University of Nottingham
UK | CHINA | MALAYSIA

Design and Analysis of a Quasigroup-based DNA Encryption Scheme

Thesis submitted to the University of Nottingham for the degree of *Master of Philosophy*

July 2025

Tiong Yih Lian

20619202

Supervisors

Dr. Chee Wing Loon

Dr. Liew Kian Wah

School of Mathematical Sciences

University of Nottingham Malaysia Campus

ACKNOWLEDGEMENT

I would like to express my deepest gratitude to everyone who has supported me throughout the journey of researching and writing this thesis.

First and foremost, to God, my anchor in every season. I give thanks to God, by whose grace and through faith, I have been guided through every challenge in my pursuit of knowledge. Thank you for lighting my path when the words would not come and for turning doubt into faith

To my parents – my forever safe place and loudest cheerleaders, I owe and immeasurable debt of gratitude. You are always here every step of the way on my path to education, and for that I am forever grateful. This achievement is as much yours as it is mine. I love you both to the moon and back.

I would also like to extend my profound thanks to my supervisors, Dr. Chee and Dr. Liew, whose expertise and critical insight were indispensable to this research. Your ability to navigate complex mathematical landscapes with clarity, your willingness to engage in deep discussions, and your constructive feedback has made this thesis possible.

Last but certainly not least, I gratefully acknowledge the support of University of Nottingham Malaysia's School of Mathematical Sciences. The university has given me access to invaluable resources and a comfortable environment for me to focus on my research. I would also like to thank the administrative and technical staff of the Graduate School for their assistance.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

ABSTRACT

DNA cryptography is an interdisciplinary field of cryptography inspired from DNA computing which uses DNA molecules' role as information carrier for cryptographic purposes. In this thesis, we present an improvement on the existing algorithm with the implementation of quasigroup in the process of encryption and decryption of DNA cryptography. As opposed to traditional cryptography, which is based on numerical values, the proposed scheme makes use of DNA bases as elements of a quasigroup and unlike conventional approaches that rely solely on standard DNA bases (A, T, C, G), the proposed method introduces a DNA base U as an additional element, which appears only in the process of encryption. The encryption process involves 2 phases, namely Phase I, in which the DNA form of the plaintext undergoes transformation through a randomly generated leader and a quasigroup of order 5, and Phase II, in which the process repeats itself but the quasigroup is replaced by one of its random parastrophes. The utilisation of quasigroup operations for the proposed cryptographic scheme provides a mathematical foundation for data transformation. Notably, since the total number of quasigroups of order n increases exponentially with n, this makes them advantageous for constructing cryptosystems with extensive key space, thus ensuring enhanced security without increasing computational complexity. In summary, this thesis proposes a novel, two-phase cryptographic scheme that successfully integrates quasigroup operations with DNA encoding. The introduction of the Uracil base and the use of parastrophes were shown to produce ciphertext with near-ideal

entropy, providing enhanced security against statistical attacks while maintaining linear-time efficiency suitable for larger plaintexts.

CHAPTER 1: INTRODUCTION

1.1 Organisation of Chapters

This thesis is structured into six chapters. Chapter 1 introduces the fundamental concepts of cryptography which includes classical and modern cryptographic system and provides background information on DNA cryptography and quasigroup-based cryptography. The motivation for integrating quasigroup operations in DNA-based cryptographic system is also outlined in this chapter. Chapter 2 presents a comprehensive literature review on the history and related works in cryptography, DNA cryptography and quasigroup-based cryptography. Chapter 3 outlines the methodology adopted in this chapter. It covers the basics of DNA cryptography (such as DNA encoding and decoding rules), quasigroup theory (including definitions, properties and parastrophes), and the encryption and decryption functions. It also introduces the proposed encryption scheme. Chapter 4 focuses on the implementation of the proposed system, including the algorithm design and demonstrations. Chapter 5 covers the results and discussion of the study. This includes the security analysis of the proposed method, performance evaluation through efficiency and complexity analysis as well as comparative analysis with selected existing DNA and quasigroup-based cryptosystem along with trade-offs in quasigroup size and encryption phases. Finally, Chapter 6 concludes the thesis by summarising the contributions, highlighting limitations and suggesting directions for future work, followed by references and appendices.

1.2 Introduction to Cryptography

Cryptography is the pillar of modern information security, which is crucial for ensuring the confidentiality, integrity and authenticity of digital communication. Confidentiality guarantees that only authorised individuals can access the encrypted data. Integrity assures that no alterations are made to the message during transmission, while authenticity ensures that the message transmitted is genuine and originates from a trusted source.

Living in a digital era where data security has become an overwhelming concern, there is a constant need for innovative cryptographic systems to safeguard personal, corporate and government data. At present, modern cryptosystems are expected to maintain strong security measures without sacrificing efficiency. The growing interest in cryptosystems which are both computationally simple and cryptographically strong continues to increase as most security environments possess limited storage and processing power.

Generally, cryptography is broadly divided into two classifications, classical cryptography, which predates the 1980s, and modern cryptography, which has developed in the years since.

