and let $\sigma_1, \ldots, \sigma_d$ be the d monomorphisms $\sigma_i : \mathbb{Q}(\theta) \to C$ of Theorem 3.15.

The **norm of an element** $\alpha \in K$ is defined as: [40, §2.5]

$$N(\alpha) = \prod_{i=1}^{n} \sigma_i(\alpha)$$

We can also define the norm of an ideal:

Definition 3.17. Let $\mathfrak{I}$ be a non-zero ideal. The **norm of an ideal** is defined as:

$$N(\mathfrak{I}) = |O_K/\mathfrak{I}|$$

The following two results highlight the usefulness of these two functions.

Theorem 3.18. *Given a monic, irreducible polynomial of degree d with integer coefficients, a root $\theta \in \mathbb{C}$, and an element $\alpha \in \mathbb{Z}[\theta]$, $N(\alpha)$ is a multiplicative function $N(\alpha) : \mathbb{Z}[\theta] \to \mathbb{Z}$.*

Proof. See [40, §5.2] □

Theorem 3.19. *Given a monic, irreducible polynomial of degree d with rational coefficients, a root $\theta \in \mathbb{C}$, and $\mathfrak{I}$ a non-zero ideal of K, $N(\mathfrak{I})$ is a multiplicative function $N(\mathfrak{I}) : O_K \to \mathbb{Z}^+$. If $\mathfrak{a} = \langle a \rangle$ is a principal ideal, then $N(\langle \mathfrak{a} \rangle) = |N(a)|$.*

Proof. See [40, §5.3, Corollary 5.10] □

Thus the norm conveniently maps elements $\alpha \in \mathbb{Z}[\theta]$ to integers. For a principal ideal generated by α, the norm of the ideal is a positive integer and is equal to the absolute value of the norm $N(\alpha)$. Rather than working with $\alpha \in \mathbb{C}$, we can now work with integers. To simplify sieving, we restrict the algebraic factor base to first-degree prime ideals only. [8]

Definition 3.20. A **first-degree** prime ideal $\mathfrak{p}$ of a Dedekind domain is a prime ideal such that $N(\mathfrak{p}) = p$ for some prime p.

This definition is justified by the following theorem, which relates prime ideals $\mathfrak{p}$ to prime numbers in $\mathbb{Z}$.

Theorem 3.21. *Let D be a Dedekind domain, and $\mathfrak{a}$ an ideal of D:*

- *If $N(\mathfrak{a}) = p$ with p prime, then $\mathfrak{a}$ is a prime ideal.*
- *If $\mathfrak{a}$ is a prime ideal, then $N(\mathfrak{a}) = p^m$ for some prime p with $m \leq d$.*

Proof. See [40, §5.3, Theorem 5.14] □

First-degree prime ideals are convenient to work with as they may be simply represented by pairs of integers (r, p), due to the following theorem:

Theorem 3.22. *Let $f(x)$ be a monic, irreducible polynomial with integer coefficients, and $\theta \in \mathbb{C}$ a root of $f(x)$.*

Consider the set of pairs (r, p) with $r \in \mathbb{Z}/p\mathbb{Z}$, $f(r) \equiv 0 \pmod p$ and p a prime integer. There exists a bijection between this set and the set of first-degree prime ideals of $\mathbb{Z}[\theta]$.

Proof. We first require the following definitions:

Definition 3.23. A ring **epimorphism** is a surjective ring homomorphism.

Definition 3.24. The **kernel** of a ring homomorphism $\phi : R \to S$, denoted $ker(\phi)$, is the set: $ker(\phi) = \{r \in R : \phi(r) = 0_S\}$

We now show that each first-degree prime ideal maps to a unique (r, p) pair satisfying the conditions of the theorem. Let $\mathfrak{p}$ be a first-degree prime ideal of $\mathbb{Z}[\theta]$. Then $N(\mathfrak{p}) = p$ for some prime p by definition.

$N(\mathfrak{p}) = p \implies \mathbb{Z}[\theta]/\mathfrak{p} \cong \mathbb{Z}/p\mathbb{Z}$. There exists an epimorphism $\phi : \mathbb{Z}[\theta] \to \mathbb{Z}/p\mathbb{Z}$ with $ker(\phi) = \mathfrak{p}$. [8, Theorem 3.1.7]

Let $r = \phi(\theta) \in \mathbb{Z}/p\mathbb{Z}$. Let $f(x) = x^d + a_{d-1}x^{d-1} \dots a_1 x + a_0$ as in (3.4).
Then we have:

$$\begin{aligned} 0 \pmod p \equiv \phi\left(f(\theta)\right) &\equiv \phi(\theta^d + a_{d-1}\theta^{d-1} \dots a_1\theta + a_0) \\ &\equiv \phi(\theta)^d + a_{d-1}\phi(\theta)^{d-1} \dots a_1\phi(\theta) + a_0 \\ &\equiv r^d + a_{d-1}r^{d-1} \dots a_1 r + a_0 \\ &\equiv f(r) \pmod p \end{aligned}$$

So r is a root of $f(x) \pmod p$ and $\mathfrak{p}$ maps to the unique pair (r, p). We now show that a pair (r, p) with $r \in \mathbb{Z}/p\mathbb{Z}$, $f(r) \equiv 0 \pmod p$ and p prime maps to a unique first-degree prime ideal $\mathfrak{p}$.

There exists an epimorphism $\phi : \mathbb{Q}[\theta] \to \mathbb{Q}[r]$, $\phi(a) \equiv a \pmod p$ for all $a \in \mathbb{Z}$ and $\phi(\theta) = r \pmod p$. [8, Theorem 2.2.2]
Let $ker(\phi) = \mathfrak{p}$ so that $\mathfrak{p}$ is an ideal of $\mathbb{Z}[\theta]$. Then $\mathbb{Z}[\theta]/\mathfrak{p} \cong \mathbb{Z}/p\mathbb{Z}$ and $N(\mathfrak{p}) = p$. So (r, p) maps to a unique first-degree prime ideal $\mathfrak{p}$ of $\mathbb{Z}[\theta]$. □

Theorem 3.22 shows that there is a unique (r, p) representative for each first-degree prime ideal in $\mathbb{Z}[\theta]$. We are finally in a position to define the algebraic factor base.

