الجـمـهـوريـة الجـزائـريـة الديمـقــراطـيــة الــشـعــبـيــة
REPUBLIQUE ALGERIENNE DEMOCRATIQUE ET POPULAIRE
وزارة التـعـليــم العـالي والبـحـث العــلــمــي
Ministère de l'Enseignement Supérieur et de la Recherche Scientifique
جامعة أبي بــكـــر بـلـــقـــايــد – تــلـمـســان –
Université Aboubakr Belkaïd – Tlemcen –
Faculté de TECHNOLOGIE

MEMOIRE

Présenté pour l'obtention du **diplôme** de **MASTER**

En : Télécommunications

Spécialité : Réseaux et Télécommunications

Par : MEKKIOUI Abdallah Salah Eddine & MIR Zakaria

Sujet

Performance study and evaluation of the GNFS factorization algorithm

Etude et évaluation des performances de l'algorithme de factorisation GNFS

Soutenu publiquement, le 18/06/2023 , devant le jury composé de :

Mr BOUSAHLA Miloud	Maitre de Conférences	Université de Tlemcen	Président
Mme SLIMANE Zohra	Maitre de Conférences	Université de Tlemcen	Examinateur
Mr ABDELMALEK Abdelhafid	Maitre de Conférences	Université de Tlemcen	Encadreur

Année universitaire : 2022/2023

Acknowledgments

I want to dedicate this work to my mother for her unconditioned help and support to my family and my Siblings . To my Partner and friend which was a pleasure working with and also to all my friends. As i would like to thank my professor Mr.Abdelmalek Abd el haffid for his great help and guidance that he has provided all along our work on this thesis.

Mekkioui Abd Allah

Praise be to God who helped me complete this thesis and helped me in my scientific path.
I dedicate this thesis to my parents for their endless love and support throughout my pursuit for education To my master's promotion in Networks and Telecommunications To our supervisor professor for his compassionate and To my not only colleague at this work but a dear friend that working with was an absolute pleasure To all those I have not mentioned their names but I never forget their helpers .

Mir Zakaria

Abstract

Numeric Security is the topic of the century ,To protect the secrecy of sensitive numerical data during transmission and storage, it requires the use of powerful encryption techniques. Strong authentication and access controls also aid in preventing unauthorized parties from obtaining private numerical data. Numeric data integrity and security must be regularly monitored, audited, and compliant with industry standards to safeguard people and businesses from potential financial loss, identity theft, and other cyberthreats.

This thesis study the integers factorization since Many public key cryptosystems depend on the difficulty of factoring large integers,but first we will be going through cryptography from the beginning and tracing it development until arriving to the most efficient algorithm for factorization wish is the GNFS algorithm

key words:Cryptography/security/Algorithms/Factorization /public key

ملخص

الأمن الرقمي هو موضوع القرن، ولحماية سرية البيانات العددية الحساسة أثناء النقل والتخزين، يتطلب استخدام تقنيات التشفير القوية. تساعد التحقق القوي وضوابط الوصول أيضًا في منع الأطراف غير المصرح لها من الحصول على البيانات رقمية الخاصة. يجب مراقبة سلامة البيانات العددية وأمنها بشكل منتظم، وإجراء التدقيق، والإمتثال للمعايير الصناعية لحماية الأفراد والأعمال من خسائر مالية محتملة وسرقة الهوية وتهديدات الأمان السيبراني الأخرى .تتناول هذه الرسالة دراسة عامة التفصيل، حيث يعتمد العديد من نظم التشفير العمومية على صعوبة عملية تفصيل الأعداد الكبيرة، ولكن أولاً سنقوم بمراجعة علم التشفير من البداية وتتبع تطوره حتى الوصول إلى الخوارزمية الأكثر كفاءة لعملية التفصيل وهي خوارزمية منخل حقل الأعداد العام .

Résumé

La sécurité numérique est le sujet du siècle ,Pour protéger le secret des données numériques sensibles lors de leur transmission et de leur stockage, elle nécessite l'utilisation d'un système de sécurité numérique.

L'intégrité et la sécurité des données numériques doivent être régulièrement contrôlées, auditées, et conformes aux normes de l'industrie afin de protéger les personnes et les entreprises des pertes financières potentielles, de l'usurpation d'identité et d'autres cyber-menaces.

Ce memoire étudie la factorisation des nombres entiers car de nombreux cryptosystèmes à clé publique dépendent de la difficulté de la factorisation des nombres entiers. mais nous allons tout d'abord étudier la cryptographie depuis ses débuts. et retracer son développement jusqu'à l'algorithme de factorisation le plus efficace, à savoir l'. Algorithme de factorisation GNFS

mot-clés: cryptographie /sécurite/Algorithms/Factorisation/clé public

Contents

List of Figures

Acronym	**Signification**
ϕ	Euler totient
DES	Data encryption standard
AES	Advanced encryrtion standard
PII	Personal identifying information
TLS	Transport layer security
SSL	Secure socket layer
GCHQ	British government communication headquarter
RSA	Encryption system by R.Rivest/A.shamir/L.Adelman
HTTPS	Hypertext Transfer Protocol Secure
UTXO	Unspent Transaction Output
DSA	Digital signature algorithm
$\equiv$	congruent
SHA	Secure Hash Algorithm access control
CCA	Chosen cipher-text attack
$\mathcal{O}(n)$	Measure for algorithm complexity
GCD	Greatest common divisor
QS	Quadratic sieve algorithm
NFS	Number field sieve
GNFS	Generam number field sieve
SNFS	Special number field sieve
AFB	Algebraic factor base
RFB	Rational factor base
QCB	Quadratic character base

Introduction:

Internet and data usage and transmission is at it's peak and it's exponentially increasing day after day along with hardware evolution and processing power and so on the increasing necessity for data encryption Throughout the years so many encryption algorithms have been developed and adopted , the most known and used cryptosystem is RSA which is a public-key cryptosystem that is widely used to secure data transmission , a cryptosystem reliability depends on it's structure and immunity against factorization algorithms .

GNFS or general number field sieve algorithm is the most efficient algorithm for factoring big composite number which is what encryption cryptosystems is built on , GNFS can factorize composites of a size up to 300 hundred digits.
The work presented falls within this framework, and aims to study the general number field sieve in all it's aspects .

This thesis consists of three chapters :
The first chapter addresses modern cryptography both symmetric and asymmetric along with their functionality , uses and applications , also we have addressed attacks on RSA and the most know factorization algorithms . The second chapter addresses the General Number Field Sieve or " GNFS " and all the steps that construct it's functionality followed by an extended example .

The last chapter is an evaluation of GNFS performances preceded by all the steps required for this evaluation This thesis ended with a general conclusion, bibliographical references are added at the end of this document.

June 24, 2023

Chapter1:Cryptography and security

1 Cryptography and security

1.1 Introduction

In this chapter , we present in the first place an Introduction of modern cryptography in addition to it's two main categories (the Symmetric and the Asymmetric cryptography) including their functionality , application and their pros and cons , further we will see attacks on RSA and factorization algorithms along with the notion of complexity .

The main goal of this chapter is to define modern cryptography and to introduce the problem of factorization .

1.1.1 Cryptography

1.1.2 Cryptography and modern cryptography

According to the Concise Oxford English Dictionary, cryptography is "the art of writing or cracking codes." The primary goal of cryptography is to permit the secure exchange of information between two entities via an insecure route that cannot be spied on by an intruder.

The adoption of classical encryption (say, before the 1980s) and current cryptography is a significant distinction.Historically, military and government institutions were the primary users of cryptography.Cryptography is now everywhere! If you've ever authenticated something, You have definitely utilized cryptography when you have typed a password, purchased something with a credit card over the Internet, or downloaded a confirmed update for your operating system. Furthermore, programmers with limited experience are increasingly being requested to "secure" the programs they build by implementing cryptographic techniques. Cryptography currently includes systems for assuring integrity, techniques for sharing secret keys, protocols for authenticating users, electronic auctions and elections, digital cash, and much more.Without attempting to provide a comprehensive definition, we may say that modern cryptography is the study of mathematical strategies for protecting digital information, systems, and distributed computations from adversarial assaults.Cryptography is also described as an art form in the dictionary.The term "cyber-crime"; refers to the act of committing a crime.Creating good codes or breaking current ones relied on ingenuity and a deep grasp of how codes work.There was little theory to rely on, and there was no working definition of what constitutes a good code for a long time.Beginning in the 1970s and 1980s, this perception of cryptography began to shift dramatically.A rich theory emerged, allowing for the careful study of cryptography as a science and a mathematical discipline.This viewpoint has influenced how researchers view the broader field of computer security.

In short, cryptography has evolved from a heuristic set of techniques aimed at securing secret communication for the military to a science that aids in the security of systems for ordinary people all around the world. This also implies that cryptography has grown in importance in computer science.Contemporary cryptography is the foundation of computer security and communication.It is founded on various mathematical principles such as: number theory, complexity theory, and probability theory.

1.2 Symmetric cryptography

1.2.1 Definition

Symmetric cryptography, known also as secret key cryptography, is the use of a single shared secret to share encrypted data between parties. Ciphers in this category are called symmetric because you use the same key to encrypt and to decrypt the data. In simple terms, the sender encrypts data using a password, and the recipient must know that password to access the data.

Symmetric encryption is a two-way process. With a block of plain-text and a given key, symmetric ciphers will always produce the same cipher-text. Likewise, using that same key on that block of cipher-text will always produce the original plain-text. Symmetric encryption is useful for protecting data between parties with an established shared key and is also frequently used to store confidential data.

1.2.2 Functionality

In symmetric cryptography, two entities, traditionally known as Alice and Bob, share a key. When Alice wishes to encode a message to send to Bob, she uses a symmetric algorithm, using the secret key and the message as parameters. When Bob receives the message, he applies the corresponding decryption algorithm, using the same key as a parameter. where E is the encryption function and E^{-1} the corresponding decryption function. 3 Data Encryption Standard **(DES)** and Advanced Encryption Standard **(AES)** are two of the best-known and most robust symmetric encryption algorithms.

Despite the existence of robust algorithms and strong performances in terms of calculations, symmetric cryptography presents two main limitations:

- the number of keys to manage: a different symmetric key is needed for each pair of correspondents. Thus, the number of keys required increases in line with the square of the number of individuals;
- the exchange of the secret key: we know that Alice and Bob share a key, but the way in which this key is exchanged is not specified. Security at this stage is a significant issue; asymmetric cryptography offers one possible solution.

1.2.3 History of symmetric Algorithms

The rapid increase of computing power beginning in the 1970s transformed the cryptography landscape. Hundreds of algorithms have been developed and hundreds have been broken. There are three algorithms which are notable for their resistance to decryption and their wide-spread usage.

- **The Data Encryption Standard** (DES) algorithm was developed at IBM and first published in 1977. It was one of the first N.A.S.A. approved standards for encryption. It was widely used from 1977 until 2000. It was also widely studied and informs the design of later algorithms. However, it is now considered insecure. Modern computers can decrypt a DES encrypted message in less than a day. There is a successor to DES which is Triple DES (DES encryption applied three times) which is still considered secure and is commonly used.
- **The Advanced Encryption Standard** (AES) algorithm was selected by the N.A.S.A. as the winner of an algorithm competition held in 2002. It is also known as"Rijndael" (pronounced rain-dahl) because that was its name during the competition. AES has not yet been broken is still considered strong enough to encrypt U.S. classified data.

- **The Blow fish** algorithm was designed in 1993. It has not yet been broken, even though a few technical and theoretical weaknesses have been identified. It is also widely used in many encryption software products. The feature that make Blow fish interesting is that it is slow compared to other encryption algorithms. This is a useful trait for password encryption.

1.2.4 Uses of symmetric cryptography

Due to the better performance and faster speed of symmetric encryption, symmetric cryptography is typically used for bulk encryption of large amounts of data.

Banking Sector :
Applications of symmetric encryption in the banking sector include:

- Payment applications, such as card transactions where PII (Personal Identifying Information) needs to be protected to prevent identity theft or fraudulent charges without huge costs of resources. This helps lower the risk involved in dealing with payment transactions on a daily basis.
- Validations to confirm that the sender of a message is who he claims to be.

Securing Data at Rest :
When a website or organization stores personal information regarding their users or the company itself, it is protected using Symmetric encryption. This is done to prevent all kinds of snooping from either outside hackers or disgruntled employees inside the office, looking to steal crucial information.
SSL/TLS Handshake :
Symmetric encryption plays a significant role in verifying website server authenticity, exchanging the necessary encryption keys required, and generating a session using those keys to ensure maximum security, instead of the rather insecure HTTP website format.

1.2.5 Symmetric cryptography pros and cons

Symmetric key algorithms are very fast and can encrypt large amounts of data and is easy to implement with only a single key needed for both encryption and decryption of data, setting up symmetric infrastructure for an organization is relatively easy compared to asymmetric encryption The primary drawback of symmetric key algorithms is the ”key distribution problem”. Once data is encrypted it can be sent publicly, however, the key or password must also be given to the recipient. If the key is sent with the

data or sent in plain text through another visible channel, then it is easily intercepted and the data will be easily decrypted. One common solution is to use symmetric key algorithms to encrypt the data and then to use asymmetric key algorithms (a.k.a. public key cryptography) to encrypt only the key which unlocks the encrypted data.[1][2]

1.2.6 The application of symmetric encryption

The most used and known symmetric encryption applications are **DES** , **3DES** and **AES**

- **DES functionality** DES is a block cipher–meaning it operates on plain-text blocks of a given size (64-bits) and returns cipher-text blocks of the same size. Thus DES results in a permutation among the 2^{64} (read this as: "2 to the 64th power") possible arrangements of 64 bits, each of which may be either 0 or 1. Each block of 64 bits is divided into two blocks of 32 bits each, a left half block L and a right half R. (This division is only used in certain operations.) DES operates on the 64-bit blocks using key sizes of 56- bits. The keys are actually stored as being 64 bits long, but every 8th bit in the key is not used (i.e. bits numbered 8, 16, 24, 32, 40, 48, 56, and 64). However, we will nevertheless number the bits from 1 to 64, going left to right, in the following calculations. But the eight bits just mentioned get eliminated when we create sub-keys.[3]

  Decryption
  Decryption is simply the inverse of encryption, following the same steps as above, but reversing the order in which the sub-keys are applied.

- **Triple-DES** is just DES with two 56-bit keys applied. Given a plain-text message, the first key is used to DES- encrypt the message. The second key is used to DES-decrypt the encrypted message. (Since the second key is not the right key, this decryption just scrambles the data further.) The twice-scrambled message is then encrypted again with the first key to yield the final cipher-text. This three-step procedure is called triple-DES. Triple-DES is just DES done three times with two keys used in a particular order. (Triple-DES can also be done with three separate keys instead of only two. In either case the resultant key space is about 2^{112}.)