1.3 Classical Cryptography

Classical cryptography is more commonly known as “breakable” ciphers as they are designed in a nonrigorous way which causes them to be terribly vulnerable to various attacks [1]. The methods in this type of cryptography primarily relied on manual techniques such as pen-and-paper ciphers or early computers. There are two main types of classical cryptography:

substitution ciphers and transposition ciphers. In short, substitution ciphers are ciphers where each letter of the plaintext is replaced by another, and transposition ciphers are ciphers where the letters arranged in different orders [2], [3]. The more commonly known examples of these ciphers are Caesar cipher, Vigenère Cipher and Scytale cipher. While these classical ciphers have provided a strong foundation for early cryptographic techniques, they have become extremely susceptible to attacks with the introduction of modern computers which has the ability to solve complex problems with great speed.

1.4 Modern Cryptography

In modern cryptography, cryptosystems are developed based on complexity theory. In simpler terms, complexity theory is the theory of computational difficulty of a given problem, some prime examples of difficult problem include integer factorisation problem and discrete logarithm problem [4]. Two major types of cryptographic systems in modern cryptography are symmetric key cryptography and asymmetric key cryptography. In symmetric key cryptography, one single key is used for both encryption and decryption processes [5]. Symmetric key cryptography is infamous for being efficient and fast, however, it requires a secure channel for key transmission, which can be a limitation in some scenarios. Examples of such systems include Data Encryption Standard (DES) [6] and its more advanced successor, the Advanced Encryption Standard (AES) [7], which has become the global standard for secure data encryption. For asymmetric key cryptography, also known as public key cryptography, the process requires a pair of keys, one public and the other private [5]. Asymmetric key cryptography ensures that even if the public key is widely shared, only the holder of the private key can decrypt the information.

RSA and Elliptic Curve Cryptography (ECC) are examples of such cryptography. Modern cryptography has evolved from the principles of classical cryptography and plays a crucial role in meeting the security needs of the current advanced digital era.

1.5 Basic Concepts

It has been a known fact that cryptography has been utilised throughout decades for purposes of secure communication between two parties. In its simplest form, two individuals who want to communicate with each other are commonly referred to as Alice and Bob. When Alice, A wishes to convey a secret message to Bob, B, they will both agree on a cryptographic method and a shared secret key. The key is used to convert the original message (plaintext) into unintelligible text (ciphertext). This process is called encryption. Bob, who has a key in possession as well, is able to decipher the text back to its original form. This process is referred to as decryption. The scenario above is described more clearly using the following flow map:

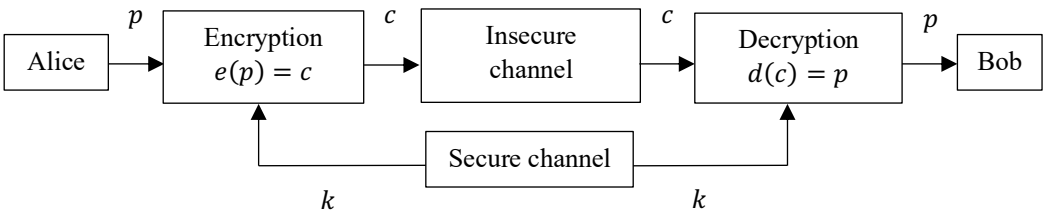

Figure 1.1 Flow Map of Encryption and Decryption Process

where p is plaintext, c is ciphertext, k is key, e is encryption function and d is decryption function. The message is transmitted through an insecure channel whilst the key is distributed through a secure channel between Alice and Bob.

Cryptanalysis, on the other hand, is the practice of analysing and breaking cryptosystems. While cryptography aims to protect information, cryptanalysis seeks to exploit weaknesses. In cryptography, an adversary is a malicious entity, which aims to uncover confidential information or data. It has always been a cryptographer's instinct to assume that adversaries are able to intercept the insecure channel to retrieve any information [8], [9]. Hence, in order to protect secret data, it is always crucial to not rely heavily on a simple and straightforward algorithm to encipher and hide the data to be conveyed.

1.6 Types of Attacks in Cryptanalysis

Adversarial Models/ Capabilities

In cryptographic systems, it is necessary to formally define the capabilities and goals of potential adversaries in order to establish rigorous security notions which allows for provable guarantees about a scheme's resilience against various attacks [10]. This section therefore distinguishes between two key aspects: (i) Adversarial Models, which describe the level of access an adversary has to the cryptosystem; and (ii) Adversarial Goals, which describe the specific security property the system must satisfy under such attacks. Together, these formalise the security notions used to assess the robustness of a cryptographic scheme.

1.6.1 Adversarial Models (Capabilities)

Adversarial models are often described to be the information and level of interaction an attacker may have with the cryptographic system, ranging from passive observation to active manipulation. These capabilities determine the

adversary's strength and the cryptosystem's threat landscape. The following adversarial capabilities are considered:

Ciphertext-Only Attack (COA)

In a COA, the attacker only has access to a sequence of ciphertexts. The attacker aims to break the system by only observing the ciphertexts. In [11], it is stated that a cryptosystem is deemed completely insecure if it is not resistant against this attack. In addition, any public-key encryption scheme must be secure against chosen-plaintext attacks (CPA); otherwise, it cannot be considered a practical cryptosystem.

Known-Plaintext Attack (KPA)

The attacker possesses both the plaintext and corresponding ciphertext in a KPA. With access to this information, the attacker could attempt to reverse-engineer the encryption process by using frequency analysis. Some examples of KPA include linear and differential cryptanalysis [12].