Definition 3.25. The **algebraic factor base** is a set P_A of first-degree prime ideals of $\mathbb{Z}[\theta]$, represented by pairs (r, p) with $r \in \mathbb{Z}/p\mathbb{Z}$, $f(r) \equiv 0 \pmod p$ and p a prime, $p \leq B_A$ for some bound B_A. Each pair represents a unique first-degree prime ideal of $\mathbb{Z}[\theta]$.

By the unique factorisation of a principal ideal $\langle\beta\rangle = \mathfrak{p}_1^{e_1}\mathfrak{p}_2^{e_2}\ldots\mathfrak{p}_k^{e_k}$ to prime ideals $\mathfrak{p}_i$, the multiplicativity of $N(\alpha)$ and Theorems 3.19 and 3.21, we have: [8]

$$N(\langle\beta\rangle) = |N(\beta)| = N(\mathfrak{p}_1^{e_1}\mathfrak{p}_2^{e_2}\ldots\mathfrak{p}_k^{e_k}) = N(\mathfrak{p}_1^{e_1})N(\mathfrak{p}_2^{e_2})\ldots N(\mathfrak{p}_k^{e_k}) = (p_1^{f_1})^{e_1}(p_2^{f_2})^{e_2}\ldots(p_k^{f_k})^{e_k} \tag{3.5}$$

for primes $p_i; e_i, f_i \in \mathbb{Z}^+$ and $1 \leq i \leq k$.

Note that the primes need not be distinct, as the norms of different prime ideals may be factorised by the same prime p, but to different powers. We now define a function that tracks the powers of p in $N(a+b\theta)$ for $(a+b\theta)$ generating a principal ideal $\langle a+b\theta\rangle$ which factorises fully over the algebraic factor base. [10, Remark 5.2]

Definition 3.26. Let f be a monic, irreducible polynomial of degree d with integer coefficients, and a root $\theta \in \mathbb{C}$.
Let (a, b) be coprime integers, p a prime integer and $r \in R(p) = \{r : f(r) \equiv 0 \text{ (mod p)}\}$.

We define:

$$e_{p,r}(a+b\theta) = \begin{cases} ord_p(N(a+b\theta)) & \text{if } a+br \equiv 0 \pmod p \\ 0 & \textit{otherwise} \end{cases}$$

where $ord_p\,(N(a+b\theta))$ is the exponent of p in the prime factorisation of $N(a+b\theta)$.

Then by Definition 3.26,

$$N(a+b\theta) = \pm\prod_{p,r} p^{e_{p,r}(a+b\theta)}$$

To use the function $e_{p,r}$ we need to generalise for the case in which $\mathbb{Z}[\theta] \neq O_K$. (This will usually be the case in practice.) To do this, we consider the following group homomorphism.

Theorem 3.27. *There exists, for each prime ideal $\mathfrak{p}_i \in \mathbb{Z}[\theta]$ a group homomorphism*
$\mathfrak{l}_{\mathfrak{p}_i} : \mathbb{Q}(\theta) \to \mathbb{Z}$ such that:

- $\mathfrak{l}_{\mathfrak{p}_i}(\beta) \geq 0$ *for all* $\beta \in \mathbb{Z}[\theta], \beta \neq 0$

- $\mathfrak{l}_{\mathfrak{p}_i}(\beta) > 0$ *if and only if* $\mathfrak{p}_i$ *divides* $\langle\beta\rangle$
- $\mathfrak{l}_{\mathfrak{p}_i} = 0$ *for all but a finite number of* $\mathfrak{p}_i \in \mathbb{Z}[\theta]$*, and* $|N(\beta)| = \prod N(\mathfrak{p}_i{}^{\mathfrak{l}_{\mathfrak{p}_i}})$

Proof. See §7 of [10] for an extended proof of this result. □

Using this homomorphism, we obtain the following important results: [10, Corollary 5.5]

Theorem 3.28. *Let (a, b) be coprime integers and $\mathfrak{p}$ a prime ideal in $\mathbb{Z}[\theta]$.*
If $\mathfrak{p}$ is not a first-degree prime ideal, then $\mathfrak{l}_{\mathfrak{p}}(a + b\theta) = 0$.

Proof. Let $\mathfrak{p}$ be a prime ideal of $\mathbb{Z}[\theta]$ with $\mathfrak{l}_{\mathfrak{p}}(a + b\theta) > 0$. As in Theorem 3.22, there exists an epimorphism $\phi : \mathbb{Z}[\theta] \to \mathbb{Z}[\theta]/\mathfrak{p}$ with $ker(\phi) = \mathfrak{p}$.

Then $\mathbb{Z}[\theta]/\mathfrak{p} \cong \mathbb{F}_{p^e}$ by Theorem 3.21, where $\mathbb{F}_{p^e}$ is the finite field with p^e elements and p prime. So $\mathbb{Z}[\theta]/\mathfrak{p}$ contains an isomorphic copy of $\mathbb{Z}/p\mathbb{Z}$.
$\mathfrak{l}_{\mathfrak{p}}(a + b\theta) > 0 \implies \mathfrak{p} \mid \langle a + b\theta\rangle$ by point 2 of Theorem 3.27, so $(a + b\theta) \in \mathfrak{p}$.
Then $ker(\phi) = \mathfrak{p} \implies \phi(a + b\theta) \equiv 0 \pmod{p}$.

Suppose for a contradiction, $p \mid b$, then $\phi(b) \equiv 0 \pmod{p}$ and we have:
$0 \equiv \phi(a + b\theta) \equiv a + b \cdot \phi(\theta) \equiv a \pmod{p}$, so that $p \mid a$.
This is a contradiction as $(a, b) = 1$ by assumption. So $p \nmid b$, which implies that b has an inverse mod p.