- **AES functionality** The number of rounds 10, is for an encryption key that is 128 bits long. (As previously stated, the number of rounds is 12 for 192-bit keys and 14 for 256-bit keys.)

  The input state array is XORed with the first four words of the key schedule before any round-based encryption processing can occur. During decryption, the same thing happens, only we now XOR the cipher-text state array with the last four words of the key schedule.

  Each cycle of encryption consists of the four processes listed below:

  1) Substitute bytes, **2)** Shift rows, **3)** Mix columns, and **4)** Add round key. The last step consists of XORing the output of the previous three steps with four words from the key schedule.

  For decryption, each round consists of the following four steps: **1)** Inverse shift rows, **2)** Inverse substitute bytes, **3)** Add 12 Computer and Network Security by Avi Kak Lecture 8 round key, and **4)** Inverse mix columns. The third step is to XOR the results of the previous two processes with four words from the key schedule.Take note of the variations between the order in which substitution and shifting operations are performed in a decryption round and the order in which equivalent operations are performed in an encryption round. The last round for encryption does not involve the “Mix columns” step. The last round for decryption does not involve the “Inverse mix columns” step.

1.3 Asymmetric cryptography

1.3.1 The history of asymmetric cryptography

Whitfield Diffie and Martin Hellman, researchers at Stanford University, first publicly proposed asymmetric encryption in their 1977 paper, ”New Directions in Cryptography.”

James Ellis independently and discreetly presented the idea several years prior, while he was employed by the British government communication headquarters(GCHQ). The asymmetric algorithm described in the Diffie-Hellman paper generates decryption keys by raising numbers to certain powers. The first time Diffie and Hellman collaborated was in 1974 to address the issue of key distribution.

Ronald Rivest, Adi Shamir, and Leonard Adleman were the three men who created the RSA algorithm, which was based on Diffie’s work. The RSA algorithm was created in 1977, and it was first published in Communications of the ACM in 1978.[4]

1.3.2 The mechanism of asymmetric cryptography

Asymmetric cryptography, commonly referred to as public-key cryptography, is a method for encrypting and decrypting messages and safeguarding them from unwanted access or use. It makes use of a pair of linked keys, a public key and a private key.

A public key is a cryptographic key that anybody can use to encrypt messages in such a way that only the intended recipient can decrypt them using their private key. A private key, usually referred to as a secret key, is only disclosed to the key’s creator.

A person can obtain the intended recipient’s public key from a public directory and use it to encrypt the message before sending it when they want to send an encrypted communication. With their respective private key, the message’s recipient can then decrypt it.

If the sender encrypts the communication with their private key, only their public key may be used to decrypt it, authenticating the sender. Users do not need to physically lock and unlock the message in order for these encryption and decryption processes to take place.

The transport layer security (TLS) and secure sockets layer (SSL) are protocols, which enable HTTPS, are among the many technologies that depend on asymmetric cryptography.

Software applications that need to validate a digital signature or create a secure connection over an unsecured network, such browsers on the internet, also require encryption.

The main advantage of asymmetric cryptography is increased data security. Because users are never compelled to divulge or exchange their private keys, there is less possibility that a cybercriminal will intercept a user's private key during transmission, making it the most secure encryption procedure available.[5]

A public key and a private key, which are related mathematically, are used in asymmetric encryption . The associated private key is used for decryption if the public key is utilized for encryption. The associated public key is used for decryption if the private key is employed for encryption.

The sender and the recipient are the two parties involved in the asymmetric encryption process. Each has a unique set of public and private keys. The sender must first have the recipient's public key. The plain-text communication is then encrypted by the sender using the public key of the recipient. Hence, cipher-text is produced. The recipient receives the cipher-text, which they decrypt with their private key to produce readable plain-text. Even though each sender has the receiver's public key, they cannot read each other's communications because the encryption function is one-way..

1.3.3 Uses of asymmetric cryptography

Data is often authenticated using digital signatures and asymmetric cryptography. A communication, piece of software, or digital document can have its integrity and validity verified using a digital signature, which is a mathematical process. It serves as a digital substitute for a handwritten or stamped seal.

Digital signatures, which are based on asymmetric cryptography, can guarantee evidence of the origin, identity, and status of an electronic document, transaction, or message and acknowledge the signer's informed permission.

Systems with multiple users who may need to encrypt and decode messages can also employ asymmetric cryptography, such as:

Secure email. A message can be encrypted using a public key and decrypted using a private key. SSL/TLS. Asymmetric encryption is used to create secure connections between browsers and websites. Crypto-currencies, Asymmetric cryptography is used by Bitcoin and other crypto-currencies. Users have private keys that are kept private and public keys that are visible to everyone. Using a cryptographic method, Bitcoin makes sure that only authorized users can access the money. Each unspent transaction output (UTXO) on the Bitcoin ledger is often linked to a public key. To pay money to user Y, for instance, user X, who has a UTXO linked to his public key, signs a transaction using his private key to spend the existing UTXO and generate a new UTXO linked to user Y's public key.

1.3.3.1Asymmetric cryptography Pros and Cons:

Asymmetric cryptography has several advantages, including:

There is no need to exchange keys, hence the issue with key distribution is solved. As the private keys never need to be communicated or made public, security is increased. It is possible to use digital signatures so that a recipient may confirm the identity of the sender of a message. It allows for non-repudiation so the sender can't deny sending a message. Disadvantages of asymmetric cryptography include:

Compared to symmetric cryptography, it is a slow procedure. Hence, decrypting bulk messages is not a practical use for it. An individual cannot decrypt the communications he receives if he misplaces his private key. Nobody can verify that a public key belongs to the person supplied because public keys aren't authenticated. Users must therefore confirm that their public keys are theirs. A malicious actor can read a person's messages if they know that person's private key.

1.3.4 The difference between asymmetric vs. symmetric cryptography

The primary distinction between asymmetric and symmetric cryptography is the use of two distinct but linked keys in asymmetric encryption techniques. Data can be encrypted with one key and decrypted with another. The same key is used in symmetric encryption to carry out both encryption and decryption operations. The size of the keys is another distinction between symmetric and asymmetric encryption. The length of the keys, which are chosen at random in symmetric cryptography, is commonly set at 128 bits or 256 bits, depending on the required level of security. The public and private keys in asymmetric encryption must have a mathematical relation. Asymmetric keys must be longer to provide the same level of security since malicious actors may use this pattern to break the encryption. A 2048-bit asymmetric key and a 128-bit symmetric key offer roughly the same level of security due to the stark contrast in key lengths.

Compared to symmetric encryption, which executes more quickly, asymmetric encryption is noticeably slower.[4]

1.3.5 The applications of asymmetrical encryption.

The asymmetrical encryption have a wide range of use such as: digital signatures to maintain the authenticity of documents encrypted browsing for a better protection against hackers managing crypto-currency for safe transactions and last but not least sharing keys for symmetric key cryptography

1.3.6 some examples about asymmetric encryption

1.3.6.1 DSA algorithm (Digital signature Algorithm): Digital Signature Algorithm (DSA) is a widely used public-key cryptography algorithm for digital signatures. It was developed by the US National Institute of Standards and Technology (NIST) and published as a federal standard in 1994. DSA is based on the discrete logarithm problem in a finite field.

Here is how the DSA algorithm works:

Key generation: The first step is to generate a public key and a private key. The public key consists of two large prime numbers, p and q, and a generator g, where g is a primitive root modulo p. The private key is a random number, x, where $0 < x < q$.

Signing a message: To sign a message, the sender needs to have their private key and a hash of the message. The signature process involves the following steps:
a. Choose a random number k, where $0 < k < q$. b. Calculate

$$r = (g^k \mod (p)) \mod (q)$$

. c. Calculate

$$s = (k^{-1} * (H(m) + x * r)) \mod (q)$$

, where H(m) is the hash of the message.
d. The signature is the pair (r, s).

Verifying a signature: To verify a signature, the recipient needs to have the sender's public key and the signature. The verification process involves the following steps:
a. Verify that r and s are in the range 1 to (q-1).
b. Calculate :

$$w = (s^{-1}) \mod (q) \tag{1}$$

c. Calculate:

$$u_1 = (H(m) * w) \mod (q) \tag{2}$$

$$u_2 = (r * w) \mod (q) \tag{3}$$

d. Calculate :

$$v = (((g^{u1} * y^{u2}) mod(p)) mod(q)) \tag{4}$$

where y is the sender's public key

e. The signature is valid if and only if v = r.

DSA provides a relatively short signature compared to RSA, and it is also considered to be more secure than RSA for the same key size. However, one disadvantage of DSA is that it can be slower than other signature algorithms. It is commonly used in many security applications, including digital certificates, secure email, and electronic voting systems.

1.3.6.2The Diffie-Hellman key exchange :

The Diffie-Hellman key exchange is a cryptographic algorithm that allows two parties to establish a shared secret key over an insecure communication channel. The algorithm was developed by Whitfield Diffie and Martin Hellman in 1976.

Here is how the Diffie-Hellman key exchange works:

Key generation: Both parties agree on a prime number, p, and a primitive root modulo p, g. They each generate a private key, a and b, respectively, where 0 < a,b < p-1.

Public key exchange: Both parties then publicly exchange their values of $g^a mod(p)$ and $g^b mod(p)$, respectively. These values are often referred to as their "public keys".

Secret key derivation: Each party then computes a shared secret key by taking the other party's public key and raising it to the power of their private key. Specifically, the shared secret key is calculated as follows:
a. Party A computes the shared secret key as

$$(g^b mod(p))^a mod(p)$$

. b. Party B computes the shared secret key as

$$(g^a mod(p))^b mod(p)$$

.

Because the discrete logarithm problem is difficult to solve, an attacker who intercepts the public keys exchanged by the two parties would not be able to calculate the shared secret key.

The Diffie-Hellman key exchange is vulnerable to a man-in-the-middle attack, where an attacker intercepts the public keys exchanged by the two parties and replaces them with their own public keys. To prevent this, the two parties must authenticate each other's public keys using a digital signature or a trusted third party.

The Diffie-Hellman key exchange is widely used in many cryptographic protocols, including SSL/TLS for secure web communication, SSH for secure shell communication, and PGP for email encryption. It is also used as a building block for other cryptographic protocols such as the ElGamal encryption algorithm and the Digital Signature Algorithm (DSA)

1.3.6.3 ElGamal Algorithm:

ElGamal is an asymmetric encryption algorithm that can be used for both encryption and digital signatures. It was named after its inventor Taher Elgamal and was first described in 1985. ElGamal is based on the Diffie-Hellman key exchange and uses modular arithmetic to encrypt and decrypt messages.

Here is how the ElGamal encryption algorithm works:

Key generation: To use the ElGamal encryption algorithm, you need to generate a public key and a private key. The public key consists of two numbers, p and g, where p is a large prime number and g is a primitive root modulo p. The private key is a random number, x, where 0 < x < p-1.

Encryption: To encrypt a message, you need the recipient's public key (p and g) and a random number, k, where 0 < k < p-2. The encryption process involves the following steps:
a. Convert the message into a number, M, where 0 < M < p.
b. Calculate two numbers:

$$C_1 = g^k (mod(p))$$

and

$$C_2 = M * (y^k)(mod(p))$$

, where

$$y = g^x(mod(p))$$

.
c. The encrypted message is the pair

$$(C_1, C_2)$$

.

Decryption: To decrypt a message, the recipient uses their private key (x) and the encrypted message

$$(C_1, C_2)$$

to recover the original message. The decryption process involves the following steps:
a. Calculate the value of$y = C_1^x(mod(p))$.
b. Calculate the modular inverse of y,

$$y^{-1}(mod(p))$$

.
c. Recover the original message by computing

$$M = (C_2 * y^{-1})(mod(p))$$

. Note: A disadvantage of ElGamal encryption is that there is message expansion by a factor of 2. That is, the cipher-text is twice as long as the corresponding plain-text.

ElGamal can also be used for digital signatures. To create a digital signature, the sender generates a pair of keys (public and private), hashes the message using a one-way hash function, and then encrypts the hash using their private key. The recipient can verify the signature by decrypting the hash using the sender's public key and comparing it to the hash of the original message. If the hashes match, the signature is valid.

1.3.6.4 RSA encryption:

A popular form of public-key cryptography used for data encryption of email and other digital transactions via the Internet is RSA encryption, or Rivest-Shamir-Adleman encryption. RSA was developed by Leonard M. Adleman, Adi Shamir, and Ronald L. Rivest while they were professors at the Massachusetts Institute of Technology.

Key Generation: a. Choose two large prime numbers, p and q, and compute

$$n = p * q$$

. b. Compute

$$\varPhi(n) = (p - 1) * (q - 1)$$

, where $\varPhi$ is Euler's totient function.
c. Choose an integer e such that 1 < e < $\varphi(n)$ and $\gcd(e, \varPhi(n)) = 1$.
d. Compute d such that

$$d * e \equiv 1(mod\varPhi(n))$$

. In other words, d is the modular multiplicative inverse of $emod\Phi(n)$. e. The public key is **(n, e)**, and the private key is **(n, d)**.

Encryption: Given a plain-text message M, the sender computes the cipher-text C as follows:

$$C = M^e(mod(n))$$

Decryption: Given a cipher-text C, the recipient computes the plain-text message M as follows:

$$M = C^d(modn)$$

It is more difficult to digitally sign a message and calls for a secure "hashing" mechanism. This function, which is widely known, reduces any message into a digest, a smaller message in which each bit depends on every other bit in such a way that changing even one bit in the message is likely to change half of the bits in the digest in a cryptosecure manner. It is computationally impossible for anyone to generate a message that will produce a preassigned digest, and it is as difficult to find a message that produces the same digest as a known one. This is what is meant by the term "cryptosecure." A encrypts the digest with the secret e, which A appends to the message, to sign a message—which may not even need to be kept secret. The digest can then be obtained by anyone using the public key d to decrypt the message, which can be done separately from the message. If the two are in agreement, it must be assumed that A invented the cipher because only A understood e and could, thus, have encoded the message. The separation of the authentication or signature channel from the privacy or secrecy channel comes at a very high cost in all of the two-key cryptosystems that have been suggested so far. The channel capacity is severely reduced by the substantial increase in computation required by the asymmetric encryption/decryption process (bits per second of message information communicated). 20 years or so for comparable security systems, Single-key algorithms have been able to attain through-puts that are 1,000–10,000 times greater than those of two-key algorithms. As a result, hybrid systems are where two-key cryptography is most commonly used. In such a system, the main communication is conducted at fast speed using a single-key algorithm while a two-key method is utilized for authentication and digital signatures. This key is discarded after the session has ended.