Chosen-Plaintext Attack (CPA)

In a chosen-plaintext attack, the attacker is able to temporary infiltrate the cryptosystem and choose arbitrary plaintexts and obtain their corresponding ciphertexts from the encryption system. Generally, if a cryptosystem is breakable under a weaker model (e.g., known-plaintext attack), then it is also breakable under stronger models (e.g., chosen-plaintext attack) [13].

Chosen-Ciphertext Attack (CCA)

In this scenario, the attacker can choose arbitrary ciphertexts and obtain their decrypted plaintexts. By iteratively modifying the ciphertext and observing the

changes in the decrypted output, the adversary can gradually recover the secret message [11].

Brute-Force Attack

A brute-force attack involves systematically trying all possible keys until the correct one is found. The feasibility of this method depends on the size of the key space. This attack becomes infeasible for algorithms with large key spaces as it can be time-consuming. Modern encryption schemes are designed to have key lengths that make brute-force attacks computationally infeasible within a reasonable time frame [9], [14].

These models form a hierarchy of adversarial strength, where CCA is strictly more powerful than CPA, and so forth. Demonstrating security against a stronger adversarial model inherently implies security against weaker ones.

1.5.2 Adversarial Goals (Security Notions)

Beyond defining what adversaries can do, it is also important to specify what the attacker aims to achieve under these attack models. The security of a cryptosystem is evaluated with respect to indistinguishability properties, which formalise the confidentiality requirement: ciphertexts should not reveal any meaningful information about the corresponding plaintexts. This is expressed through standard game-based security notions.

Indistinguishability under Chosen-Plaintext Attack (IND-CPA)

The adversary gains access to an encryption oracle and attempts to distinguish ciphertexts of chosen messages [15]. This is the minimal acceptable security for encryption schemes, ensuring semantic security under passive attacks.

Indistinguishability under Chosen-Ciphertext Attack (IND-CCA)

IND-CCA security extends IND-CPA security by granting the adversary access to a decryption oracle, except for the challenge ciphertext [15]. A cryptosystem satisfying IND-CCA security maintains confidentiality even in the presence of adversarial tampering or partial compromise of the decryption process.

Both IND-CPA and IND-CCA are consistent with semantic security, which asserts that ciphertexts reveal no partial information regarding plaintexts. The standard security goal in modern cryptography is indistinguishability, the inability of an adversary to distinguish between encryptions of two different messages. These indistinguishability-based notions formalised confidentiality in modern cryptography. A scheme that satisfies IND-CCA is also secure against all weaker adversarial models, while IND-CPA security guarantees protection in contexts where only encryption oracle access is available.

1.7 DNA Cryptography

DNA cryptography possesses many potentials from its high storage capacity to massive parallelism. The idea of DNA cryptography stems from the properties of DNA molecules to store, process and transmit information. Basically, DNA cryptography functions on the concept of DNA computing which utilises 4 DNA bases i.e. Adenine (A), Guanine (G), Cytosine (C) and Thymine (T) to perform computations. The concept for DNA information storage was first proposed by the American physicist, Richard P. Feynman in 1959 [16]. In the current system, the information is encoded in binary form before being encoded into DNA form through DNA synthesis technology and

subsequently stored. As we approached the 21st century, with the development of 5G, Internet of Things (IoT) and artificial intelligence (AI), high density and long-term storage solutions have become a worrying necessity. The global data volume is expected to reach 175ZB (zettabytes) by 2025 according to the Internet Data Center (IDC) and will continue to grow with an annual rate of 31.8%, far exceeding the storage capacity of any currently available storage solutions [17]. As we reach 2025, forecasts suggest that this trend will only continue and by 2040, the global datasphere could reach as much as 5000ZB if the current growth rate persists [18]. DNA storage, in theory, can encode two bits per nucleotide (nt) or 455 exabytes (1 billion gigabytes) per gram of single-stranded DNA maximum. To put this into perspective, one cubic decimetre of DNA solution has the potential to store one trillion bits of binary data [19]. It is far denser compared to any traditional storage media, which proves its great potential in cryptographic applications.

Compared to traditional silicon-based computers, DNA computers possess certain advantages, most notably its massive parallelism, high data density and minimal energy requirement [20]. Unlike traditional computers which process tasks sequentially, DNA computers have the ability to perform many operations simultaneously. Millions of DNA sequences can be synthesised and read at the same time, which enable simultaneous processing of vast amounts of data. The computing speed of a DNA computer can reach up to 1 billion times per second and its power consumption is only equal to one-billionth of a traditional computer [19], [21]. This property of DNA computing allows it to solve many complex problems faster than any conventional method. For instance, Adleman's seminal 1994 experiment solved a seven-node

Hamiltonian path problem in a single biochemical reaction using DNA strands to represent graph nodes and edges [22]. The molecular processing occurred in parallel, with trillions of molecules exploring all possible paths simultaneously, achieving in minutes what digital computers would take far longer to enumerate [22].