$0 \equiv a + b \cdot \phi(\theta) \pmod{p} \implies b \cdot \phi(\theta) \equiv -a \pmod{p}$.
Then $bb^{-1} \cdot \phi(\theta) \equiv -ab^{-1} \pmod{p}$ with b^{-1} the inverse of b mod p.
So $\phi(\theta) \equiv -ab^{-1} \pmod{p}$, which implies $\phi(\theta) \in \mathbb{Z}/p\mathbb{Z}$ and $Im\ \phi = \mathbb{Z}/p\mathbb{Z}$.

$\mathbb{Z}[\theta]/ker(\phi) \cong Im\ \phi$ and $ker(\phi) = \mathfrak{p} \implies \mathbb{Z}[\theta]/\mathfrak{p} \cong \mathbb{Z}/p\mathbb{Z}$, so $\mathfrak{p}$ is a first-degree prime ideal of $\mathbb{Z}[\theta]$ by Theorem 3.22. [8, Theorem 3.1.9] □

Theorem 3.29. *Let (a, b) be coprime integers and $\mathfrak{p}$ a prime ideal in $\mathbb{Z}[\theta]$.*
If $\mathfrak{p}$ is a first-degree prime ideal, corresponding to a pair (r, p), then $\mathfrak{l}_{\mathfrak{p}}(a+b\theta) = e_{p,r}(a+b\theta)$.

Proof. Let $\mathfrak{l}_{\mathfrak{p}}(a + b\theta) > 0$. Then $\mathfrak{p} \mid \langle a + b\theta\rangle$ by Theorem 3.27 and $(a + b\theta) \in \mathfrak{p}$.
Consider the epimorphism $\phi : \mathbb{Z}[\theta] \to \mathbb{Z}/p\mathbb{Z}$, with $\phi(\theta) = r \pmod{p}$ and $\phi(a) = a \pmod{p}$ for $a \in \mathbb{Z}$. Then $ker(\phi) = \mathfrak{p}$. $(a + b\theta) \in \mathfrak{p} \implies \phi(a + b\theta) \equiv 0 \pmod{p}$.
Then $\phi(a + b\theta) \equiv a + br \equiv 0 \pmod{p}$.

Let $a + br \equiv 0 \pmod{p}$ for the first-degree prime ideal $\mathfrak{p}$, represented by the pair (r, p).
Then $a + br \pmod{p} \equiv 0 \equiv \phi(a + b\theta)$ and $a + b\theta \in ker(\phi)$. So $a + b\theta \in \mathfrak{p}$.
$a + b\theta \in \mathfrak{p} \implies \mathfrak{p} \mid \langle a + b\theta\rangle$, so $\mathfrak{l}_{\mathfrak{p}}(a + b\theta) > 0$ by point 2 of Theorem 3.27.

We have shown $\mathfrak{l}_\mathfrak{p}(a+b\theta) > 0 \iff a+br \equiv 0 \pmod{p}$.

We now show for a first-degree prime ideal $\mathfrak{p}$ represented by pair (r,p), that
$p \mid N(a+b\theta) \iff a+br \equiv 0 \pmod{p}$. Then $\mathfrak{l}_\mathfrak{p}(a+b\theta) = e_{p,r}(a+b\theta)$ as required.

Let $\sigma_1,\ldots,\sigma_d$ be the d monomorphisms $\sigma_i : \mathbb{Q}(\theta) \to C$ of Theorem 3.15.
By Definition 3.16:

$$\begin{aligned} N(a+b\theta) &= \sigma_1(a+b\theta)\cdot\sigma_2(a+b\theta)\cdots\sigma_d(a+b\theta) \\ &= b^d\left((\frac{a}{b}+\theta_1)\cdot(\frac{a}{b}+\theta_2)\cdots(\frac{a}{b}+\theta_d)\right) \\ &= (-b)^d\left((-\frac{a}{b}-\theta_1)\cdot(-\frac{a}{b}-\theta_2)\cdots(-\frac{a}{b}-\theta_d)\right) \\ &= (-b)^d f(-\frac{a}{b}) \end{aligned}$$

So $p \mid N(a+b\theta) \iff p \mid f(-\frac{a}{b})$ as $p \nmid b$.
$p \mid f(-\frac{a}{b}) \implies f(-\frac{a}{b}) \equiv 0 \pmod{p}$, so $a \equiv -br \pmod{p}$ for r with $f(r) \equiv 0 \pmod{p}$. Thus the (r,p) pair determines a first-degree prime ideal $\mathfrak{p}$.

To show uniqueness of the (r,p) pair corresponding to $\mathfrak{p}$, let $\mathfrak{l}_{\mathfrak{p}_1}(a+b\theta) > 0$ and $\mathfrak{l}_{\mathfrak{p}_2}(a+b\theta) > 0$ for first-degree prime ideals $\mathfrak{p}_1$ & $\mathfrak{p}_2$, represented by pairs (r_1,p) and (r_2,p) respectively.
Then $a+br_1 \equiv 0 \pmod{p}$ and $a+br_2 \equiv 0 \pmod{p} \implies r_1 \equiv r_2 \pmod{p}$.
So $\mathfrak{p}_1 = \mathfrak{p}_2$ and the (r,p) pair with $\mathfrak{l}_\mathfrak{p}(a+b\theta) > 0$ is unique. [8, Theorem 3.1.9] □

Applying Theorems 3.28 & 3.29, we have a way of determining whether an ideal factors over the algebraic factor base; namely, it factors if $N(a+b\theta)$ factors completely into primes p_i of the (r,p) pairs that represent the first-degree prime ideals. Such an element $(a+b\theta)$ is smooth over the algebraic factor base.

Now that we can find smooth values of $(a+b\theta)$, we need to consider how to find a square $\beta^2 \in \mathbb{Z}[\theta]$ to satisfy the congruence (3.2). Unfortunately, the usual approach of forming a product with only even exponents gives a necessary, but not sufficient, condition for constructing a square $\beta^2 \in \mathbb{Z}[\theta]$ (See Proposition 5.3 of [10]). This process ensures that $N(\prod(a+b\theta))$ is a square, which is not a strong enough condition to guarantee that $\prod(a+b\theta)$ itself is a square in $\mathbb{Z}[\theta]$. We consider the problem of finding $\beta^2 \in \mathbb{Z}[\theta]$ in the next section.