1.3.7 Advantages and disadvantages of asymmetric encryption:

Advantages - Convenience: It addresses the issue of sharing the encryption key. Everyone makes their public keys available while keeping their private keys a mystery. - Facilitates message authentication The integration of digital signatures with public key encryption enables the recipient of a message to confirm that the message actually came from a certain sender. The use of digital signatures in public key encryption enables the receiver to determine whether the communication was tampered with while in transit. A message that has been digitally signed cannot be changed without the signature becoming invalid. - Enable non-repudiation: Signing a communication digitally is equivalent to physically signing a document. The communication has been acknowledged, thus the sender cannot retract it. See More: How to Correctly Encrypt Your Mail Disadvantages - Public keys should/have to be verified: Everyone must confirm that their public keys belong to them because no one can be certain that a public key belongs to the person it claims to represent. Comparatively speaking to symmetric encryption, public key encryption is slow. Not practical for use in mass message decryption. - Consumes more system resources: Compared to single-key encryption, it takes a lot more computer hardware. - There is a chance of widespread security compromise: An attacker can see a person's whole history of messages if they discover their private key. - A lost private key may be beyond repair: The absence of a private key prevents the decryption of any received communications. Diverging opinions exist on which of them is more secure. Some experts believe that symmetric key encryption is more secure, while others favor the use of public keys for encryption. In order to profit from their advantages, it is ideal that they are both employed together. See More: How to Correctly Encrypt Your Mail.[6]

1.4 Attacks on RSA

1.4.1 cyber attacks

Any attempt to obtain unauthorized access to a computer, computing system, or computer network with the intention of causing harm is referred to as a cyber attack. The goal of a cyber attack is to disable, disrupt, destroy, or take control of a computer system, as well as to change, block, delete, modify, or steal the data stored on it.

A cyber attack can be launched by any person or group from any location using one or more different attack tactics.

The majority of the time, those who commit cyber-attacks are referred to as cyber-criminals. These include persons who act alone and use their computer abilities to plan and carry out malicious assaults. They are also frequently referred to as bad actors, threat actors, and hackers. They may also be a part of a gang and collaborate with other potential threat actors to discover vulnerabilities—weaknesses or issues with the computer systems—that they might use to their advantage.

Cyber-attacks are also carried out by organizations of computer professionals funded by the government. They have been accused of assaulting the information technology (IT) infrastructure of other governments as well as non-governmental organizations including companies, charities, and utilities. They have been classified as nation-state attackers.[7]

1.4.2 The most known types of attacks on RSA:

- ***Brute force attack***: the brute force attack also known as trial and error hacking method has no spacial algorithms it just the act of trying every possible combination for a PIN or a password or in our case a privet key ,However, the success of the attack depends on a number of variables, including the length and password's combination of letters, numbers, and special characters. Sites typically advise using strong passwords that are long and have a variety of characters. Due to this, brute force attacks are challenging but not impossible. By using brute force, getting the password will take longer. Thus, organizations frequently utilize brute force password cracking to assess the security of weak passwords. Another technique to stop this attack is to limit a user based on the number of failed login attempts. Due to this, web-based services will either display captchas or restrict your IP address if you enter incorrect passwords three times. there are several types of brute force attack that will be cited below.

1 .***Attack using reverse brute force:*** It is a reverse approach to password cracking. Assume the attacker knows the password but is unaware of the username. For this aim, the attacker will test the same password against any username guess that might be made. Every website that is not blocked after a predetermined number of unsuccessful tries is vulnerable to this strategy.[8]

2 .***Simple brute force attack:*** When a hacker uses no software at all and tries to guess a user's login information manually, this is known as a simple brute force assault. Usually, this is done using personal identification number (PIN) codes or common password combinations.

These attacks are straightforward since a lot of individuals continue to use easy passwords like "password123" or "1234" or bad password habits like using the same password across different websites. Hackers can also guess passwords by using simple reconnaissance techniques to decipher a person's prospective password, such as the name of their preferred sports team.

3 .***Dictionary Attack***:A dictionary attack is a fundamental type of brute force hacking in which the attacker chooses a target and then compares potential passwords to the user name of that person. Although the assault technique itself is not technically a brute force attack, it can be a key step in a malicious actor's password cracking operation.

Dictionary attacks are so named because they involve hackers going through dictionaries and replacing words with symbols and numbers. Compared to more recent, more successful attack strategies, this form of attack is often time-consuming and has a lower likelihood of success.

4 .***hybrid brute force attack***:A dictionary attack method combined with a straightforward brute force attack is known as a hybrid brute force attack. The hacker first needs to have access to a username before using a dictionary attack and straightforward brute force techniques to find an account login combination.

Starting with a list of probable words, the attacker tries various character, letter, and number combinations before settling on the right one. Hackers can use this method to find passwords like "Salaheddine123" or "zaki2020" that combine well-known or often used nouns with numbers, years, or random characters.[9]

- ***Side-channel attack***: A side-channel attack is a kind of cryptographic system attack that takes advantage of data leakage through a channel other than the system's typical inputs and outputs. For instance, side-channel attacks may take advantage of electromagnetic radiation, power usage, or even sound or vibration that a gadget emits while it is in use.

A side-channel attack's fundamental premise is that the physical implementation of a cryptographic algorithm can provide details about the secret key that was used to encrypt the algorithm. For instance, an attacker may be able to determine the key by observing how much power a device uses when performing cryptographic operations. Although there are many various kinds of side-channel attacks, the following are some typical examples:

1 ***power analysis attacks***: These attacks take advantage of the fact that a device's power consumption can vary depending on the tasks it is performing, including cryptographic tasks.

2 ***Timing-Attacks***: that take use of the fact that the length of time required to carry out cryptographic operations can vary depending on the parameters of the secret keys being utilized.
In the case of RSA, a timing attack takes advantage of the fact that the value of the secret key affects how long it takes to conduct a private key action, such as decryption or signature. Particularly, when the private key has more 1-bits in its binary form, the multiplication operation utilized in RSA's private key operation takes longer. By performing a decryption or signing operation on a large number of carefully selected cipher-texts, for instance, an attacker can time the length of time needed to complete a sequence of private key operations. The attacker can determine the value of the private key by comparing the measured times.

When an RSA implementation is susceptible to other sorts of attacks, like buffer overflow or format string flaws, which give an attacker access to the memory of the RSA implementation, timing attacks can be very successful. An attacker can make the RSA implementation leak timing data that reveals the private key by carefully modifying the input data. There are two basic categories of timing attacks: passive and active. An attacker who uses a passive timing attack merely keeps track of how long cryptographic processes take to complete. An active timing attack involves the attacker actively changing the system's timing behavior, such as via introducing pipeline stalls or cache misses.

Not just private keys but also other types of information can be gleaned from cryptography implementations through timing attacks. Timing data can be used, for instance, to determine whether a specific plain-text is being encrypted or to extract intermediate values from a cryptographic protocol, such as session keys or nonce's. Time attacks can be used against symmetric ciphers like AES and hash functions like SHA-256 in addition to other cryptographic methods outside RSA. The core idea is the same: an attacker can use this information to determine the values of the keys being used because it depends on those values how long it takes to complete cryptographic operations.

Timing attacks are not always simple to execute in practice, it should be noted. They frequently call for exact measurement tools and meticulous statistical examination of enormous data sets. Nonetheless, timing attacks are becoming more practical as computational power and measurement methods advance, therefore it's critical for cryptographers and implementers to be aware of this issue and take precautions to reduce it. It's crucial to implement RSA in a method that doesn't reveal time information in order to fend off timing attacks. This can be done by employing strategies like constant-time algorithm implementation or adding random delays to operations. Also, by increasing the unpredictability in the length of time required to carry out a private key operation, using a higher key size might make timing attacks more challenging.

3 ***electromagnetic analysis Attacks***: These attacks take use of the electromagnetic radiation that a device emits while it is in use, which can reveal information about the calculations that are being done by the device.

4 ***Acoustic attacks***: These attacks take use of the sound or vibration that a device emits while it is in use, which can reveal details about the computations that are being carried out by the device. Because side-channel attacks frequently take advantage of a cryptographic algorithm's implementation's physical properties, which are challenging to regulate or quantify, they can be challenging to protect against.

Nonetheless, there are methods to reduce the likelihood of side-channel assaults, including:

- employing constant-time algorithms to implement cryptographic algorithms in a way that is resistant to side-channel attacks.
- using hardware or software countermeasures, such as increasing noise in power usage or restricting the quantity of electromagnetic radiation a device emits, to lessen the amount of information that is spilled through side-channels.
- Tamper-proof hardware and other physical security measures can aid in preventing attackers from physically accessing a device and launching side-channel assaults. All things considered, side-channel attacks pose a serious risk to the security of cryptographic systems, hence it is crucial for those who develop and implement cryptographic algorithms to be aware of this risk and take precautions to lessen it.

- ***Chosen cipher-text attacks (CCA)***: In a chosen-ciphertext attack, the attacker chooses the cipher-text, delivers it to the target, and receives the associated plain-text, or a portion of it, back. If the attacker has the ability to select the cipher-texts, the attack is said to as adaptive.

  depending on how the attack has performed in the past. It is commonly known that a chosen cipher-text attack can be used against ordinary RSA. An attacker can select a random integer s and request the decryption of the seemingly harmless message $c_0 = secmodn$ in order to learn the decryption of a cipher-text $m \equiv cd(modn)$. It is simple to deduce the original message from the response $m0 = (c0)d$ because $m \equiv m_0 s^{-1}(modn)$. Another well-known outcome of RSA encryption is that the least important bit is just as secure as the entire message. In particular, if a second algorithm is capable of predicting the least significant bit of a message using nothing more than the public key and the matching cipher-text, then the first method can decrypt cipher-texts. Recently, Hastad and Naslund extended this finding to demonstrate the security of every single RSA bit. As a result, in a chosen-ciphertext attack, it may not be necessary for the attacker to know the entire decrypted message: a single bit per selected cipher-text may be sufficient.[10][11][12]

1.5 Factorization algorithms attack

1.5.1 Factorization attack

The factorization attack or factorization algorithms ,are algorithms that target systems that base their security on large composite numbers as privet keys. these attacks aim to exploit vulnerability or weakness in the implementation of the factorization algorithm it is often used against RSA like systems .
if the attacker successfully factorize the modulus of the RSA key pair then he could find the privet key hens break the security system.

1.5.2 big O notation and time complexity

Before going through the factorization algorithm we are going to introduce the notion of **big O notation** or **time complexity**
Big-O notation is a mathematical notation used to express the upper bound of a function's growth rate.It is commonly used in computer science to describe an algorithm's time complexity, or how the time required to execute the algorithm grows with the size of the input.The notation is $\mathcal{O}(f(n))$, where $f(n)$ is a mathematical function that describes the algorithm's growth rate as the size of the input n increases.The notation $\mathcal{O}$(f(n)) indicates that the time complexity of the algorithm grows no faster than $f(n)$ as n increases.
An algorithm with a time complexity of $\mathcal{O}(n)$, for example, means that its execution time grows linearly with the size of the input.The execution time of an algorithm with a time complexity of $\mathcal{O}(n2)$ grows quadratically with the size of the input.In general, the lower the algorithm's growth rate, the more efficient it is.As a result, algorithms with lower Big-O notations are preferable to those with higher Big-O notations.However, it is important to note that the actual performance of an algorithm can be affected by a variety of factors, including the hardware and software used to execute it.Complexity refers to how a program's or algorithm's resource requirements scale; complexity affects performance but not the other way around.

1.5.3 The most encountered orders of Big-O's

Here is a list of classes of functions that are commonly encountered when analyzing algorithms. The functions that grow more slowly are listed first.c is an undefined constant. notation names

Notation	**Name**
$\mathcal{O}(1)$	constant
$\mathcal{O}(\log(n))$	Logarithmic
$\mathcal{O}((\log(n))^c)$	Polylogarithmic
$\mathcal{O}(n)$	Linear
$\mathcal{O}(n^2)$	Quadratic
$\mathcal{O}(n^c)$	Polynomial
$\mathcal{O}(c^n)$	Exponential

Constant Time: $\mathcal{O}(1)$ - This means that the algorithm's run-time does not change as the size of the input increases.
Linear Time:$\mathcal{O}(n)$ - This means that the algorithm's run time increases linearly with the size of the input.
Quadratic Time:$\mathcal{O}(n^2)$ - This means that the algorithm's run-time increases quadratically with the size of the input.
Cubic Time:$\mathcal{O}(n^3)$ - This means that the algorithm's run-time increases cubically with the size of the input.
Exponential Time:$\mathcal{O}(2^n)$ - This means that the algorithm's run-time doubles with each additional input element, making it grow exponentially.
Logarithmic Time: $\mathcal{O}(logn)$- This means that the algorithm's run-time increases logarithmically with the size of the input.
Linearithmic Time:$\mathcal{O}(nlogn)$ - This means that the algorithm's run-time grows linearly with the input size while also taking into account the logarithmic increase in its runtime complexity.
These are just a few examples of how algorithms can be expressed using big-O notation. There are many other complexity classes, and the best way to understand them is to study the algorithms that fall into them.

1.5.4 Factorization algorithms

1.5.4.1 Fermat Algorithm

Fermat's algorithm : Fermat Factorization method is based on the evaluation of an odd integer value as it differs from two squares. This was named after Pierre de Fermat. Let's consider an integer N considering a and b such as :

$$N = x^2 – y^2 = (x + y)(x - y)$$

where (a+b) and (a-b) are the factors and we try to find each using the following formula :

$$x^2 = n + y^2$$

Exp : Factor n=187 Want to find $x = \sqrt{n + y^2}$ W entry different y values from 1 up When x is an integer , we found y .
$x = \sqrt{187 + 1^2} \neq int$
$x = \sqrt{187 + 2^2} \neq int$
$x = \sqrt{187 + 3^2} = int$
So , $14^2 = 187 + 3^2$ **then:**$x = 14andy = 3$
Recall : $N = (a + b)(a - b)$

$N = (14+3)(14-3)$
$N = (17)(11)$
Hence we found the prime factors of n

1.5.4.2 Pollard ρ-1 algorithm:

The factorization procedure Pollard p-1 is used to identify the prime factors of a given number. It was created by John Pollard in 1974 and is based on the notion that there is a significant likelihood that p-1 shares a nontrivial factor with a randomly chosen integer a if p-1 is the product of tiny primes.
This is how the algorithm works:

1 : we start by choosing a random integer "a" and a bound B (The bound B is a limit that we establish for **m** so that the algorithm ceases to operate once it reaches it because the equation that follows can easily become massive, with no need to compute such large numbers.)