As mentioned in [23], various research is currently at works over the world to introduces novel DNA cryptographic approaches and improve current methods in this field. It is also stated that in coming decades when DNA computers are available, it would be able to replace the current silicon-based technology. It should also be noted that Luca Cardelli from Microsoft has taken the lead to explore the field of DNA computing [23]. However, it is a fact that DNA cryptography is still at infancy stage and is faced with a myriad of unresolved challenges. Even with a growing number of researchers contributing to the field, it has yet to achieve maturity in either theory or practical. Current DNA-based methods depend heavily on advanced laboratory procedures, and there is yet a unified theoretical framework for employing DNA molecules in cryptographic applications [24].

1.8 Quasigroup-based Cryptography

Quasigroup-based cryptography, a cryptographic technique built upon nonassociative algebraic structures known as quasigroups, whose flexible structure and large size had deemed it suitable for designing lightweight, high-speed and efficient cryptosystems [25]. Although less commonly used than groups or fields (some well-studied algebraic structures in mathematics),

quasigroups possess distinctive features that make them highly suitable for modern cryptographic applications.

The main factor which allows quasigroup theory to be applied in the field of cryptography is vast number of quasigroup operations over a given finite set [26]. The third party would face difficulty in uncovering the encrypted message if these operations are used to define the encryption and decryption processes [26]. Unlike the usual group-based cryptosystems, where operations tend to follow predictable patterns due to properties such as associativity and the existence of identity elements. Quasigroup-based systems present no such regularities as each element in a quasigroup table (Latin square) appears only once per row and column, thus ensuring that the transformation of input symbols yields unique and non-repeating results. This nonlinearity significantly increases resistance to attacks such as linear and differential cryptanalysis which exploit structural patterns in encryption schemes. For instance, the INRU cipher utilises quasigroup-based string transformations to achieve high nonlinearity which strengthens the system against linear and differential cryptanalysis [27]. There are also several other quasigroup-based methods which have been proven to be resistant against differential cryptanalysis [27], [28], [29], [30].

Quasigroup-based cryptography is also a strong candidate for lightweight cryptography, especially in resource-constrained environments such as embedded systems, wireless sensor networks, and Internet of Things (IoT) devices [31]. There is a study on an efficient quasigroup block cipher which highlights its low memory and computational requirements, rendering them suitable in resource-constrained settings [32]. Moreover, the construction of

cryptographically strong 4×4-bit S-boxes using quasigroups of order 4 has been proposed as a method for lightweight cryptographic applications [33].

Traditional cryptographic algorithms often rely on group-based structures or number-theoretic problems, which, while effective, may face limitations in computational efficiency or vulnerability when it comes to future quantum attacks. As an example, widely used cryptosystems such as RSA and ECC are susceptible to quantum attacks whilst quasigroup-based cryptographic schemes are able to present alternative approaches that may resist quantum attacks more effectively. In the work by Nager D. in 2021 [34], the proposed Xifrat cipher, which is based on multiple quasigroups with restricted commutativity, is shown to have a quantum attack complexity of approximately 2^{118}. This is significantly higher than the 2^{64} quantum attack complexity of AES-128 under Grover's algorithm, thereby suggesting stronger post-quantum security. This result underscores the promising potential of quasigroup-based cryptosystems not only in modern cryptography but also in post-quantum cryptographic design, particularly in symmetric key environments where lightweight and efficient structures are needed without compromising security [34].

In addition to their cryptographic strength, quasigroups have also contributed to parallel processing and high-speed encryption, which are increasingly important in today's digital landscape. Due to their nonassociative nature, quasigroup operations allow each symbol in a message to be encrypted independently of others. In contrast to group-based operations which often depend on the outcomes of previous computations, quasigroup-based transformations can be applied across all data points simultaneously, enhancing

overall efficiency without compromising the security. A notable example is the Multivariate Quadratic Quasigroup (MQQ) cryptosystem, which has demonstrated exceptional performance in terms of encryption and decryption speeds [35]. Implemented on four Xilinx Virtex-5 FPGA chips running at 276.7MHz, the MQQ achieves an encryption throughput of 44.27 Gbps, which is 10,000 times faster than RSA implementations on similar FPGA platforms. This remarkable speed is attributed to the efficient use of quasigroup-based transformations, which facilitate parallel processing and high-speed encryption. Other research has also shown that quasigroup-based S-boxes can be implemented efficiently in hardware with reduced area and power consumption [36]. For instance, a study demonstrated over a 40% area reduction compared to lookup table-based implementations and more than a 16% area reduction in a parallel implementation of the PRESENT cipher. These efficiencies are due to the properties of quasigroups which allow for parallelisable operations and compact hardware designs. The MQQ stream cipher, which combines a linear feedback shift register (LFSR) with a quasigroup filter, is another example of a high performance quasigroup-based encryption system [37]. The quasigroup filter enhances the cipher's performance by enabling parallel processing and efficient data handling, making it well-suited for high-speed encryption applications.

1.9 Motivation of Using Quasigroup with DNA Cryptography

With the development of DNA computers, DNA cryptography does provide massive parallelism by enabling simultaneous operations on multiple DNA bases. However, for applications with groups, parallelism is partially limited due to structural constraints caused by group properties. Groups have

certain algebraic properties that must always hold, such as associativity, existence of identity element and inverses. To maintain these properties, group operation often depends on prior results. Therefore, group-based cryptographic systems often force sequential dependencies in their operations, thus making it hard to process all elements simultaneously and independently, even if DNA computing's parallelism is available. With that being said, for groups, DNA computing's parallelism can still be applied across multiple DNA strands. For example, if you have 1000 DNA sequences, you can process each sequence simultaneously, but within each sequence, the group operation remains constrained by sequential dependencies. Now compared to quasigroups, since quasigroups do not require associativity, the transformation of each base is independent of others. Hence, DNA's parallelism can be fully exploited by transforming all bases across all strands simultaneously.