3.3.3 Quadratic characters

Our issue arises, as mentioned previously, as we are working with ideals of $\mathbb{Z}[\theta]$ and not of O_K (See §6 of [10] for further details). This problem was resolved by a proposal of Adleman.

[35] Adleman suggested the use of first-degree primes not in the factor base as a means of heuristically testing $\prod(a+b\theta)$ for squareness in $\mathbb{Z}[\theta]$.

Recall that our ring homomorphism in (3.2) is defined for $\beta \in \mathbb{Z}[\theta]$. In practice, we allow the product $\prod(a+b\theta)$ to be square in $\mathbb{Q}(\theta)$, rather than only $\mathbb{Z}[\theta]$, as this is less restrictive. By a clever use of the Legendre symbol, we can test the product for squareness in $\mathbb{Z}[\theta]$. We first require a lemma of [10, 6.6]:

Lemma 3.30. *Let S be as in* (3.1) *and* $\prod_{(a,b)\in S}(a+b\theta) = \alpha^2$ *for some* $\alpha \in \mathbb{Q}(\theta)$.
Let f be a monic, irreducible polynomial of degree d with integer coefficients, and a root $\theta \in \mathbb{C}$. Then $\alpha \in O_{\mathbb{Q}(\theta)}$ and $f'(\theta)^2 \cdot \prod_{(a,b)\in S}(a+b\theta)$ is a square in $\mathbb{Z}[\theta]$.

Theorem 3.31. *Let S be a set of pairs of coprime integers (a,b) with $\prod_{(a,b)\in S}(a+b\theta) = \alpha^2$ for some $\alpha^2 \in \mathbb{Q}(\theta)$. Let q be an odd prime and $s \in R(q)$ with R the set in Definition 3.26, such that:*

- *$a+bs \neq 0$ (mod q) for each $(a,b) \in S$*
- *$f'(s) \neq 0$ (mod q)*

Then:

$$\prod_{(a,b)\in S}\left(\frac{a+bs}{q}\right) = 1 \tag{3.6}$$

Proof. Let $\mathfrak{q}$ be the kernel of the ring homomorphism $\phi : Z[\alpha] \to \mathbb{F}_q$ which maps θ to s (mod q). Then $\mathfrak{q}$ corresponds to the pair (s,q).

Define $\chi_{\mathfrak{q}} : \mathbb{Z}[\theta] - \mathfrak{q} \to \{\pm 1\}$ as the composition of $\mathbb{Z}[\theta] - \mathfrak{q} \to \mathbb{F}_q - \{0\}$ with the Legendre symbol $\mathbb{F}_q - \{0\} \to \{\pm 1\}$.

Then $\chi_{\mathfrak{q}}(a+b\theta) = (\frac{a+bs}{q})$. We have by the previous lemma that $f'(\theta)^2 \cdot \prod_{(a,b)\in S}(a+b\theta) = \delta^2 \in \mathbb{Z}[\theta]$ for some δ.

As the factors are not in $\mathfrak{q}, \delta \notin \mathfrak{q}$, then applying $\chi_{\mathfrak{q}}$ gives the result. [10, Proposition 8.3] □

To put this theorem to use, we essentially generate a further set P_Q of first-degree prime ideals not in the algebraic factor base.

Definition 3.32. The **quadratic character base** P_Q is a set of first-degree prime ideals, represented by (s, q) pairs with $p_A < q_i$, where p_A is the largest prime in the (r, p) pairs of the algebraic factor base.

By checking that (3.6) holds across all (s, q) pairs in P_Q, it is highly likely that $\prod_{(a,b)\in S}(a+b\theta)$ is a square. By adding more (s, q) pairs to P_Q, we can increase this probability.

To summarise , we have developed methods for testing whether an (a, b) pair is smooth over the algebraic factor base, and can test whether the product of such pairs $\prod_{(a,b)\in S}(a+b\theta)$ is a likely square in $\mathbb{Z}[\theta]$. In the next section, we outline how to apply these ideas to construct the algebraic sieve.

3.4 Sieving

3.4.1 Technical implementation of the sieve

To sieve in $\mathbb{Z}[\theta]$, we use a procedure almost identical in form to the sieving process in $\mathbb{Z}$ outlined in §3.3.1. By Theorem 3.29, $N(a + b\theta)$ factorises over the primes of $(r, p) \in P_A$ if and only if, $a + br \equiv 0 \pmod{p}$.

Note that $\mathfrak{p} \mid \langle a + b\theta\rangle \iff a \equiv -br \pmod{p}$. Then $a = -br + kp$ for some $k \in \mathbb{Z}$.

To sieve, we again fix b and sieve over an interval $[-u \leq a \leq u]$. Once the positions are calculated, we initialise the sieve with the values $N(a + b\theta)$ and divide by p at the a_ith positions where $a_i = -br + kp$. We then increment b and the process is repeated until enough smooth values have been obtained.

Sieving in $\mathbb{Z}[\theta]$ is thus ostensibly identical to sieving in $\mathbb{Z}$, which simplifies computer implementation of the GNFS. One further point of implementation is to store the elements of P_R as pairs of integers $(m \pmod{p},\ p)$, to align with the form of elements of P_A. This further simplifies the sieving code. [8, §3.8]

3.4.2 The Lanczos algorithm

Once we have found enough simultaneously smooth values in $\mathbb{Z}[\theta]$ and $\mathbb{Z}$, we need to perform the linear algebra step to find a set S such that (3.1) holds. Due to the large size of the factor base used in a GNFS factorisation, the exponent matrix becomes increasingly unwieldy to work with. To give an idea of the size of this matrix, we first show the form of a binary vector v_i representing an (a, b) pair with $(a + bm)$ and $(a + b\theta)$ smooth over their respective factor bases.

$$v_i = \underbrace{0}_{\text{Sign bit}} \overbrace{0\,1\,0\,0\,0\,1\,\ldots\,0}^{\text{Exponents of } \mathrm{p_i \in P_R}} \underbrace{1\,1\,0\,1\,1\,0\,0\,0\,\ldots\,1}_{\text{Exponents of } \mathrm{p_i \in P_A}} \overbrace{0\,0\,1\,1\,0\,0\,\ldots\,0}^{\text{Quadratic character bits}}$$