2 :we then calculate the greatest common divisor (GCD) of $a^{(m!)} - 1$(there are other expression used other than this but we will be working with this formula) and the integer to be factored N, where m = 2, 3, 5, 7, 11, 13, ..., and ! denotes the factorial function.

3 :If the GCD is a nontrivial factor of the integer, then we have found a factor. Otherwise,continue steps 2 and 3 with a bigger value of m until the bound B is attained

4 :If no factor is found, increase the value of B and repeat steps 1-3.

The approach works well for determining factors of integers that are "smooth," or have small prime factors. It may fail for some numbers and is not guaranteed to find all prime factors.

One of the many factorization methods used in contemporary cryptography is Pollard ρ-1, whose effectiveness in locating small factors can be exploited to break RSA-based cryptography keys. The RSA keys are generated using big prime factors to make RSA resistant to Pollard ρ-1.

here is a numerical application for ρ-1 algorithm: Consider that we wish to factor the composite number N = 1729 and that we select a random base a = 2. To maintain simplicity, let's also fix the bound B = 10.
Next, we compute the greatest common divisor of N and $a^{(m!)} - 1$ for various values of m, starting with m = 2:

$$\gcd(N, a^{(2!)} - 1) = gcd(1729, 3) = 1$$

$$\gcd(N, a^{(3!)} - 1) = gcd(1729, 63) = 1$$

$$\gcd(N, a^{(4!)} - 1) = gcd(1729, 16777215) = 1$$

$$\gcd(N, a^{(5!)} - 1) = gcd(1729, 8997457107) = 13 \textbf{ (found a nontrivial factor!)}$$

We can end here and print the factorization N = 13 x 67 because we discovered a nontrivial N factor, namely 13.

Notably, we only needed to compute four GCDs in this scenario before identifying a factor(we didn't get to the limit that we set B=10). It is challenging to anticipate how effective the method will be in general because the input values, the chosen expression, and the bound can all have a significant impact on the number of GCDs required.

1.5.4.3 Pollard rho algorithm:

Pollard's rho algorithm is a general-purpose factoring algorithm since it may be used to factor an arbitrary number N = p*q. The algorithm factors N with constant probability in $O(N^{1/4})$ heuristically.this is still exponential, but is a vast improvement over trial division.

To start the algorithm, we choose a random integer x0 and apply the function $f(x0)$ to get $x1 = f(x0)$. We then apply the function f to x1 to get $x2 = f(x1)$, and so on, generating a sequence of integers x0, x1, x2, x3, ... The algorithm's fundamental

concept is to create an integer series using a recurrence relation, and then to search for nontrivial factors of the sequence's numbers. The function $f(x)$, which accepts an integer x as input and returns another integer, creates the sequence. Despite being simple to compute, the function $f(x)$ has unpredictable and chaotic behavior.

here are the steps of the algorithm:

1 :Choose a random integer x_0 and compute $x_1 = f(x_0)$, where $f(x)$ is a function that maps an element in the cyclic group to another element.

2 :Compute $x_2 = f(f(x_0))$, $x_3 = f(f(x_1))$, $x_4 = f(f(x_2))$, and so on. In general, we can compute $x_i = f(x_{i-1})$ for i = 1, 2, 3, ...

3 :At each step, compute the greatest common divisor (GCD) of the difference between two elements in the sequence, say x_i and x_j, and the integer n we want to factor. If the GCD is a non-trivial factor of n, then we have successfully factored n and the algorithm terminates.

4 :If we do not find a factor in Step 3, then we continue generating the sequence of elements. Eventually, the sequence will repeat itself (due to the pigeonhole principle), and we will find a repeated element, say $x_i = x_j$, for some i and j. We can then compute the GCD of the difference between x_i and x_j with n. If this GCD is a non-trivial factor of n, then we have successfully factored n.

The selection of the function $f(x)$ determines whether the algorithm is successful . One typical option is $f(x) = (x^2 + c) \mod (n)$, where c is a random constant. Another popular option is $f(x) = (x^2 + c) \mod (n)$, where$c$ is once more a random constant. The algorithm's speed and success rate can be affected by different $f(x)$ choices.

here is a numerical example about pollard rho factorization algorithm:

step 1 : choose a random function $f(x)$ to generate a sequence of numbers ,let's take $f(x) = x^2 + 1$

step 2 : Choose a random starting value $x0$. We can choose $x0 = 2$.

Step 3 :Use the beginning value x0 and the function $f(x)$ to produce two sequences of numbers. The sequences can be created as follows:
Sequence 1:$x0, f(x0), f(f(x0)), f(f(f(x0)))$, ...
Sequence 2: $f(x0), f(f(x0)), f(f(f(x0))), f(f(f(f(x0))))$ ...
Both sequences are produced using the same initial value $x0$ and function $f(x)$.

step4 :let's try pairing up the first few numbers in each sequence and calculate their gcd:
Pair 1: $x0 = 2$ and $f(x0) = 5. \gcd(2, 5) = 1$.
Pair 2: $f(x0) = 5$ and $f(f(x0)) = 26. \gcd(5, 26) = 1$.
Pair 3: $f(f(x0)) = 26$ and $f(f(f(x0))) = 677. \gcd(26, 677) = 1$.
Pair 4: $f(f(f(x0))) = 677$and $f(f(f(f(x0)))) = 45812. \gcd(677, 45812) = 1$.
we haven't found a non-trivial number yet so wee continue:
Pair 5: $f(f(f(f(x0))))) = 45812$ and $f(f(f(f(f(x0)))) = 2100661. \gcd(45812, 2100661) = 1$.
Pair 6:$f(f(f(f(f(x0)))) = 2100661$ and $f(f(f(f(f(f(x0)))))) = 4416953706$. $\gcd(2100661, 4416953706) = 11$.

We have found a factor of N! We can verify that $143/11 = 13$, so the factors of N are 11 and 13.
The Pollard rho technique is a probabilistic approach, therefore even if a factor exists, there is a risk that it won't be discovered. Yet, it is frequently highly effective in practice for locating factors of huge composite numbers.

1.5.4.4 Quadratic sieve algorithm:

Trial division is inferior than Pollard's rho technique, which still takes an exponential amount of time. Run-time for the quadratic sieve algorithm is sub-exponential.was the fastest known factoring algorithm until the early 1990s and remains the factoring algorithm of choice for numbers up to about 300 bits long.
Here's a simplified explanation of how the quadratic sieve algorithm works:

1 Choose a number p such that p is slightly greater then N's square root .

2 Find a set of integers $x_1, x_2, ..., x_k$ such that each $x_i^2 \mod p$ can be expressed as a product of small prime factors.

3 Calculate the values of$y_i = x_i^2 - N$ for each xi in the set.

4 Factorize the y_i values using the factorization of$x_i^2 \mod p$ obtained in step 2, to find a subset of $y_i$values that have a product equal to a perfect square.

5 Express the product of the selected y_i values as a difference of two squares, say $a^2 - b^2$.

6 If a is congruent to $b \mod N$ or $\gcd(a-b, N)$is a non-trivial factor of N, then we have found a factorization of N

Given enough iterations, the quadratic sieve approach has a high probability of success even though it may not always find a factorization of N. For factoring integers made up of two big primes, the procedure is especially effective. **here is a numerical example about the quadratic sieve factorization algorithm**[13]

1.6 Conclusion

In this chapter we have treated the two main categories of modern cryptography along with their pros and downsides , we have introduced the problem of modern cryptography factorization .
In the next chapter we will introduce and go in details with the General Number Field Sieve the most know algorithm for factoring big composite number for it's efficiency and wholeness .

Chapter2:The GNFS algorithm

2 The GNFS algorithm

2.1 Introduction

In the past chapter we have talked about cryptography(symmetric and asymmetric) the difference between them,We also talked about the different algorithms of asymmetrical cryptography (RSA ,DSA ...) and the different attacks to break those algorithms , and at last we have briefly talked about the different factorization algorithms .
In this chapter we are going to understand the most efficient factorization algorithm the general number field sieve we are going to have a general view of it then a step-by-step walk through this algorithm and finally we will have an extended numerical example

2.1.1 Number field sieve(NFS)

Before getting into **the GNFS Algorithm** we will have a quick view about the Number field sieve (NFS) as it is the origin of it.
In 1988 John Pollard circulated the famous letter that presented the idea of the number field sieve (NFS).
In this thesis when we say "number field sieve" we are actually talking about the general number field sieve. The original number field sieve is denoted special number field sieve (SNFS) today.
The number field sieve is the fastest general algorithm known today, but because of its complexity and overhead it is only faster than the QS for number larger than 110- 120 digits. The number field sieve took the quadratic sieve to another level by using algebraic number fields.
we will describe the fundamental idea behind the algorithm and try to convince you that it works.
The next Chapter describes the algorithm in details. The number field sieve is a lot like the quadratic sieve but it differs on one major point: the field it works in.
A number field is a common word for all sub-fields of C and one way to construct a number field K is to take an irreducible polynomial $f(x)$ of degree **d** with a root $\alpha \in C$ and $K = Q[\alpha]$ is a degree d extension field.
A number ring is a sub-ring of a number field; the number field sieve uses the number ring

$$\mathtt{Z}[\alpha] = \mathtt{Z}[\mathtt{X}]/f(\mathtt{Z}[\mathtt{X}])$$

A familiar example of a number ring is the sub-ring $\mathtt{Z}[\mathtt{i}]$ of Gaussian integers which is a sub-ring of the number field $\mathtt{Q}[\mathrm{i}]$ derived from the polynomial$f(x) = x^2 + 1$ with $\alpha = i$.
The number field $Q[\alpha]$ consists of elements on the form$\sum_{j=0}^{d1} q_j \alpha^j$ with $q_j \in Q$, the number ring $\mathtt{Z}[\alpha] = \mathtt{Z}[\mathtt{X}]/f(\mathtt{Z}[\mathtt{X}])$ contains elements on the form $\sum_{j=0}^{d1}$ sj αj , with sj $\in \mathtt{Z}$. We need an embedding into $\mathtt{Z}$ for the number ring $\mathtt{Z}[\alpha]$ before it is usable for factoring, but fortunately we have the ring homo-morphism ϕ

Theorem : Given a polynomial $f(x) \in \mathtt{Z}[\alpha]]$, a root $\alpha \in$ C and m $\in \mathtt{Z/nZ}$ such that
$f(m) \sim= 0(modn)$, there exists a unique mapping

$$\phi : \mathtt{Z}[\alpha] \to \mathtt{Z/nZ}$$

satisfying

$$\phi(1) \equiv 1 \mod n$$

$$\phi(\alpha) \equiv m \mod n$$

$$\phi(ab) = \phi(a) * \phi(b)$$

$$\phi(a + b) = \phi(a) + \phi(b)$$

For all a,b $\in \mathtt{Z}[\alpha]$
The ring homo-morphism ϕ leads to the desired congruent squares. If we can find a

non-empty set S with the following properties

$$y^2 = \prod_{a,b \in S} (a + bm) : y^2 \in Z$$

$$\beta^2 = \prod_{a,b \in S} (a + b\alpha) : y^2 \in Z$$

we get the congruence

$$\begin{aligned} \phi(\beta^2) &= \phi(\beta) * \phi(\beta) \\ &= \phi(\beta^2) \\ &= \prod_{a,b \in S} \phi((a + b\alpha)) \\ &= \prod_{a,b \in S} \phi((a + b\alpha)) \\ &= \prod_{a,b \in S} (a + bm) \\ &= y^2 \end{aligned}$$

and one of the major question remaining is how to find the set **S**?

2.1.2 Complexity of the NFS Algorithm

Under some reasonable heuristic assumption the NFS Algorithm factors n in:

$$\ln[\frac{1}{3}, C] = \mathcal{O}(e^{C + \mathcal{O}(1) \sqrt[3]{\log n} \sqrt[3]{(\log \log n)^2}})$$

2.2 The GNFS Algorithm

The reader is now familiar with the many types of factoring algorithms and is aware that the fastest algorithms are based on sieving over a clearly defined domain; one can even go so far as to state that the fastest algorithms calculate

congruence's and resolve linear equation systems. The number field sieve variations have shown to be the most effective factoring algorithms for big composites in practice and have the lowest time complexity when compared to other factoring techniques.

The number field sieve variations have shown to be the best algorithms to utilize in practice for big composites and have the lowest time complexity when compared to other factoring techniques. The number field sieve and the quadratic sieve are quite similar and can be viewed as a development of the QS to make use of polynomials with degrees more than two.

The special number field sieve (SNFS) and the general number field sieve (GNFS) are the two primary variations. The GNFS works on all forms of composites, but the SNFS only works on a specific type of composites, namely integers of the form: r e s, for tiny integers r, s, and integer e. The polynomial selection portion of the method is where SNFS and GNFS differ from one another, since the special numbers that SNFS can be applied to create a unique class of especially appealing polynomials, and the work in the square root step is also more difficult for the GNFS.

2.2.1 Overview of the GNFS Algorithm

The algorithm is intricate, so it will be helpful to begin with a broad description in order to fully comprehend it. I'll give you an overview of the GNFS algorithm in this Section, look at implementation issues in the following Section by building algorithms for the various steps, and then describe the characteristics that make the algorithm effective. The algorithm is very similar to other sieving-based algorithms that were discussed in the previous chapter. The method consists of four basic steps that cannot be completed in parallel due to mutual dependence, while some of the steps can be completed in parallel internally. I've decided to break it down into 5 phases to create the integer factorization.

The algorithm's flow is easier to understand, but each step involves numerous calculations and multiple algorithms working together.

input: composite integer **n**. **output**: a nontrivial factor **p** of **n**.
Step 1: (Polynomial selection)
Find an irreducible polynomial $f(x)$ with root m, i.e. $f(m) \equiv 0 \mod (n)$, $f(x) \in Z[x]$.
Step 2: (Factor bases)
Choose the size for the factor bases and set up the rational factor base, algebraic factor base
and the quadratic character base.
Step 3: (Sieving)
Find pairs of integers (a, b) with the following properties:
$\gcd(a, b) = 1$
$a + bm$ is smooth over the rational factor base
$b^{deg(f)} f(a/b)$ is smooth over the algebraic factor base
A pair (a, b) with these properties is called a relation. The purpose of the sieving stage is to collect as many relations as possible (at least one larger than the elements in all of the bases combined). The sieving step results in a set S of relations.
Step 4: (Linear algebra)
Filter the results from the sieving by removing duplicates and the relations containing a prime ideal not present in any of the other relations.
The relations are put into relation-sets and a very large sparse matrix over **GF(2)** is constructed.
The matrix is reduced resulting in some dependencies, ie. elements which lead to a square modulo n.
Step 5: (Square root)
Calculate the rational square root, i.e. y with

$$y^2 = \prod_{(a,b)\in S} (a - bm)$$

Calculate the algebraic square root, i.e. x with

$$x^2 = \prod_{(a,b)\in S} (a - b\alpha)$$

where α is a root of $f(x)$.
p can then be found by $\gcd(n, x - y)$ and $\gcd(n, x + y)$.