In addition to parallelism, security through nonlinearity is another reason for selecting quasigroups. As previously mentioned, a quasigroup operation output does not follow predictable patterns based on the input and such nonlinearity makes the relationship between plaintext and ciphertext highly complex. Groups, however, are associative by definition. This causes linear dependencies between operations. With the presence of an identity element, the operation might produce ciphertext with predictable patterns.

Although both groups and quasigroups are able to offer large key space, groups, however, will become slower as the size increases due to the structural constraints. In comparison, quasigroups can provide a larger and more flexible key space. Additionally, parastrophes of quasigroups allow for multi-phase encryption which further complicates the system for attackers. The concept of

parastrophes is specific to quasigroups. In groups, the binary operation is fixed and cannot be rearranged while preserving group properties, hence, no parastrophes.

1.10 Aims and Objectives

This thesis aims to develop, implement and evaluate a novel hybrid DNA cryptosystem which integrates quasigroup operations to enhance both security and efficiency. The main objectives of this research are outlined as follows:

1. To design a quasigroup-based DNA encryption scheme.

The main objective of the research is to develop a novel and unconventional encryption scheme which involves the properties of both quasigroups and DNA bases. Many researchers have been exploring new methods for encoding and decoding hidden messages in DNA sequences. The proposed method shares with prior DNA-based and quasigroup-based methods the foundational principles. Like previous DNA cryptosystems, it uses DNA encoding rules to convert plaintext into symbolic biological representations. Similarly, it adopts the core concept from quasigroup cryptography which is the quasigroup operations to generate nonlinear substitutions that are difficult to invert without the correct quasigroup table. The proposed work further expands these principles by combining the two previously separate domains into a unified framework. Unlike traditional DNA-based schemes, which rely primarily on biological encoding and complementary pairing but lack strong mathematical mechanisms to enhance confusion. In contrast, quasigroup-based cryptographic systems provide algebraic nonlinearity and have large key spaces,

yet they have not been integrated with DNA representations. This thesis bridges these two domains by combining DNA encoding with quasigroup and parastrophe transformations to achieve both biological-inspired parallelism and mathematically grounded security, thereby extending the current scope of DNA cryptography research.

This thesis seeks to contribute to ongoing research and development which are essential and necessary for realising the full potential of interdisciplinary encryption techniques.

2. To analyse the security of the proposed scheme against potential vulnerabilities and attacks.

One of the aims of the research is also to conduct a thorough analysis of the security of the proposed system which will include examining its resistance to common attacks such as brute-force attack and known plaintext attack through statistical analysis. Key parameters such as Shannon entropy and normalised entropy are used to evaluate the robustness of the system. We will hopefully be able to identify any inherent vulnerabilities in the system and discuss how these vulnerabilities could be exploited by attackers.

3. To evaluate the computational efficiency and complexity of the scheme

An essential goal of the research is to demonstrate the computational efficiency of the algorithm. The thesis measures the encryption and decryption time across various plaintext lengths, as well as analysing the time and space complexity of the system. It also explores the trade-offs between performance and cryptographic strength, particularly when quasigroup order and encryption phases increases.

4. To compare the proposed scheme with existing cryptographic schemes.

The proposed method is compared against established DNA and quasigroup based methods such as those by Padmapriya [38] and Markovski [39], to evaluate improvements in efficiency, entropy and overall effectiveness. These comparisons aim to position the proposed algorithm as a viable alternative for secure communication.

CHAPTER 2: LITERATURE REVIEW

2.1 Development of Cryptography and Related Works

'Cryptography', derived from the Greek words 'Krypto' and 'graphene', translate to 'secret' and 'writing' respectively. The roots of cryptography can be traced back to ancient Roman and Egyptian civilisations. The earliest known use of cryptography dates back to 1900 BCE with the use of hieroglyphs among Egyptians [1]. The hieroglyphic symbols were discovered to be carved in the chamber of the tomb of Khnumhotep II, an ancient Egyptian Great Chief in Egypt. These hieroglyphic symbols, carved on tomb walls, served not only as artistic and ceremonial purposes but also as encoded secret messages. Fast forward to 100 BCE, cryptography had evolved further in ancient Rome, when Julius Caesar, a Roman general, developed a simple substitution cipher, known as Caesar Cipher [40]. This simple substitution cipher involves shifting each letter in a plaintext by a fixed number of positions in the alphabet. The cipher was a way for Julius Caesar to send covert military orders to his generals in the field so that even in the event of the messages being intercepted, it would still remain unreadable and unintelligible to his foes without knowledge of the shift value. Cryptography, although a beautiful art of secret messages, is more commonly and actively used as a strategic tool in warfare between men in the past. At the beginning of the 20th century, with the outbreak of World War I and World War II, there was a surge in the demand for cryptography experts which was well observed with the invention of Hebern Rotor Machine by Edward Hebern in 1917 and shortly thereafter, the Enigma Machine by Arthur Scherbius in 1918 [41]. The Enigma Machine was regularly used by the Germans for military communication purposes. To secure victory during World War II,

codebreaking played a pivotal role. The British at Bletchley Park were successful in cracking the Enigma Machine when they constructed the first electronic computers, named Colossus.