With the sign bit $=\begin{cases} 0 & \text{if } a + bm > 0 \\ 1 & otherwise \end{cases}$

To preserve multiplicity, for each $(s, q) \in P_Q$, the corresponding quadratic character bit is set as follows: P_Q bit $=\begin{cases} 0 & \text{if } \frac{a+bs}{q} = 1 \\ 1 & otherwise \end{cases}$

Therefore, if there are k primes in F_R, l first-degree prime ideals in F_A, and m first-degree prime ideals in F_Q, the vector will consist of $1 + k + l + m$ bits. [8] For this reason, rather than using Gaussian elimination to find a dependency, the block Lanczos algorithm is used. This algorithm is particularly suited to dealing with a large, sparse matrix.

For further details on the block Lanczos algorithm, see [30].

3.5 Calculating the square root

After sieving and linear algebra processing, we obtain a set S of (a, b) pairs satisfying (3.1). One final issue is that in order to calculate $(x \pm y, n)$ and obtain a factorisation, we need to calculate $x = \phi(\beta)$.

We know that $\beta \in \mathbb{Z}[\theta]$, so it may be written $\beta = a_{d-1}\theta^{d-1} + a_{d-2}\theta^{d-2} + \cdots + a_1\theta + a_0$. Letting $x = a_{d-1}m^{d-1} + a_{d-2}m^{d-2} + \cdots + a_1 m + a_o$, then satisfies the congruence (3.2). [8] As $\beta^2 \in \mathbb{Z}[\theta] = \prod_{(a,b)\in S} (a+b\theta)$, β^2 is the product of a very large number of algebraic numbers, so calculating the square root β directly is difficult.

There are a number of solutions to this problem, as discussed in [43]. We shall sketch one of the earlier ideas, proposed by Couveignes in [12]. By a clever use of the Chinese Remainder theorem, we can circumvent the difficulties of finding β and calculate x without requiring intermediate values.

Theorem 3.33. ***The Chinese Remainder theorem:***
Let $p_1, p_2, \ldots, p_k$ be distinct primes and $x_1, x_2 \ldots, x_k$ any integers.
Set $P = \prod_{i=1}^{k} p_i$, $P_i = P/p_i$, and $a_i = P_i^{-1}$ (mod p_i).
For the congruences:

$$\begin{aligned} z &\equiv x_1 \ (mod\ p_1) \\ z &\equiv x_2 \ (mod\ p_2) \\ &\vdots \\ z &\equiv x_k \ (mod\ p_k) \end{aligned}$$

$z = \sum_{i=1}^{k} a_i x_i P_i$ is the unique solution modulo P.

Proof. See [20, Theorem 3.10] □

Our approach is to calculate x modulo primes p_i, which can be done efficiently by the Shanks-Tonelli algorithm, amongst other methods. If we use enough primes such that their product $P > x$, and solve the system of congruences by Theorem 3.33, then we find $x \equiv z \pmod{P}$.

By rounding z/P to the nearest integer $r = \lfloor \frac{z}{P} + \frac{1}{2} \rfloor$, we have $x = z - rP$. [12] We can calculate r efficiently without needing to calculate z directly by the following:

$$\begin{aligned} \frac{z}{P} &= \frac{\sum_{i=1}^{k} a_i x_i P_i}{P} \\ &= \frac{\sum_{i=1}^{k} a_i x_i \frac{P}{p_i}}{P} \\ &= \sum_{i=1}^{k} \frac{a_i x_i}{p_i} \end{aligned}$$

So $r = \lfloor (\sum_{i=1}^{k} \frac{a_i x_i}{p_i}) + \frac{1}{2} \rfloor$. We now calculate $x \pmod{n}$ as follows: [8]

$$\begin{aligned} x \ (\mathrm{mod}\ n) &= z - rP \ (\mathrm{mod}\ n) \\ &= z \ (\mathrm{mod}\ n) - rP \ (\mathrm{mod}\ n) \\ &= \sum_{i=1}^{k} a_i x_i P_i \ (\mathrm{mod}\ n) - rP \ (\mathrm{mod}\ n) \\ &= \sum_{i=1}^{k} a_i x_i P_i \ (\mathrm{mod}\ n) - \lfloor (\sum_{i=1}^{k} \frac{a_i x_i}{p_i}) + \frac{1}{2} \rfloor P \ (\mathrm{mod}\ n) \end{aligned}$$

In this manner, none of the intermediate values calculated will exceed n and x can be found efficiently. For further details on calculating $x \pmod{p_i}$, see [12] and [8, §4.7 & §4.8].

Once x has been computed, we calculate the square root of $y^2 = \prod_{(a,b)\in S} (a+bm)$ and generate our required congruence (3.2). Finally, we calculate $(x \pm y, n)$ and the factorisation of n is complete.

Chapter 4

Time complexity

4.1 Big O notation

In order to compare the efficiency of the algorithms presented in previous chapters, we now outline their time complexities. To measure time complexity, we consider the algorithm's asymptotic behaviour, that is, the number of elementary operations required to complete the algorithm in the worst-case scenario. By design, an RSA modulus n, being a large semiprime with no small prime factors, represents the worst-case for factorising.

To describe the time complexity of an algorithm, we use Big O notation. [39, §7]

Definition 4.1. Big O notation:
Let f be a function $f : \mathbb{C} \to \mathbb{C}$ and g a real valued function.

Then we say $f(x) \in O(g(x))$ as $x \to \infty$ if $\exists C, x_0 \in \mathbb{R}$ such that $|f(x)| \leq Cg(x)$ for all $x \geq x_0$.

For the remainder of the chapter, let $f(x)$ represent the number of elementary operations of an algorithm as a function of the size of the input. When working with large inputs, the highest order of the expression for running time dominates the other terms, so it is this term that describes the asymptotic behaviour.

Example 4.2. The total number of steps taken to complete an algorithm with input x is $f(x) = 7x^5 + 3x^3 + 2x + 1$.