The algorithm's flow is depicted in the above flowchart. The stages listed above do not even come close to describing the implementation-ready algorithm. The steps must be revealed to various algorithms for implementation purposes, with no steps hidden behind mathematical premises or presumptions. we will make reference to the aforementioned stages throughout the remainder of this chapter.

2.2.2 Implementing the GNFS Algorithm

we will discuss the GNFS algorithm in this Section in sufficient detail to allow for its practical implementation. we detailed the main theory underlying the algorithm and explained how and why it operates mathematically.

2.2.2.1 Polynomial Selection[14]

Finding a good polynomial is difficult since the definition of "a good polynomial" is not well defined, yet choosing a usable polynomial is not difficult either. It is one of the parts of the number field sieve that has received the least amount of research. Murphy's thesis is the greatest source on the topic.[1]
The quantity of smooth values that a polynomial f(x) generates in its sieve region is referred to as its yield. The yield of a polynomial is primarily influenced by two things. These are the attributes of size and root. If a polynomial f(x) has a good yield, or a good mix of size and root qualities, it is said to be excellent.
Definition 1 If the values that a polynomial $f(x)$ takes are tiny, the polynomial has a good size property. You can accomplish this by choosing a polynomial with few coefficients. The size attribute improves with decreasing size.
Definition 2 If a polynomial f(x) has numerous roots modulo tiny primes, it is said to have good root characteristics.
There is no simple formula for selecting a decent polynomial. The best approach is to guarantee a certain level of size and root characteristics before creating candidates. These candidates are then examined, and the candidate with the highest yield is picked.
Finding quality polynomials among the employable candidates is the issue. In his thesis, Brian Antony Murphy[1] extensively researched polynomial yield.
The following characteristics of the polynomial f(x) are required.

1 :is irreducible over$Z[x]$

2 : has a root m modulo n

By utilizing a base-m representation, it is simple to construct a polynomial in Z/Zn[x] with any desired root m. In most circumstances, this will also produce an irreducible polynomial; otherwise, we would have found n's factorization

If we could have, $f(x) = g(x)h(x)$ and $f(m) = g(m)h(m) = n$ This should of course be checked before proceeding.
The results from constructing a polynomial $f(x)$ by using the base-m representation of n can be modified freely as long as $f(m) \equiv 0 mod n$. That is why the polynomial does not have to be a true base-m expansion but an expansion of kn for some integer k.
The base-m representation of n is

$$\sum_{i=0}^{d} a_i m^i$$

using$0 < a_i < m$. d is predetermined and typically falls between 3-6. The ai's can be used as coefficients to build a degree d polynomial $f(x)$. M will serve as the polynomial's root, i.e.

$$f(x) = a_d x^d + a_{d-1} x^{d-1} + ... + a_0$$

$$f(m) \equiv 0 \mod (n)$$

By reducing the coefficients, for instance by increasing replacements, the size property can be attained.

$$ai = a_i - m$$

$$a_{i+1} = a_{i+1} + 1$$

which decreases $a_i's$ while maintaining $f(m) \equiv 0 mod n$. It is crucial to have a_d and a_{d-1} in particular small. Skewed polynomials appear to provide the highest levels of polynomial yield, but because they are far more difficult to construct, we will instead employ non-skewed polynomials.
which decreases $a_i's$ while maintaining $f(m) \equiv 0 mod n$. A_d and A_{d1} should be made as little as possible. Skewed polynomials appear to provide the highest levels of polynomial yield, but because they are far more difficult to construct, we will instead employ non-skewed polynomials. A practical heuristic for estimating polynomial yield has been developed by Murphy.
He introduces the $\alpha(F)$ function, which provides a very accurate estimate and enables the removal of polynomials that are plainly wrong before subjecting the remaining ones to a brief sieving test to determine which has the highest yield. the definition of the α-function is

$$\alpha(F) = \sum_{P \leq B} (1 - q_p \frac{p}{p+1}) \frac{\log p}{p-1}$$

where B is some smoothness bound, and q_p is the number of distinct roots of f mod p. The polynomial selection consists of the following steps

1 identify a large set of usable polynomials.

2 using heuristics remove obviously bad polynomials from the set

3 do small sieving experiments on the remaining polynomials and choose the one with the best yield.

this we think that we have provided enough information for creating polynomials suitable for the general number field sieve

---

Algorithm: **Polynomial Selection (non-skewed)**

input: composite integer n, integer d.
output: polynomial f(x)

```
1 choose $m_0$ in a way that $\lfloor n^{\frac{1}{d+1}} \rfloor \leq m_0 \leq \lceil n^{\frac{1}{d}} \rceil$
2 choose an interval of convenient $a_d's$ in a way that $X_1 < \frac{|a_d|}{m_0} < X_2$, where $0 < X_1 < X_2 < 0.5$
3 define the interval of m's such that $|a_{d-1}|$ are smallest. This will be the interval
  from m0 to where ad change, which is
      $$m_{change} = \lfloor \frac{n}{a_d}^{\frac{1}{d}} \rfloor$$
4 for all usable combinations of ad and$m \in [m_0 : m']$ with $a_d$ having a large b-smooth co-factor do
  (a) write n in base-m representation, such that
      $$n = a_d m^d + a_{d-1} m^{d-1} + ..... + a_0$$
  (b) write the polynomial $f_m(x) = a_d x^d + a_{d-1} x^{d-1} + ..... + a_0$
  (c)check yield, by calculating $\alpha(f_m)$.
  i. if yield is satisfying add $f_m(x)$ to the set F'
  ii. else discard polynomial.
5 choose $f(x)$ among the polynomials in F' by doing small sieving experiments.
6 return$f(x)$
```

end of the Algorithm

2.2.2.2 Factor bases
The factor bases are used to specify the domain that the algorithm will operate in. Empirically determining the sizes of the factor bases is required, and these sizes are
Depending on the sieving code's accuracy, it will determine whether all smooth elements are located or if some are missed using special-q techniques.

The rational factor base, or RFB, is made up of pairs (p, p mod m)a, i.e., all the primes pi up to some bound and pi mod m that we store.

$$RFB = (p_0, p_0 \mod m), (p_1, p_1 \mod m), ..$$

The algebraic factor base AFB consists of pairs (p, r) such that $f(r) \equiv 0 \mod p$, ie.

$$AFB = (p_0, r_0), (p_1, r_1), ...$$

The algebraic factor base should be two to three times as large as the rational factor base.
there is also The pairs (r, p) in the quadratic character base (QCB) that have the same characteristics as those of the elements in the algebraic factor base, except the p's are greater than the largest in the latter, i.e.

$$QCB = (p_0, r_0), (p_1, r_1), ...$$

Comparatively speaking to RFB and AFB, the number of elements in QCB is typically quite low. The number is less than 100 in record breaking factorization

2.2.2.3 Sieving

Although the sieving step iterates over a large domain with some expensive calculations like division and modulo, some of these can be avoided by using logarithms, it is the most time-consuming step of the algorithm even though it is not the theoretically most complex.
The sieving step can generally be optimized to reduce the algorithm's actual operating time the most.
In this step, it is simple to use a lot of memory, so one should be aware of this and try to reuse arrays and use the smallest data types possible.
Millions of items can make up the factor bases for record factorization, hence it is important to find the optimum on-disk/in-memory trade-off possible.
Finding useful relationships, i.e., elements (a, b) with the following characteristics, is the goal of the sieving stage.

- $\gcd(a, b) = 1$
- $a + bm$ is smooth over the rational factor base
- $b^{deg(f)} f(a/b)$ is smooth over the algebraic factor base

Different sieving techniques, such as the traditional line sieving or the quicker lattice sieving, can be used to find elements with these characteristics.
Line sieving is done by fixing b and sieving over a range of (a, b) for a $\in [-C : C]$.
This means that the algebraic norm over the AFB is computed for all values of (a, b), the rational norm is calculated for all values of (a, b), it is divided by elements from the RFB, and the entries with a result of 1 are smooth over the RFB.
It is not essential to verify each element on a sieving line against each element from

the RFB and AFB to see if they all factor entirely over the factor bases. A straightforward pattern determines how the elements on the sieving line that contain a specific element from the factor base as a factor are distributed along the sieving line.
• The elements with a on the form $a = bm + kp$ for $k \in \mathtt{Z}$ are those with a rational norm divisible by element (p, r) from RFB.
• The elements with a on the form $a = br + kp$ for $k \in \mathtt{Z}$ are those with algebraic norms divisible by elements (p, r) from AFB.
With these characteristics in mind, the goal behind the line sieving algorithm presented here is to take each element from the factor base and eliminate its contribution from the elements that have it as a factor.

Algorithm: **Line sieving**

```
input: RFB,AFB,QCB,polynomial $f(x)$, root m of $f(x) \mod n$, integer $C$
output: list of pairs $rels = (a0, b0), ..., (at, bt)$
1.$b = 0$
2. $rels = []$
3. while $rels < RFB + AFB + QCB + 1$
    (a) set b = b + 1
    (b) set $a[i] = i + bm$ for $i \in [-C; C]$
    (c) for each (p, r) ∈ RFB.
        i. Divide out largest power of p from $a[j]$ where $j = -bm + kp$ for k ∈ Z so that $-C \leq j \leq C$.
    (d) set $e[i] = b^{deg(f)} f(i/b)$ for i ∈ [-C; C]
    (e) for each (p, r) ∈ AFB
        i. Divide out largest power of p from $e[j]$ where $j = -br + kp$ for k ∈ Z so that -C ≤ j ≤ C.
    (f) for i ∈ [-C; C] i. if $a[i] = e[i] = 1$ and gcd$(i, b) = 1$ add $(i, b)$ to rels
    (g) $b = b + 1$
4. return rels.
```

end of the Algorithm

2.2.2.4 Speeding up the sieve
Lattice sieving, which John Pollard proposed in, is another frequently used sieving algorithm. It is comparable to the Quadratic sieve algorithm's special-q approach. The elements that are divisible by a big prime q are sieved after the factor bases are divided into smaller sets. By speeding up the filter, some smooth components are

lost.
Use of logarithms to avoid divisions is an obvious optimization that can be made for both the line- and lattice sieves.
This means that one should store the logarithm of the norms and then subtract $\log(p)$ instead of dividing it, due to one of the laws of logarithms

$$\log\frac{n}{p} = \log(n) - \log(p)$$

So we avoid the many divisions of large integers, but there are some issues one should be aware of under this optimization.

1. There is no way of detecting powers of primes p^j in the norms, so if $3^7$is a factor in the norm only $\log(3)$ will be subtracted.
2. Because of the floating point operations, correct elements can be bypassed and wrong elements can be included in the result.

A fuzz factor can help the logarithms in various ways to solve the first issue. If we subtract $\log(max(RFB))$ from all the entries, we should capture more of the elements that are divisible by prime powers without accepting additional incorrect ones since if a norm is not RFB-smooth, it must have a factor that is bigger than the largest prime in RFB.
Both of the aforementioned problems suggest that the components should be divided into trials to ensure that they are all RFB- and AFB-smooth.

2.2.2.5 Linear Algebra
We have a list of (a, b)'s that are RFB- and AFB-smooth as a result of the sieving stage, and we want to identify a subset of them that gives a square, i.e., a combination of relation set components whose product is a square.
We use this attribute to identify square elements. For a number to be a true square, the elements in its unique factorization must have an even power.
To clarify we can simply say that we have a list of numbers:

$$34, 89, 46, 32, 56, 8, 51, 43, 69$$

We want to find a subset of these numbers which forms a product that is a square, one solution is:

$$34, 46, 51, 69$$

With the product

$$34 * 46 * 51 * 69 = 5503716 = 2^2 * 3^2 * 17^2 * 23^2 = (2 * 3 * 17 * 23)^2$$

This is equivalent to solving a system of linear equations and can be done by building a matrix and eliminate it. The bottleneck in this operation is the dimensions of the linear system - which can result in a matrix with dimensions larger than $10^9 10^9$, which can be hard to represent in memory on a modern computer.
The matrix consists of the factorization over the rational- and algebraic bases and some information derived from the quadratic character base along with the sign of

the norm. Since we are only interested in the parity of the power of the elements in the factorization (even/odd), we put a 1 if the element appears as an odd power in the factorization and a 0 if its even or zero. This should be more clear from the algorithm below and the extended example.
This results in a matrix M over $GF(2)$ with dimensions $(relations).(RFB + AFB + QCB + 1)$.

Algorithm: **Building the matrix**

```
input: RFB,AFB,QCB,polynomial $f(x)$, root m of $f(x)$, list $rels = (a0, b0), ..., (at, bt)$ of smooth pairs
output: Binary matrix M of dimensions $(rels).(RFB + AFB + QCB + 1)$
1. set all entries in $M[i, j] = 0$
2. foreach (ai, bi) $\in$ rels
    (a) if $ai + bim < 0$ set $M[i, 0] = 1$
    (b) for each (pk, rk) $\in$ RFB i. let l be the biggest power of $pk$ that divides $a_i + b_i m$
        ii. if l is odd set $M[i, 1 + k] = 1$
    (c) for each (pk, rk) $\in$ AFB
        i. let l be the biggest power of pk that divides $(-bi)^d f(\frac{ai}{bi})$
        ii. if l is odd set $M[i, 1 + RFB + k] = 1$
    (d) for each $(pk, rk) \in QCB$
        i. if the Legendre symbol $\frac{a_i+b_i*p_k}{r_k} \neq 1$ set $M[i, 1 + RFB + AFB + k] = 1$
3. return M.
```

end of the Algorithm

Then, by putting this matrix into reduced echelon form, we can derive solutions which yield a square.
The process of solving linear systems by reducing a matrix of the system has been thoroughly studied since the birth of algebra and the most commonly used method is Gaussian Elimination. It is fast and easy to implement but has one drawback in this case; as the matrix is quite large, memory-usage becomes a problem.
There exist variations of Gaussian Elimination which utilize block partitioning of the matrix, but often a completely different method is used, namely the Block Lanczos method specialized for the GF(2) case by Peter Montgomery . It is probabilistic but in both theory and practice the algorithm is faster for large instances of the problem. It utilizes an iterative approach by dividing the problem into orthogonal sub-spaces.

we will only describe Gaussian Elimination which will work even for large problems if sufficient memory is available, but the interested reader should search for more details on the Block Lanczos method.