In the 1970s, researchers at International Business Machines Corporation (IBM) created a block cipher called Lucifer which went on to become what is now known as the Data Encryption Standard (DES) [42]. DES was a significant milestone as it combined transposition and substitution techniques into a systematic algorithm and became the first cryptosystem to be certified by the National Bureau of Standards (now known as the National Institute of Standards and Technology (NIST)). However, in years to come, with advancements in computational power and cryptanalysis techniques, the system became vulnerable and was broken by exhaustive search attack due to its short key length.

A year after the inception of DES, the first public key cryptography, Diffie-Hellman key exchange method was introduced by Whitfield Diffie and Martin Hellman [40]. Not long after, inspired by Diffie and Hellman's concept of public key cryptography, the RSA algorithm, conveniently named by the researchers of Massachusetts Institute of Technology who invented it: Ron Rivest, Adi Shamir and Leonard Adleman, was created. The algorithm involves two keys, one private and one public. Unlike Diffie-Hellman, the basis of its security lies in the mathematical difficulty of factoring two large prime numbers instead of discrete logarithm problem.

Following the downfall of DES, cryptographic research shifted towards developing more robust systems. AES superseded its predecessor in 2001 when

it was selected by the NIST to replace DES. This symmetric-key algorithm, which is also a block cipher, operates with larger key lengths of 128, 192 and 256 bits whilst maintaining a fixed block size of 128 bits [43]. Its design offers enhanced security and efficiency, making it suitable for wide range of applications in today's digital world.

2.2 Development of DNA Cryptography and Related Works

DNA cryptography is an interdisciplinary field that merges the knowledge of molecular biology and cryptographic techniques. Unlike conventional cryptography which generally relies on numerical algorithms, DNA cryptography utilises DNA to encode and secure information.

The concept of DNA-based computation was pioneered by Leonard Adleman in the year of 1994, when he demonstrated that DNA molecules could be used to solve a searching problem, a directed Hamiltonian path problem known as the "Travelling Salesman Problem" with seven vertices which he assumed the molecules to be. In his study in 1998, he discovered that DNA possesses high storage and computational capability [44]. This has led the study to subsequently demonstrate the feasibility of using biological molecules for complex computational tasks, setting the stage for further exploration into DNA-based cryptographic systems. Following Adleman's pioneering work, early foundational exploration between the 1990s and the early 2000s focused on the use of DNA in codebreaking and solving complex problems. The first known application of DNA cryptology in codebreaking was performed by Boneh et al. in 1996 on DES, which was broken in just 4 months [45] and a NP-complete problem, the maximal clique problem was solved using the

approaches of DNA molecular theory by Ouyang et al. in 1997 [46]. These early studies have also provided insight into how knowledge of DNA could be used not only for computation but also for securing information.

During the early 2000s, researchers began exploring how DNA computing principles can be applied to encryption. In 2003, Chen pioneered DNA-based image encryption using one time pad (OTP) framework [47] and in 2004, Gehani et al. proposed a DNA-based one-time pad encryption technique which is based upon DNA substitution method and bitwise XOR operations, where the digital messages were translated into synthetic DNA sequences [48]. As one-time pads assure perfect secrecy, it is almost impossible for the adversaries to break the encrypted message. This proposal has shown the potential of DNA as a medium for secure communication.

As interest in DNA cryptography grew, researchers have expanded their focus into symmetric key systems. In 2006, Amin et al's symmetric key-based DNA cryptography derived a single key for both sender and receiver is obtained from a genetic database, which integrated publicly available biological data into cryptographic process [49]. Shortly thereafter, in 2007, Lu et al. proposed the DNA Symmetric Key Cryptosystem (DNASC), which has proven its resistant to highly efficient quantum computers due to the massive parallelism and information storage of DNA molecules [50]. Apart from symmetric key systems, DNA cryptography has also been integrated into asymmetric key cryptosystem and signature schemes such as Cui et al in 2008 who developed a public key encryption method which involved processes like DNA synthesis, DNA encoding and PCR amplification [51]; and Lai et al. in 2010 with their

DNA-PKC system, which combined DNA-based techniques with traditional asymmetric cryptographic algorithms [52].

The subsequent decade had also witnessed advancements in this field as more and more DNA-based methods are designed as well as refinement for existing techniques. Research in this period also explored adaptations of classical ciphers into DNA cryptography, for instance, Sabry et al. proposed a playfair cipher using DNA and amino acids in 2010 [53]. Data hiding techniques have also been enhanced by DNA properties when in 2010, Shiu et al. proposed three separate methods which are: the Insertion Method, the Complementary Pair Method and the Substitution Method [54]. In this paper, all three methods utilise a reference DNA which only the sender and receiver know from public DNA databases such as EBI database. The authors have also provided security analysis on the methods which indicated better performance compared with other competing methods. In 2012, a DNA fragment assembly-based cryptography was introduced by Zhang et al. which involves breaking a long chain of DNA encoded message into small DNA fragments and forwarding them to the receiver to be reassembled to uncover the original message [55]. Other research such as the DNA cipher based on DNA indexing by Tornea et al. in 2013 [56] and a DNA scheme with dynamic sequence table by Hossain et al. in 2016 [57] are notable.