The x^5 term will dominate the runtime for large inputs, with the constant coefficient 7 having relatively minimal effect.

We write $f(x) \in O(x^5)$.

Some common time complexities are shown in Figure 4.1 below:

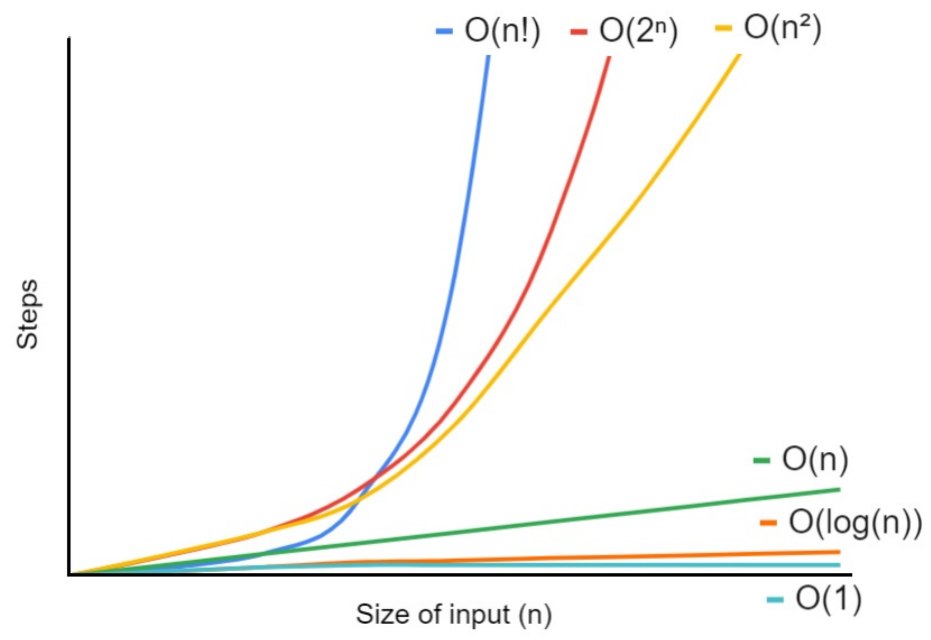

Figure 4.1: Comparison of common time complexities

As the runtime grows with the size of the input, the theoretical time complexity of the algorithm translates directly to the amount of real-world computing time required to run the algorithm.

Definition 4.3. We say that a problem is **tractable** [42, §2.4] if there is an algorithm to solve it that runs in polynomial time or faster; i.e. $f(x) \in O(x^n)$ for some $n \in \mathbb{N}$.

It is important to note that an expression for the time complexity of an algorithm is dependent upon the encoding of the input. [42] For our purposes, the RSA modulus n is encoded in binary and therefore the input has size $\log_2(n)$ bits. So a function $f(n) \in O(n)$ is exponential in the size of the input.

We now outline the time complexities of the algorithms discussed in §2. We begin with the exponential time algorithms.

4.2 Exponential time algorithms

Definition 4.4. An algorithm has **exponential** running time if $f(x) \in O(2^{x^k})$ for some constant k.

4.2.1 Trial division

As we may need to divide by all prime numbers up to $\sqrt{n}$, the worst case running-time for trial division depends on the number of primes less than n. This number is given by the prime-counting function $\pi(x)$, which by the prime number theorem is asymptotically:

Theorem 4.5. *Prime number theorem:*

$$\pi(x) \sim \frac{x}{ln\ x}$$

Proof. See §22 of [18]. □

For an r-bit number, we need to check up to $\pi(2^{r/2})$ primes. By Theorem 4.5:

$$\begin{aligned}\pi(2^{r/2}) &\approx \frac{2^{r/2}}{ln\ 2^{r/2}} \\ &\approx \frac{2^{r/2}}{\frac{r}{2} ln\ 2}\end{aligned}$$

So we need to make approximately $\frac{2^{r/2}}{\frac{r}{2} ln\ 2}$ trial divisions, so $f(x) \in O(2^{\frac{r}{2}})$, which is exponential in the size of the input.

4.2.2 Fermat's method

In the worst case, Fermat's method will take up to $\left(\frac{n+1}{2} - \sqrt{n}\right)$ steps to find a square and generate a factor. [45]

Fermat's method therefore has time complexity $O(\sqrt{n})$, which is also exponential in the size of the input.

4.2.3 Euler's method

Provided n does not satisfy the conditions of Theorem 2.8, it can be factorised using Euler's method in time $O(\sqrt{n})$.

4.2.4 Pollard's rho algorithm

By considering the Birthday Paradox, for a prime p dividing n we would expect a sequence of $f(x_i)$'s to cycle before $O(\sqrt{p})$ steps. [13, §5.2] This is only conjectured, as it is based on the assumption that the $f(x_i)$ values behave like random numbers. Pollard's rho thus has time complexity $O(\sqrt{p}) \leq O(\sqrt[4]{n})$, so is heuristically exponential in the size of the input.

For a more rigorous analysis of Pollard's rho algorithm, see [1].

4.3 Subexponential time algorithms

We now consider the subexponential time algorithms, which are asymptotically faster than the previous algorithms. We first require some additional notation.

Definition 4.6. Little-o notation:

Let f be a function $f : \mathbb{C} \to \mathbb{C}$ and g a real valued function.

Then we say $f(x) \in o(g(x))$ as $x \to \infty$ if $\forall \varepsilon \in \mathbb{R}^+, \exists C$ such that $|f(x)| \leq \varepsilon g(x)$ for all $x \geq C$.

Little-o notation is used to show that a function $f(x)$ grows much more slowly than some function $g(x)$.

Definition 4.7. An algorithm has **subexponential** running time if $f(x) \in 2^{o(n)}$ where $o(n)$ represents little-o notation as in Definition 4.6.

4.3.1 Dixon's algorithm

Dixon's algorithm was the first subexponential algorithm with a rigorous proof of its runtime, hence its primacy in the discussion of §2.