Algorithm: **Gaussian Elimination**

```
input: n × m matrix M
output: M on reduced echelon form
1. set i = 1
2. while i ≤ n
   (a) find first row j from row i to n where M[j, i] = 1, if none exists try with i = i
       + 1 until one is found.
   (b) if j ≠ i swap row i with row j
   (c) for $i < j < n$
       i. if M[j, i] = 1 subtract row i from rwo j
   (d) set i = i + 1
3. for each row $0 < j < n$ with a leading 1 (in column k)
   (a) for $0 < i < j$
       i. if M[i, k] = 1 subtract row j from row i
   4. return M.
```

end of the Algorithm

2.2.2.6 Square Root

From the linear algebra step we get one or more solutions, i.e. products which are squares and thereby can lead to a trivial or non-trivial factor of n. We need the square root of the solution, a rational square root Y and an algebraic square root X . The rational square-root is trivial, although the product of the solution from the linear algebra step is large, it is still an integer and a wide variety of methods for deriving the square root is known.

!

Œ.4ptwidth
!
!

Algorithm: **Rational square root**

input: n,polynomial $f(x)$, root m of $f(x)$, list of smooth elements the product of which is a square $deps = (a0, b0), ..., (at, bt)$
output: integer Y

```
1. compute the product $S(x) in Z[x]/f(x)$ of the elements in $deps$
2. return $Y = \sqrt{S(m).f'(m)^2} \mod n$
```

end of the Algorithm

Finding the square root of an algebraic integer is far from trivial, and is the most complex part of the GNFS algorithm but not the most time consuming one, and optimization of this step is therefore not important. Finding the square root of an algebraic integer is equivalent to finding the square root of a polynomial over an extension field. There exist algorithms for factoring polynomials in various fields but not many for factoring over a number field. Montgomery has once again developed a method for this in [15] and further refined in [16], but it is complex and uses lattice reduction which depends on a lot more theory than covered in this thesis. we will instead describe a method based on[17] and used in [18]. It does an approximation and uses a Newton iteration to find the true square root.

Algorithm: **Algebraic square root**

input: n,polynomial $f(x)$, root m of $f(x)$, list of smooth elements the product of which is a square$deps = (a0, b0), ..., (at, bt)$ **output:** integer X

```
1. compute the product$S(x)$ in $Z[x]/f(x)$ of the elements in $deps$
2. choose a large prime p (e.g. 2-5 times larger than n)
3. choose random $r(x) \in Zp[x]/f(x)$ with $deg(r) = deg(f) - 1$
4. compute $R_0 + R_1 y = (r(x) - y)^{\frac{p^d-1}{2}} \in (Zp[x]/f(x))[y]/(y^2 - S)$,
   i.e. compute the $\frac{p^d-1}{2}$ power of $r(x) \mod (y^2 - S)$.
5. if $SR_1^2 \neq 1$ go-to 2 and choose other p and/or $r(x)$
6. set $k = 0$
7. set$k = k + 1$
8. compute $R_{2k} = \frac{R_k(3-SR_k^2)}{2} \mod p^{2k}$
9. if $(R_k S)^2 \neq$ S go-to 7
10. compute$s(x) = \pm SR_k$.
11. return$X = s(m)f'(m) \mod n$
```

end of the Algorithm

2.3 An extended example

To clarify the different steps of the GNFS algorithm and to convince the reader,we will now go through the algorithm by providing an extended example.[14]
we want to factor the number n = 3218147, which should be tested for primality before proceeding with the factorization.

2.3.1 Setting up factor bases

The next step is to set up the factor bases. The sizes are chosen empirically and the main measure is to find the size which gives the fastest sieving as well as being small, because a smaller size in factor bases would give a smaller number of relations to work within the linear algebra stage.
we have chosen the primes below 60, and as described in Section **2.2.2.1** we store the prime mod m as well, so the rational factor base consists of the 16 elements.

$$RFB = (2,1)(3,0)(5,2)(7,5)$$

$$(11,7)(13,0)(17,15)(19,3)$$

$$(23,2)(29,1)(31,24)(37,6)$$

$$(41,35)(43,31)(47,23)(53,11)$$

Now we have to construct the algebraic factor base and we choose it to be approximately 3 times the size of the RFB. The AFB consists of elements (p, r), where $f(r) \equiv 0 mod p$, e.g. $f(7) \equiv 0 mod 17$ so (7, 17) is in the AFB, which consists of the following 47 elements

AFB = (2,0) (2,1) (3,2) (3,1) (5,1) (5,3) (13,10) (17,2) (17,16) (17,7) (19,9) (29,10) (31,0) (31,12) (31,3) (41,4) (41,8) (43,36) (43,23) (43,5) (53,40) (59,6) (61,12) (61,38) (61,41) (71,66) (79,28) (83,44) (89,9) (101,65) (103,87) (109,64) (109,85) (109,14) (127,95) (131,57) (131,108) (131,31) (149,49) (157,63) (163,155) (163,126) (179,165) (193,105) (197,93) (197,11) (197,191)

Even though we do not need the quadratic character base for the sieving step, we construct it here. It consists of the same type of elements as the AFB, but the elements are larger than the largest in the AFB. we choose to have 6 elements in the QCB

$$QCB = (233,166)(233,205)(233,211)(281,19)(281,272)(281,130)$$

After setting up the bases we have the following:

- $n = 3218147$
- $m = 117$
- $f(x) = 2x^3 + x^2 + 10x + 62$
- Factor bases: RFB,AFB,QCB

2.3.2 Sieving

Now we want to find elements that are **RFB**-smooth and **AFB**-smooth. This is done by sieving, as described in **2.2.2.2**
we start by selecting a line-width for the sieving, we choose to sieve with $a \in$ [-200; 200], then we follow the algorithm from **2.2.3**, which gives me the number of smooth elements wanted, e.g. the pair (13, 7) has rational norm

$$N_{rational}(a, b) = a + bm$$

$$\begin{aligned} N_{rational}(13, 7) &= 13 + 7117 \\ &= 832 \\ &= 2^6 13 \end{aligned}$$

i.e. it is smooth over the **RFB**. It has algebraic norm

$$N_{algebraic}(a, b) = (-b)^d f(-\frac{a}{b})$$

$$\begin{aligned} N_{algebraic}(13, 7) &= (-7)^3 f(-\frac{13}{7}) \\ &= -11685 \\ &= (-1)3.5.19.41 \end{aligned}$$

i.e. it is also smooth over the AFB, so (13, 7) is one of the desired elements.
In total we need at least 70 elements (RFB+AFB+QCB+1). we find the following72 elements

rels = (-186,1) (-155,1) (-127,1) (-126,1) (-123,1) (-101,1) (-93,1) (-68,1) (-66,1) (-65,1) (-49,1) (-41,1) (-36,1) (-31,1) (-23,1) (-12,1) (-9,1) (-7,1) (-6,1) (-5,1) (-3,1) (-2,1) (-1,1) (0,1) (2,1) (3,1) (4,1) (6,1) (7,1) (8,1) (13,1) (16,1) (19,1) (23,1) (24,1) (37,1) (81,1) (100,1) (171,1) (-65,3) (-62,3) (-31,3) (-8,3) (-1,3) (17,3) (26,3) (86,3) (181,3) (200,3) (-137,5) (-102,5) (-68,5) (-46,5) (-24,5) (-7,5) (7,5) (31,5) (39,5) (-152,7) (-83,7) (-44,7) (9,7) (13,7) (17,7) (18,7) (26,7) (41,7) (-64,9) (-17,9) (5,9) (11,9) (20,9)

After sieving we have the following:

- $n = 3218147$
- $m = 117$
- $f(x) = 2x^3 + x^2 + 10x + 62$
- Factor bases: RFB,AFB,QCB
- List of smooth elements rels

2.3.3 Linear Algebra

After the sieving step, we have a list of 72 elements which are smooth over the factor bases. Now we need to find one or more subset(s) of these which yield a product that is a square. This is done by solving a system of linear equations. The matrix we use is built as described in Section **2.2.2.3**, e.g. the element (13, 7) is represented in the following way ((13, 7) is the 63th element in **rels**). The first entry is the sign of$N_{rational}(13,\ 7)$ which is positive and therefore we have

$$M[63] = 0...$$

The next 16 entries are the factorization of $N_{rational}(13,\ 7)$ over RFB, and we only store the parity of the power, so we get

M[63] = 0000000010000000000 . . .

The next 47 entries are the factorization of $N_algebraic(13, 7)$ over AFB, and we store only the parity of the power, so we get

M[63] =
000000001000000000000101000001000010000000000000000000000000000000...

The last 6 entries are for the QCB. For each element(p_i, r_i)in the QCB we store a 1 if the Legendre symbol $\frac{13+7p_k}{r_k} \neq 1$ else a 0, so we get

M[63] =
000000001000000000000101000001000010000000000000000000000000000000100100
Doing this for all the relations gives the 72×70 matrix M

$$M = \begin{bmatrix}
1010000000100000001000100010000000000000000000100000000000000000000100101 \\
1100000010000000000100001000110000000000000000000000000000010000000100111 \\
1101000000000000000000100100000001100000000000010000000000000000000111101 \\
1000000000000000001000000000101000000000000000000000000000001000000110110 \\
1110000000000000000000010000100000000000000000000000100000000000000111100 \\
0000000000000000000101010100000000000000000000000000000000000000000100110 \\
0110000000000000000000001000000100000000100000000000000000000000100101111 \\
0000000000000000101001000001000000000000000000000000000000000000000011110 \\
0010000010000000001000100000100000000100000010000000000000000000000011110 \\
. \\
. \\
. \\
. \\
. \\
. \\
0000001000000000000010100000100001000000000000000000000000000000000100100 \\
0000010010000000000001000000010000000001000000000000000000000000000011111 \\
0010000000001000001000000000000000000000001000000000000000000000000001100 \\
0001000000000000001001000000000000010000000000010000000000000000000000111 \\
0001000000000001000001000010000010000000000010000000000000000000000010111 \\
0000000001000010010001010000000000000000000100100000000000000000000010100 \\
0000100000000100000000010000000000000000000000000000100010000000000100100 \\
0100000000000000000000000010000010000010000000000000000000000000000001111 \\
0100100010000000000000100000000000000010000000000000001000000000000000100 \\
0000000000000000000000000000000000000000000000000000000000000000000000000
\end{bmatrix}$$

The matrix is then transposed, an additional column is appended and we apply the

Gauss-Jordan algorithm to it, giving the result on reduced echelon form

$$M_e = \begin{bmatrix}
10000000000000000000000000000000000000000000000000000000000000000000000000100000 \\
01000000000000000000000000000000000000000000000000000000000000000000000000100000 \\
00100000000000000000000000000000000000000000000000000000000000000100001010100100100 \\
00010000000000000000000000000000000000000000000000000000000000000100010101101110100 \\
00001000000000000000000000000000000000000000000000000000000000000000111001010000 \\
00000100000000000000000000000000000000000000000000000000000000000001010010110000 \\
00000010000000000000000000000000000000000000000000000000000000000000000000000000 \\
00000001000000000000000000000000000000000000000000000000000000000000111000010100 \\
00000000100000000000000000000000000000000000000000000000000000000001111110000100 \\
00000000010000000000000000000000000000000000000000000000000000000001111011101100 \\
00000000001000000000000000000000000000000000000000000000000000000001110101000000 \\
. \\
. \\
. \\
. \\
. \\
. \\
00000000000000000000000000000000000000000000000000000000000001000001110101111000 \\
00000000000000000000000000000000000000000000000000000000000000100001111101010000 \\
00000000000000000000000000000000000000000000000000000000000000010000000111001000 \\
00000000000000000000000000000000000000000000000000000000000000000100111001011000 \\
00000000000000000000000000000000000000000000000000000000000000000011111011101100 \\
00000000000000000000000000000000000000000000000000000000000000000000000000000010 \\
00000000000000000000000000000000000000000000000000000000000000000000000000000000 \\
00000000000000000000000000000000000000000000000000000000000000000000000000000000 \\
00000000000000000000000000000000000000000000000000000000000000000000000000000000 \\
00000000000000000000000000000000000000000000000000000000000000000000000000000000 \\
00000000000000000000000000000000000000000000000000000000000000000000000000000000 \\
00000000000000000000000000000000000000000000000000000000000000000000000000000000 \\
00000000000000000000000000000000000000000000000000000000000000000000000000000000 \\
00000000000000000000000000000000000000000000000000000000000000000000000000000000 \\
00000000000000000000000000000000000000000000000000000000000000000000000000000000 \\
00000000000000000000000000000000000000000000000000000000000000000000000000000000 \\
00000000000000000000000000000000000000000000000000000000000000000000000000000000
\end{bmatrix}$$

From this we get a vector of free variables, i.e. elements which lead to a solution by setting one or more of them to 1.
$V_{free} = [00000000000000000000000000000000000000000000000000000000000000001001111111111101]$
After linear algebra we have the following:

- n = 3218147
- m = 117
- $f(x) = 2x^3 + x^2 + 10x + 62$
- Factor bases: RFB,AFB,QCB

- List of smooth elements rels
- Matrix M_e
- Vector of free relations V_{free}

2.3.4 Square roots

we now have different solutions to choose from by selecting one or more of the free variables fromV_{free}, i.e. we can choose to add one or more of the free variables to the solution. we take the first free variable and get the solution
$V_{sol} = [0011000000000000100010101100000001111100110000011010110001000000000000000]$
Which can easily be verified by $MeV_{sol} = 0$. So the following 20 elements are to be used

deps = (-127,1) (-126,1) (-12,1) (-5,1) (-2,1) (0,1) (2,1) (19,1) (23,1) (24,1) (37,1) (81,1) (-65,3) (-62,3) (86,3) (181,3) (-137,5) (-68,5) (-46,5) (31,5)

Now we apply the algorithm from 6.2.5, Ind we take it step by step. we start by computing the product $S(x) in Z[x]$

$$S(x) = (-127 + x)(-126 + x)...(-46 + 5x)(31 + 5x)$$

The rational square root Y is found by

$$\sqrt{S(m)f'(m)^2} = \sqrt{2049969700679090380346180211204354578841600006786134884}$$
$$= \sqrt{1391137089692141371945604152740435826018211007037440000}$$
$$= 1179464747117157958147891200$$

$$Y = \sqrt{S(m)f'(m)^2}$$
$$= 1179464747117157958147891200 \mod 3218147$$
$$= 2860383$$

From now we need the product $S(x)$ in $Z[x]/f(x)$ instead of $Z[x]$

$$S(x) = (-127 + x)(-126 + x)...(-46 + 5x)(31 + 5x) \mod f(x)$$
$$= \frac{-17470514047986971961535165201539075}{262144}.x^2 + \frac{3221408885540911801419827086788375}{131072}.x + \frac{116440133729799845699831534810433975}{131072}$$

For computing the algebraic square root X we need to do some more work. we start by choosing a random 128 bit prime p

$$p = 42.10^{42} + 43$$

we then choose a random $r(x) \in Zp[x]/f(x)$

$$r(x) = 331433955179089017205088896783327744131509664x^2 +$$
$$99351160498228990345472465287623463048288856x +$$
$$375233784773654088638092652579164104220841 20$$

As described in **2.2.2.4** we then calculate $R_0 + R_1 y \in (Z_p[x]/f(x))[y]/(y^2 - S)$ and get

$$R_1(x) = 40438443742646682162976058010107146767850928x^2 +$$

$$109758739765744771607769176310600913432507 04x +$$

$$2560245281777515905919468764456488101059368 3$$

This is usable, i.e. $SR_1^2 = 1$ so we compute R_2 and get
$R_2(x) =$ 14842804525348519321911887322528568600313069100589070521379460730026172213653606094254 53x^2 +
101243878338502139540877286172500592345110294552034268028685817452056177808996535271217 1x +
33707643386048967064886978071322595680303104670451605589553615208517742239145583274137
So we get
$S_{alg}(x) = \pm S(x)R_2(x) \mod p^2$
$=$

$$\pm(\frac{-17805551987379270x^2 + 460901612569132380 + 14073072352402140x}{262144})$$

And then we get the algebraic square root

$$X = S_{alg}(m)f'(m) \mod n$$

$$= 484534$$

So now we have Y and X and we can find the divisors of n

$$n_{divisor1} = \gcd(n, X - Y)$$

$$= \gcd(3218147, 484534 - 2860383)$$

$$= 1777$$

$$n_{divisor2} = \gcd(n, Y + X)$$

$$= \gcd(3218147, 2860383 - 484534)$$

$$= 1811$$

So we have computed that n is the product of **1777** and **1811**, and this is in fact the prime factorization.