Recent innovations include Karimi et al's DNA based algorithm which involves random number of rounds with varying key size depending on user's password lengths [58] and Kolte et al's index-based symmetric DNA encryption schemes that employed DNA sequence from NCBI database which is used as One Time Pad (OTP) symmetric key in 2017 [59]. In July 2018, Zhang et al.

introduced an image encryption scheme which is a combination of a Feistel network and dynamic DNA encoding, using GenBank sequences as keys [60]. A cryptographic scheme involving DNA and RNA processes is proposed by Nafea et al. in November 2018 [61]. Their OTP keys are generated by transcribing ssDNA pads into RNA and translating them into amino acid sequences, which were then converted into binary form for XOR encryption. In the same year, Kumar et al. has refined the DNA-based playfair cipher which was proposed in 2010 by Sabry et al [53].

In 2019, a level-based DNA security scheme which relies on DNA triplet codons for substitution was proposed by Patnala et al. [62]. This method uses a randomly arranged lookup table of codon-to-value mappings across 3 rounds where the plaintext is translated and grouped into codon triplets, substituted via a lookup table and re-encoded as DNA bases to produce the ciphertext. There are also other notable genetic algorithms proposed with implementations of DNA, RNA and amino acids like the RNA implementations on text encryption by Rashid in 2021 [63].

After more than two decades of research, the body of work in DNA cryptography from early demonstrations of molecular parallelism to modern hybrid symmetric and asymmetric encryption schemes has established the feasibility of using biological principles for cryptographic applications. Researchers have shown that DNA's massive storage density and parallel processing can achieve efficient key expansion and resistance to quantum attacks.

2.3 Development of Quasigroup-based Cryptography and Related Works

In the late eighteenth century, a new theory was proposed by Euler [64] in order to explore the idea of Latin squares. Cayley, famous for his work in the domain of group multiplication tables, proved that quasigroup tables could be represented as bordered Latin square. It was in the year 1935 that the term 'quasigroup' was first proposed by Moufang [65]. The general nature of quasigroup allows for their application in fields like coding theory, cryptography and telecommunications [66].

The widespread of cryptographic interest in quasigroups began much further before the 1990s. In fact, as Keedwell [67] recount, the very first recorded application was in Schauffler's 1948 doctoral thesis [68], where he showed that finding a suitable Latin square, which is essentially a quasigroup, is the key to breaking the Vigenère cipher. The essential idea behind quasigroup-based cryptography is that the nonassociative nature of quasigroup provides a foundation for designing cryptographic algorithms that are hard to break. The conceptual roots of quasigroup-based cryptography can be traced back to the study of Latin squares. Keedwell [67] briefly discussed the potential applications of Latin squares (which are basically quasigroups) in cryptography, more specifically in error detecting and correcting codes. From 1995 to 1996, Koscielny's work marked the initial exploration of quasigroup properties for stream and block ciphers [69]. These schemes have demonstrated that quasigroup tables could be used to construct secure and efficient encryption schemes In 1997, the work of researchers Gligoroski et al. further the momentum by focusing their research on quasigroup transformations and

demonstrating that such methods could effectively thwart brute force and statistical attacks, even when both plaintext and ciphertext were available to an adversary [70]. Ritter also contributed to the field by examining the practical uses of quasigroups in encryption in 1998 [71]. He emphasised that quasigroups possess potential in environments where lightweight computations were essential. Following the works of C. Koscielny in 1996, Ochodkova et al. introduced yet another stream cipher based upon the properties of quasigroup to encode file system [72]. In their findings, they believe that due to the simplicity of the quasigroup operations, it can be easily implemented as well as providing efficiency during the encryption and decryption procedure.

From 1999 onward, Markovski and his colleague introduced quasigroup string transformation in a series of four-part research papers from 1999 to 2007 [26], [73], [74], [75]. Their research highlighted the use of quasigroups for generating pseudorandom sequences and secure message encryption, even under known-plaintext scenarios. The resistance to such attacks was attributed to the unpredictable nature of quasigroup operations and the flexibility to vary transformation rules between sessions. Furthermore, in 2003, an All-Or-Nothing transformation (originally developed by Rivest), was combined with random quasigroups for better processing speed and security by Marnas et al. [76]. Their research has demonstrated the viability of hybrid systems which combine traditional and quasigroup-based techniques. Inspired by their research, researchers started to explore variations of quasigroup transformations. For instance, Gligoroski et al. [28] presented a novel asymmetric block cipher based on Multivariate Quadratic Quasigroups (MQQs), where its security lies primarily on the computational difficulty of

solving systems of multivariate quadratic equations defined over quasigroup operations. Based upon the concept of this system, they later proposed another digital signature scheme known as MQQ-SIG [29], which demonstrated high performance and resistance against chosen message attacks (CMA) based on their experimental evaluations. In 2010, Xu designed a stream cipher based on the concepts of quasigroup conjugates and has performed various cryptanalytic attacks to validate its security [77]. Additionally, Bakeva et al. came up with a parastrophic variation of the quasigroup string transformation in 2011 [78]. Parastrophes are alternate versions of the same quasigroup table, created by permuting the inputs and outputs. This approach has further enhanced and refined the security and flexibility of these systems.