Proposition 4.8. *Dixon's method has expected runtime* $O\left(e^{\left((3\sqrt{2})\sqrt{\ln n \ln \ln n}\right)}\right)$

We outline the derivation of this runtime with a sketch proof.

Suppose we have a list of integers L of length N in the range $[1, n]$. We seek values x_i in this range such that $f(x_i) = x_i^2 \pmod{n}$ is B-smooth. We choose $x_i \in L$ randomly and factorise $x^2 \pmod{n}$ to determine whether it is B-smooth. This factorisation step takes $O(B \ln n)$ operations. [15] Once B+1 B-smooth values are found, we proceed to the linear algebra step. Gaussian elimination requires $O(B^3)$ operations. The final step of calculating a gcd requires $O(\ln n)$ operations.

Thus, the method can be reduced to an optimisation problem in which we select our bound B on the factor base to minimise the number of $f(x_i)$'s that need to be checked for smoothness. We must also optimise the length N of our list L such that we are almost certain to find enough smooth values of $f(x_i)$.

Proposition 4.9. *Set* $B = e^{\sqrt{(2 \ln n \ln \ln n)}}$ *and* $N = (B^2 + 1)$*. Then the average number of operations to factorise by Dixon's algorithm is* $O\left(e^{\left((3\sqrt{2})\sqrt{\ln n \ln \ln n}\right)}\right)$.

Proof. See §3 of [15]. □

As the general-purpose algorithms we have discussed may be considered optimisations of Dixon's algorithm, the derivation of their runtimes is similar. One major heuristic assumption is that the numbers that are checked for smoothness in each algorithm (Pomerance calls these *'auxiliary numbers'*) have the same probability of being smooth as a randomly generated number of the same size. This may not necessarily be the case, as our generating function may be particularly good (or bad) at generating smooth values. Thus, the key to speeding up algorithms of this form is to generate smaller auxiliary numbers, which are therefore more likely to be smooth, allowing relations to be found faster.

4.3.2 Continued Fraction method

Although the Continued Fraction method was the first subexponential algorithm, its runtime has not been rigorously proved. Heuristically, the Continued Fraction method has expected runtime $O\left(e^{\left(\sqrt{2\ln n \ln\ln n}\right)}\right)$. [45]

4.3.3 The Quadratic Sieve

The Quadratic Sieve improves upon earlier methods by checking for smoothness far more quickly using the sieving process discussed in §2.3.4. This improves its expected runtime to $O\left(e^{\left(\sqrt{\ln n \ln\ln n}\right)}\right)$. [35]

4.3.4 The General Number Field Sieve

The major speedup of the GNFS comes through generating auxiliary numbers bounded by $e^{c'(\ln n)^{\frac{1}{3}}(\ln\ln n)^{\frac{2}{3}}}$ with c' a constant. [35] This compares favourably with the auxiliary numbers used in the Quadratic Sieve, which are around the size $\sqrt{n}$.

This gives the GNFS expected running time $O\left(e^{c(\ln n)^{\frac{1}{3}}(\ln\ln n)^{\frac{2}{3}}}\right)$ with $c = \sqrt[3]{\frac{64}{9}}$. [35]

See [28] for further discussion on the heuristic runtime of the GNFS.

Chapter 5

Summary

The introduction of the RSA cryptosystem has stimulated significant research into a canonical 'hard' problem. Since then, Legendre's congruence has been fruitfully mined to produce the current fastest known algorithm, the General Number Field Sieve. In 2009, Kleinjung et al. factorised the 768-bit number RSA-768 using the GNFS. The sieving process was distributed, taking almost two years, and the entire computation took more than 10^{20} operations. The team optimistically predicted that a 1024-bit modulus could be factored *'...well within the next decade.'* [23].

Although their prediction was not borne out, the factorisation of RSA-250, an 829-bit number, was announced on February 28th, 2020. [46] With the consistent increase in computational power, such records will continue to fall. However, without a further theoretical breakthrough, ever larger keys should guarantee the security of RSA for the foreseeable future. While the time complexity of our best algorithms remains non-polynomial, the problem of factorising a large integer remains insurmountable.

One potential challenge to the security of RSA lies in recent developments in quantum computing. In 1994, Peter Shor published a quantum algorithm that factorises n in $O\left((\log n)^{2+\epsilon}\right)$. [45] However, this polynomial-time algorithm remains theoretical until the development of viable quantum computers. History has shown that even the most seemingly impregnable cryptosystems eventually succumb to the determined cryptanalyst. Given that the RSA problem is well-defined and understood, it seems inevitable that it too will eventually be broken. The question remains: will this be via a classical theoretical, or a quantum technological, breakthrough?

Bibliography

[1] Bach, E. (1991). *Toward a theory of Pollard's rho method.* Information and Computation, 90(2), 139-155.

[2] Barker, E., & Dang, Q. (2015). *NIST special publication 800–57 part 3: Application-specific key management guidance.* NIST Special Publication, 800, 57.

[3] Bhaskar, J. (2008). *Sum of two squares.* University of Chicago (REU 2008).

[4] Brent, R. P., & Pollard, J. M. (1981). *Factorization of the eighth Fermat number.* Mathematics of Computation, 36(154), 627. https://doi.org/10.1090/s0025-5718-1981-0606520-5

[5] Brent, R. P. (1980). *An improved Monte Carlo factorization algorithm.* BIT, 20(2), 176–184. https://doi.org/10.1007/bf01933190

[6] Brent, R. P. (1999). *Factorization of the tenth Fermat number.* Mathematics of Computation, 68(225), 429–452. https://doi.org/10.1090/s0025-5718-99-00992-8

[7] Brent, R. P. (2000). *Recent progress and prospects for integer factorisation algorithms.* In International Computing and Combinatorics Conference (pp. 3-22). Springer, Berlin, Heidelberg.

[8] Briggs, M. E. (1998). *An introduction to the general number field sieve* (Dissertation, Virginia Tech).

[9] Brillhart, J. (2009). *A note on Euler's factoring problem.* The American Mathematical Monthly, 116(10), 928-931.

[10] Buhler, J. P., Lenstra, H. W., & Pomerance, C. (1993). *Factoring integers with the number field sieve.* In The development of the number field sieve (pp. 50-94). Springer, Berlin, Heidelberg.