2.4 Conclusion

In this chapter we had a close view about the GNFS algorithm and how it works we have detailed it step-by-step we have seen it's time complexity and most importantly we have shown a working example of this algorithm.
In the next chapter we are going to take this algorithm to work by implementing it in a c++ environment and testing how much time does this algorithm take for different key lengths.

Chapter3: GFNS Performances Evaluation

3 GFNS performances Evaluation

3.1 Introduction

This last chapter is an evaluation of GNFS performances under different parameters preceded by all the necessary steps for this process .

3.2 GMP library installation :

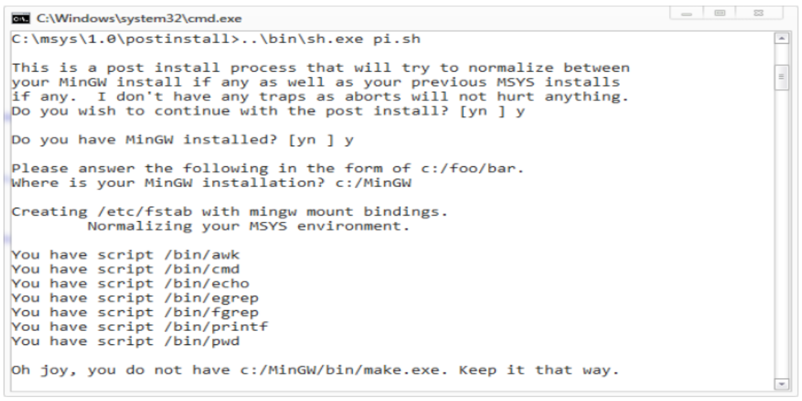
3.2.1 Before compiling

we want to make sure that MinGw is installed in our system which is going to be the case if we have CodeBlocks or Dev-Cpp rename the file “mingw32-make.exe” found in MinGW file to “make.exe” check that the path to the CodeBlocks or Dev-Cpp folder does not contain spaces, which may cause problems later. So avoid paths like ”C:/Program Files/CodeBlocks” and prefer simple paths like ”C:/CodeBlocks”.

3.2.2 MSYS installation

MSYS allows the compilation of applications or programs typically designed for UNIX systems. Can be downloaded using the link :

https://sourceforge.net/projects/mingw/files/MSYS/Base/msys-core/msys-1.0.11/MSYS-1.0.11.exe/download
after installation we get a command prompt and ask you if you want to continue the installation then if you installed MinGW. Answer with ”y” twice (see in figure 1)



```
C:\msys\1.0\postinstall>..\bin\sh.exe pi.sh

This is a post install process that will try to normalize between
your MinGW install if any as well as your previous MSYS installs
if any.  I don't have any traps as aborts will not hurt anything.
Do you wish to continue with the post install? [yn ] y

Do you have MinGW installed? [yn ] y

Please answer the following in the form of c:/foo/bar.
Where is your MinGW installation? c:/MinGW

Creating /etc/fstab with mingw mount bindings.
        Normalizing your MSYS environment.

You have script /bin/awk
You have script /bin/cmd
You have script /bin/echo
You have script /bin/egrep
You have script /bin/fgrep
You have script /bin/printf
You have script /bin/pwd

Oh joy, you do not have c:/MinGW/bin/make.exe. Keep it that way.
```

Figure 1: installation process

3.2.3 Install GMP and compile :

Now, we need to download the GMP library (zip with the extension .tar.bz2) and unzip it in an easily accessible folder (C:/gmp for example). Then, launch the previously installed MSYS software and move to the GMP folder:
cd C:/gmp

Then run the following 4 commands one after the other:
./configure
make

make check

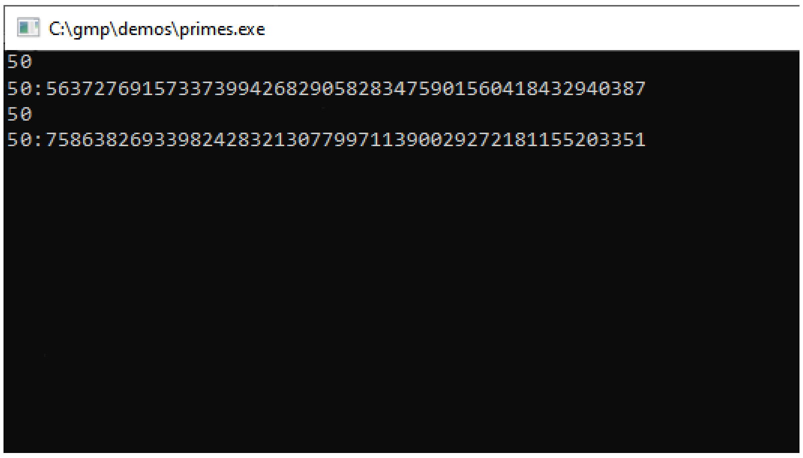
make install

3.3 GFNS performances Evaluation

We have effected factorization tests from RSA-100 (a composite number of the length of 100 digits) and up to RSA-155(as the factorization times ware exponentially growing)

3.3.1 Test 01 :

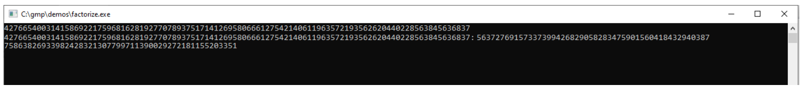
Primes generation :



```
C:\gmp\demos\primes.exe
50
50:56372769157337399426829058283475901560418432940387
50
50:75863826933982428321307799711390029272181155203351
```

Figure 2: Prime Generation 1

Composite factorization :



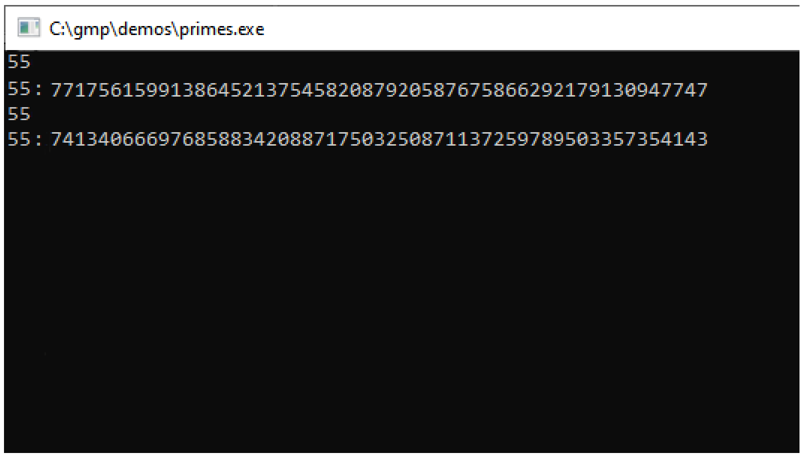
```
C:\gmp\demos\factorize.exe
4276654003141586922175968162819277078937517141269580666127542140611963572193562620440228563845636837
4276654003141586922175968162819277078937517141269580666127542140611963572193562620440228563845636837: 56372769157337399426829058283475901560418432940387
75863826933982428321307799711390029272181155203351
```

Figure 3: Factorization 1

Resault:
The factorization was accomplished in under 1 min .

3.3.2 Test 02 :

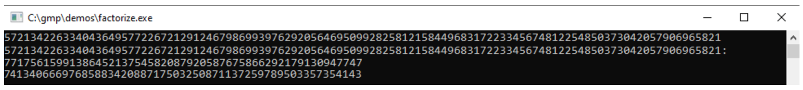
Primes generation :



```
C:\gmp\demos\primes.exe
55
55: 77175615991386452137545820879205876758662921791309477 47
55
55: 74134066697685883420887175032508711372597895033573541 43
```

Figure 4: Prime Generation 2

Composite factorization :



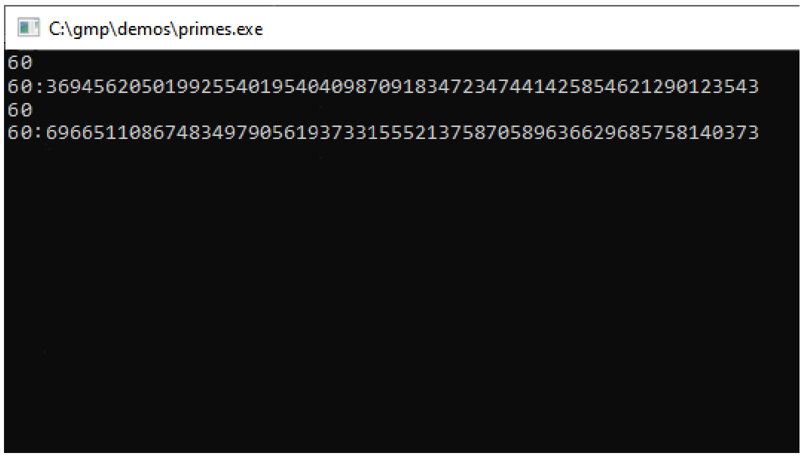
```
C:\gmp\demos\factorize.exe
5721342263340436495772267212912467986993976292056469509928258121584496831722334567481225485037304205790696582 1
5721342263340436495772267212912467986993976292056469509928258121584496831722334567481225485037304205790696582 1:
771756159913864521375458208792058767586629217913094774 7
741340666976858834208871750325087113725978950335735414 3
```

Figure 5: Factorization 2

Resault:
The factorization was accomplished in 15 min .

3.3.3 Test 03 :

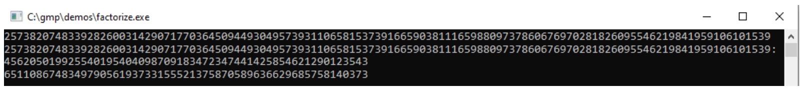
Primes generation :



```
C:\gmp\demos\primes.exe
60
60:369456205019925540195404098709183472347441425854621290123543
60
60:696651108674834979056193733155521375870589636629685758140373
```

Figure 6: Prime Generation 3

Composite factorization :



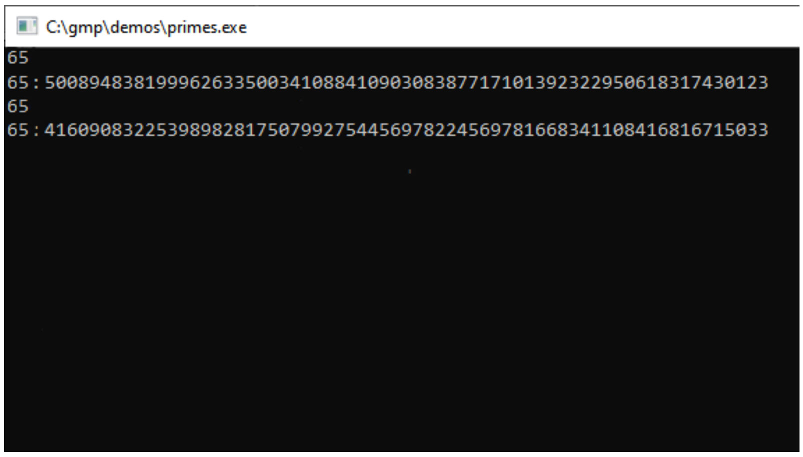
```
2573820748339282600314290717703645094493049573931106581537391665903811165988097378606769702818260955462198419591061015 39
2573820748339282600314290717703645094493049573931106581537391665903811165988097378606769702818260955462198419591061015 39:
4562050199255401954040987091834723474414258546212901235 43
6511086748349790561937331555213758705896366296857581403 73
```

Figure 7: Factorization 3

Resault:
The factorization was accomplished in under 75 min .

3.3.4 Test 04 :

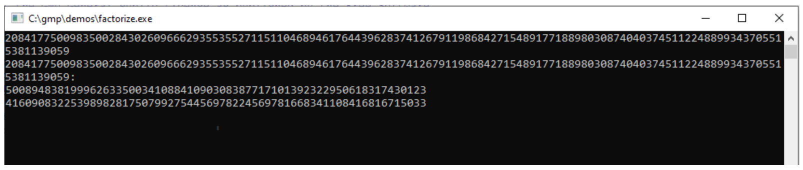
Primes generation :



```
65
65 : 50089483819996263350034108841090308387717101392322950618317430123
65
65 : 41609083225398982817507992754456978224569781668341108416816715033
```

Figure 8: Prime Generation 4

Composite factorization :



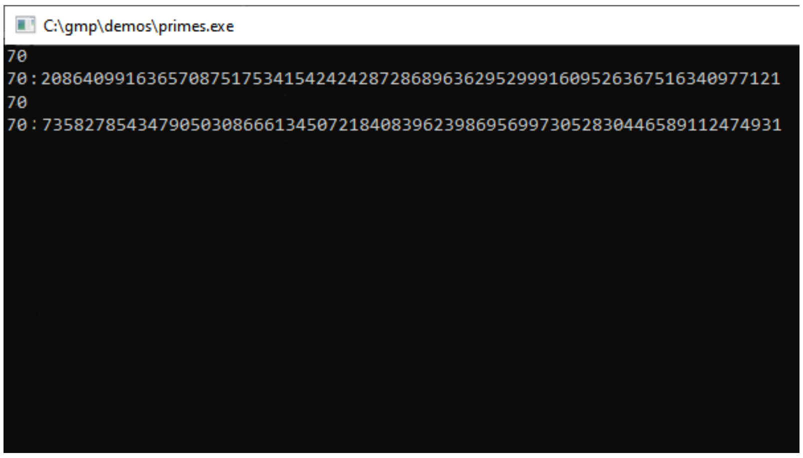
```
20841775009835002843026096662935535527115110468946176443962837412679119868427154891771889803087404037451122488993437055 1
5381139059
20841775009835002843026096662935535527115110468946176443962837412679119868427154891771889803087404037451122488993437055 1
5381139059:
50089483819996263350034108841090308387717101392322950618317430123
41609083225398982817507992754456978224569781668341108416816715033
```

Figure 9: Factorization 4

Resault:
The factorization was accomplished in under 3h and 20 min .

3.3.5 Test 05 :

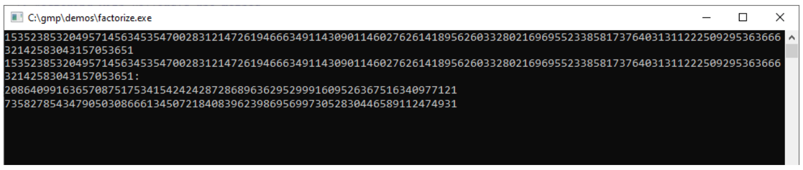
Primes generation :



```
C:\gmp\demos\primes.exe
70
70:2086409916365708751753415424242872868963629529991609526367516340977121
70
70:7358278543479050308666134507218408396239869569973052830446589112474931
```

Figure 10: Prime Generation 5

Composite factorization :



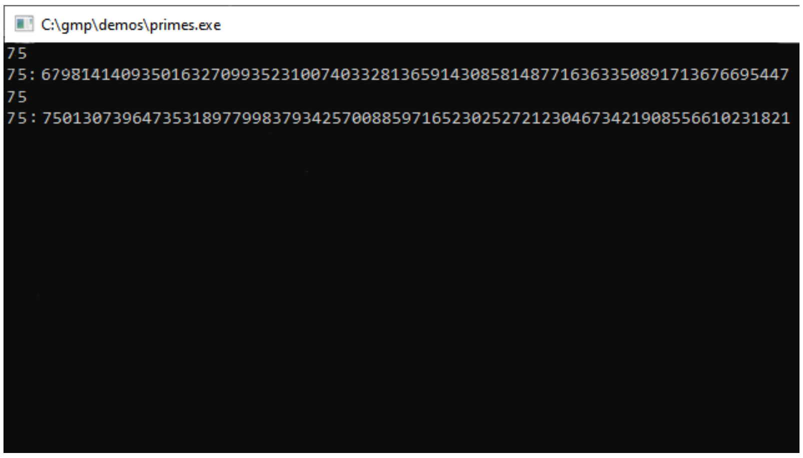
```
C:\gmp\demos\factorize.exe
153523853204957145634535470028312147261946663491143090114602762614189562603328021696955233858173764031311222509295363666
32142583043157053651
153523853204957145634535470028312147261946663491143090114602762614189562603328021696955233858173764031311222509295363666
32142583043157053651:
2086409916365708751753415424242872868963629529991609526367516340977121
7358278543479050308666134507218408396239869569973052830446589112474931
```

Figure 11: Factorization 5

Resault:
The factorization was accomplished in under 10 h 6 min .

3.3.6 Test 06 :

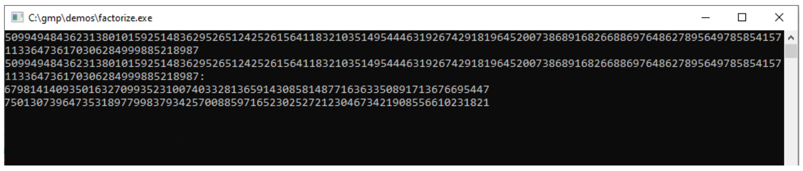
Primes generation :



```
C:\gmp\demos\primes.exe
75
75:679814140935016327099352310074033281365914308581487716363350891713676695447
75
75:750130739647353189779983793425700885971652302527212304673421908556610231821
```

Figure 12: Prime Generation 6

Composite factorization :



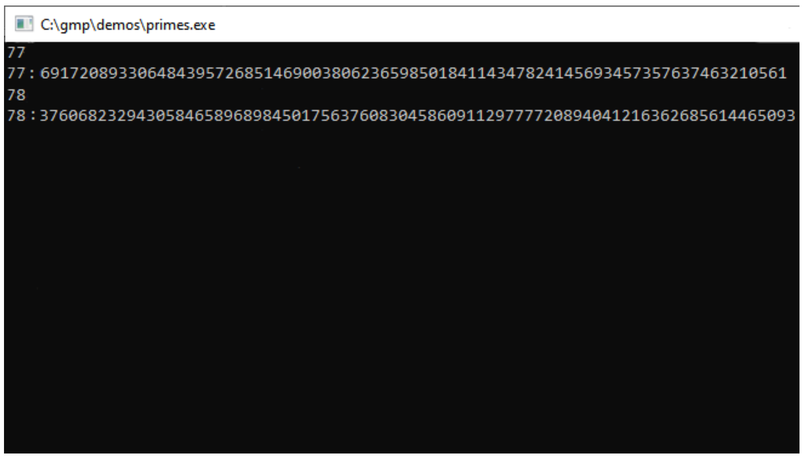
```
C:\gmp\demos\factorize.exe
509949484362313801015925148362952651242526156411832103514954446319267429181964520073868916826688697648627895649785854157
113364736170306284999885218987
509949484362313801015925148362952651242526156411832103514954446319267429181964520073868916826688697648627895649785854157
113364736170306284999885218987:
6798141409350163270993523100740332813659143085814877163633508917136766954 47
750130739647353189779983793425700885971652302527212304673421908556610231821
```

Figure 13: Factorization 6

Resault:
The factorization was accomplished in under 34 h and 25 min .

3.3.7 Test 07 :

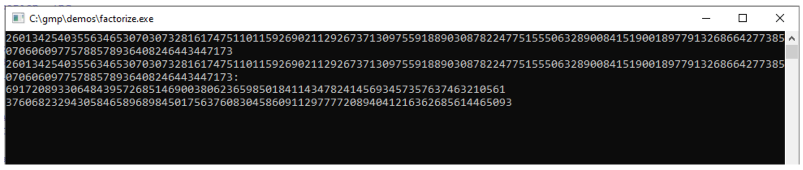
Primes generation :



```
C:\gmp\demos\primes.exe
77
77:69172089330648439572685146900380623659850184114347824145693457357637463210561
78
78:37606823294305846589689845017563760830458609112977772089404121636268561446 5093
```

Figure 14: Prime Generation 7

Composite factorization :



```
C:\gmp\demos\factorize.exe
2601342540355634653070307328161747511011592690211292673713097559188903087822477515550632890084151900189779132686642773 85
070606097757885789364082464434471 73
2601342540355634653070307328161747511011592690211292673713097559188903087822477515550632890084151900189779132686642773 85
070606097757885789364082464434471 73:
69172089330648439572685146900380623659850184114347824145693457357637463210561
37606823294305846589689845017563760830458609112977772089404121636268561446 5093
```

Figure 15: Factorization 7

Resault:
The factorization was accomplished in under 56 h and 37 min.

3.4 Efficiency of RSA Key Factorization :

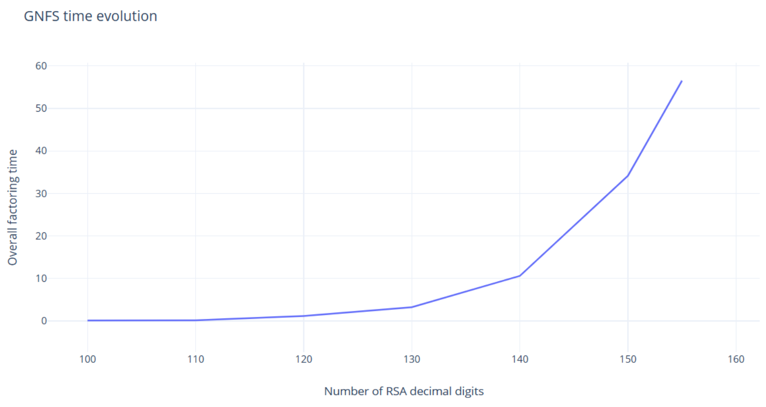


Figure 16: GNFS time evolution

the algorithm have shown very good performances considering the moderate computing power of an average computer CPU and successfully factored composites that are ably big .

Important notice: Important information: Using a computer with an Intel® CoreTM i5-11thG7 @ Processor, the RSA number with 155 decimal digits or 512 bits can be factored in fewer than 60 hours. The fact that the key can be cracked in three days shows how seriously hazardous the RSA-512 number is. We would require more than 15000 years to factor an RSA-1024 number if we followed the exponential curve and did the analysis. This RSA number has not yet been factorized by another author, and it is unlikely that our computer cluster will be able to factorize it anytime soon either.

3.5 Conclusion

GNFS is up to date the most efficient algorithm for composite factorization of big numbers and the tests done in this evaluation confirms the exponential complexity and both the efficiency and the inclusivity of the General Number Field Sieve algorithm.

4 General Conclusion and perspectives

The development of new factorization algorithms is vital for ensuring secure cryptographic systems and the work carried out within the framework of this dissertation was devoted to the General Number Field Sieve precedent by all notions related to modern cryptography and factorization problems and concluded by an evaluation of GNFS performances using GMP library and CodeBlocks and we have obtained it's results under deferent parameters.

In conclusion, the study and performance evaluation of the General Number Field Sieve (GNFS) for verifying the validity of modern cryptography techniques have provided valuable insights into the effectiveness and security of cryptographic algorithms. The research has shed light on the practicality of GNFS as a factorization method and its impact on the security of cryptographic systems that rely on the hardness of factoring large numbers. Through comprehensive analysis and experimentation, this thesis has contributed to the understanding of GNFS and its implications for modern cryptography.

Perspectives: Further Optimization: Despite the efficiency of GNFS, there is still room for improvement in terms of optimizing its algorithms and implementations. Future research could explore techniques to reduce the computational complexity and memory requirements of GNFS, making it more practical for larger numbers and enhancing its overall performance. Alternative Factoring Algorithms: While GNFS is currently the most efficient known algorithm for factoring large numbers, ongoing research focuses on developing alternative algorithms that could potentially challenge its dominance. Investigating these emerging algorithms, such as the Number Field Sieve (NFS) variant, may provide new perspectives on the security of modern cryptographic techniques. Post-Quantum Cryptography: With the advent of quantum computers, traditional cryptographic algorithms, including those relying on the hardness of factoring, face potential vulnerabilities. Investigating the impact of quantum computing on the GNFS and its implications for post-quantum cryptography is a promising avenue for future research. This could involve analyzing the resilience of modern cryptographic techniques against quantum attacks and exploring new post-quantum cryptographic algorithms. Real-World Application and Deployment: The practical implications of GNFS research on real-world cryptography applications are of utmost importance. Further studies can focus on evaluating the performance and security of cryptographic systems in practical scenarios, considering factors such as implementation details, hardware constraints, and attack models.

This research could contribute to the development of guidelines and best practices for deploying secure cryptographic solutions.

Cryptanalysis and Vulnerability Assessment: In addition to studying GNFS as a factorization method, further research could delve into cryptanalysis techniques to analyze the security of modern cryptographic techniques comprehensively. This includes exploring potential vulnerabilities, weaknesses, and attacks against cryptographic systems, providing valuable insights into their resilience and guiding the development of stronger cryptographic algorithms. By considering these perspectives, researchers can continue to advance the understanding, performance, and security of modern cryptographic techniques, ensuring the validity and effectiveness of cryptography in the face of evolving threats and computational advancements.

References

[1] Chapman and Hall/J. Katz /yLindel. Cryptography and network security. introduction to modern cryptography second edition.

[2] by by J. Orlin Grabbe. Symmetric key algorithms. guides.codepath.org, visited 26/05/2023.

[3] by by J. Orlin Grabbe. The des algorithm illustrated. page.math.tu-berlin.de, visited 26/05/2023.

[4] Kate Brush/ Linda Rosencrance/ Michael Cobb. asymmetric cryptography (public key cryptography. https://www.techtarget.com/searchsecurity/definition/asymmetric-cryptography, consulted : 18 February 2023.

[5] Aanchal kumari. What is asymmetric key cryptography?

[6] by Gustavus J. Simmons. Cryptography communication system and method, rivest-shamir-adleman encryption.

[7] by marry K.pratt. cyber attacks.

[8] Prof. Ingole R.Y Marne Gauri. A review on maintaining web applications and brute force attack by prof.

[9] fortinet.com. brute force attack definition. https://www.fortinet.com/, consulted : 19/05/2021.

[10] G. I. Davida. Chosen signature cryptanalysis of the rsa (mit) public key cryptosystem. technical report tr-cs-82-2.

[11] S. Micali S. Goldwasser and P. Tong. *Why and how to establish a private code on a public network (pages 134-144)*. 23rd IEEE Symp. on Foundations of Comp, 1982.

[12] J. Hastad and M. N¨aslund. The security of individual rsa bits. 1988.

[13] J. Hastad and M. N¨aslund. The security of individual rsa bits. 1988.

[14] Daniel J. Bernstein and Arjen K. Lenstra. A general number field sieve implementation.

[15] Brian Antony Murphy. Polynomial selection for the number field sieve integer factorisation algorithm. PhD - thesis (1999).

[16] Peter L. Montgomery. Square roots of products of algebraic numbers, mathematics of computation 1943–1993: a half-century of computational. 1994, pp. 567–571.

[17] Phong Nguyen. A montgomery-like square root for the number field sieve, lecture notes in computer science 1423. (1998),pages 151–??

[18] Jr. Hendrik W. Lenstra Joe P. Buhler and Carl Pomerance. Factoring integers.