[11] da Costa Boucinha, F. (2011). *A survey of cryptanalytic attacks on RSA.* Instituto superior técnico, Universidade Técnica de Lisboa.

[12] Couveignes, J. M. (1993). *Computing a square root for the number field sieve.* In The development of the number field sieve (pp. 95-102). Springer, Berlin, Heidelberg.

[13] Crandall, R., & Pomerance, C. B. (2010). *Prime Numbers: A Computational Perspective* (2nd ed.). Springer.

[14] Diffie, W., & Hellman, M. (1976). *New directions in cryptography.* IEEE transactions on Information Theory, 22(6), 644-654.

[15] Dixon, J. D. (1981). *Asymptotically fast factorization of integers.* Mathematics of Computation, 36(153), 255. https://doi.org/10.1090/s0025-5718-1981-0595059-1

[16] Galbraith, S. (2012). *The Mathematics of Public Key Cryptography.* Cambridge: Cambridge University Press.

[17] Gardner, M. (1977, August). *Mathematical Games.* In Scientific American. New York: Munn & Co.

[18] Hardy, G. H., & Wright, E. M. (1979). *An introduction to the theory of numbers.* Oxford University Press.

[19] Jensen, P. L. (2005). *Integer factorization.* University of Copenhagen.

[20] Jones, G. A., & Jones, J. M. (1998). *Elementary Number Theory (Springer Undergraduate Mathematics Series)* (Corrected ed.). Springer.

[21] Khinchin, Y. A. (1997). *Continued Fractions* (Dover Books on Mathematics) (Revised ed.). Dover Publications.

[22] Kleinjung, T. (2006). *On polynomial selection for the general number field sieve.* Mathematics of Computation, 75(256), 2037–2047. https://doi.org/10.1090/s0025-5718-06-01870-9

[23] Kleinjung, T., Aoki, K., Franke, J., Lenstra, A. K., Thomé, E., Bos, J. W., ... & Zimmermann, P. (2010, August). *Factorization of a 768-bit RSA modulus.* In Annual Cryptology Conference (pp. 333-350). Springer, Berlin, Heidelberg.

[24] Lehman, R. S. (1974). *Factoring large integers.* Mathematics of Computation, 28(126), 637.

[25] Lehmer, D. H., & Powers, R. E. (1931). *On factoring large numbers.* Bulletin of the American Mathematical Society, 37(10), 770–777. https://doi.org/10.1090/s0002-9904-1931-05271-x

[26] Lenstra, A. K., Lenstra, H. W., Manasse, M. S., & Pollard, J. M. (1993). *The factorization of the ninth Fermat number.* Mathematics of Computation, 61(203), 319-349.

[27] Lenstra, A. K., Lenstra, H. W., Manasse, M. S., & Pollard, J. M. (1993). *The number field sieve.* In The development of the number field sieve (pp. 11-42). Springer, Berlin, Heidelberg.

[28] Lenstra, A. K. (2000). *Integer factoring.* Towards a quarter-century of public key cryptography, 31-58.

[29] Lenstra, H. W. (1987). *Factoring Integers with Elliptic Curves.* The Annals of Mathematics, 126(3), 649. https://doi.org/10.2307/1971363

[30] Montgomery, P. L. (1995, May). *A block Lanczos algorithm for finding dependencies over GF (2).* In International Conference on the Theory and Applications of Cryptographic Techniques (pp. 106-120). Springer, Berlin, Heidelberg.

[31] Morrison, M. A., & Brillhart, J. (1975). *A Method of Factoring and the Factorization of F_7.* Mathematics of Computation, 29(129), 183. https://doi.org/10.2307/2005475

[32] Murphy, B. A. (1999). *Polynomial selection for the number field sieve integer factorisation algorithm.* Australian National University.

[33] Pollard, J. M. (1975). *A monte carlo method for factorization.* BIT, 15(3), 331–334. https://doi.org/10.1007/bf01933667

[34] Pomerance C. (1985). *The Quadratic Sieve Factoring Algorithm.* In: Beth T., Cot N., Ingemarsson I. (eds) Advances in Cryptology. EUROCRYPT 1984. Lecture Notes in Computer Science, vol 209. Springer, Berlin, Heidelberg.

[35] Pomerance, C. (1996). *A tale of two sieves.* In Notices Amer. Math. Soc.

[36] *Progress in general purpose factoring.* (2017, October 8). AI Impacts. https://aiimpacts.org/progress-in-general-purpose-factoring/

[37] 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.

[38] Silverman, R. D. (1987). *The multiple polynomial quadratic sieve.* Mathematics of Computation, 48(177), 329. https://doi.org/10.1090/s0025-5718-1987-0866119-8

[39] Sipser, M. (2014). *Introduction to the Theory of Computation* (3rd ed.). Cengage India.

[40] Stewart, I., & Tall, D. (2001). *Algebraic Number Theory and Fermat's Last Theorem: Third Edition* (3rd ed.). A K Peters/CRC Press.

[41] Stinson, D. R., & Paterson, M. (2018). *Cryptography: Theory and Practice* (Textbooks in Mathematics) (4th ed.). Chapman and Hall/CRC.

[42] Talbot, J., and Welsh, D. (2006). *Complexity and Cryptography: An Introduction* (1st ed.). Cambridge University Press.

[43] Thomé, E. (2012, July). *Square root algorithms for the number field sieve.* In International Workshop on the Arithmetic of Finite Fields (pp. 208-224). Springer, Berlin, Heidelberg.

[44] Tornaría, G. (2002, April). *Square roots modulo p.* In Latin American Symposium on Theoretical Informatics (pp. 430-434). Springer, Berlin, Heidelberg.

[45] Yan, S. Y. (2010). *Primality Testing and Integer Factorization in Public-Key Cryptography (Advances in Information Security, 11)* Springer.

[46] Zimmerman, P. (2020) *Factorisation of RSA-250.* https://listserv.nodak.edu/cgi-bin/wa.exe?A2=NMBRTHRY;dc42ccd1.2002