Throughout the years, quasigroup concepts were applied to design more complex cryptographic primitives, for instance, the n-quasigroup stream ciphers by Petrescu in 2010 [79], which was then improved in 2012 by Chakrabarti et al. [80] to enhance both security and performance. Markovski's work in 2015 provided an in-depth exploration of cryptographic primitives based on quasigroups, which covered a range of applications, specifically in block ciphers, stream ciphers, digital signatures, encryption schemes and hash functions [81]. It should also be mentioned that there is an existing quasigroup based encryption scheme with implementation of biological process and protein codon codes known as sEncrypt algorithm which was proposed in 2013 [82]. In 2021, Tiwari et al. have proposed the block cipher INRU which features 64-bit block length and 128-bit key length and had shown strong resistance against a range of cryptanalytic attacks, including differential, linear and algebraic attacks

[27]. The lightweight block cipher was further improved in 2023 by Chauhan et al. which has shown less memory consumption [83].

2.4 Research Gaps

The field of cryptography is in a constant state of evolution. The emergence of DNA cryptography pioneered by Adleman's work on DNA computing has shown immense potential for information storage and parallelism. Prior works have all been involved in OTPs, symmetric and asymmetric systems, data hiding techniques and adaptation of classical domains into the DNA domains. The potential of DNA bases as the elements of the quasigroup itself to create a seamless encryption process in the biological domain remains unexplored. On the other hand, research work on quasigroup-based cryptography has shown inherent resistance to linear and differential cryptanalysis as well as exceptional speed and suitability for lightweight, resource-constrained environment. Existing quasigroup-based cryptographic algorithm typically relies on numerical operations defined over algebraic structures such as groups, rings, or finite fields. While these structures offer well-established mathematical properties, they also impose algebraic regularities such as associativity and predictable inverses which can be exploited by modern cryptanalysis. The literature reveals the central research gap between both fields, which is a lack of an integrated cryptosystem that fully harnesses the mathematical strength and efficiency of quasigroups directly within the DNA domain.

This research directly addresses the gap by proposing a novel symmetric quasigroup-based DNA cryptographic scheme, where both encryption and

decryption are performed using the same shared secret key, which is comprised of a randomly generated quasigroup table, a random parastrohe table, two leaders and a randomly chosen DNA encoding rule. We introduce a two-phase encryption process using a quasigroup and one of its parastrophes, with the DNA bases (A, C, G, T, U) as the fundamental element set of the quasigroup operations. The novelty of the proposed scheme becomes clear when contrasted with the established norms DNA-based and quasigroup-based cryptographic schemes. Existing quasigroup cryptosystems operate directly on binary bits, integers modulo n or bytes (e.g., INRU cipher, MQQ cryptosystem), the proposed method transposed this concept into a novel domain, applying quasigroup operations on DNA bases. This makes the scheme's design uniquely suited for scenarios where data is stored or transmitted as DNA sequences. Current DNA cryptographic schemes use the DNA bases A, T, C, G whilst the proposed scheme expands the element set to five by introducing Uracil (U) as a cryptographic element, thereby increasing complexity and blindsiding the attackers from the existence of U. The only other literature which has applied such 5 elements in its system [94] however, it is applied to the mathematical concept of dihedral group.

The elements of this quasigroup could, in principle, be any set of five distinct symbols. However, the choice to use DNA bases (including U) is not arbitrary and is fundamental to the thesis's contribution. The primary motivation for using DNA bases is to design a cryptosystem that is inherently compatible with DNA computing and data storage. While the current implementation is in silico (on traditional computers), its operation is defined natively in the "language" of these biological molecules. If the future of computing involves

massive parallelism using DNA strands, a cipher that operates on bits or numbers would require a translation layer. Our cipher, however, operates directly on the fundamental units (bases) if that future platform. The quasigroup transformations could, in theory, be executed as parallel, localised biochemical reactions on a DNA strand, fully exploiting the parallelism that DNA computing promises. A quasigroup using digits would lack the ability to directly be interpreted into biological molecules. It is also important to note that there is limited published work that measures how increasing quasigroup order or adding parastrophic phases affects efficiency (encryption and decryption time) and security (entropy) in quasigroup-based and DNA-based cryptographic schemes. This research fills the gap by providing a quantitative trade-off analysis connecting the quasigroup order and number of phases to performance and security metrics.

CHAPTER 3: METHODOLOGY

3.1 Basics of DNA Cryptography

DNA, more commonly known as Deoxyribose Nucleic Acid, is a complex molecule which serves as the fundamental storage space for genetic information found within all living organisms. Every living organism carries its own unique set of DNA which determines an organism's traits, from physical characteristics to cellular processes. DNA is composed of two long polynucleotide chains that coil around each other to form a double helix structure, which was first discovered in 1953 by scientists James Watson and Francis Crick [84]. Each chain contains a sequence of four different monomers of DNA, known as nucleotide. Each nucleotide comprises of three parts: a sugar molecule, a phosphate group and a nitrogenous base. In DNA, the nitrogenous base can be any one of four types: Adenine (A), Cytosine (C), Guanine (G) or Thymine (T). The structure of DNA is held together by hydrogen bonds between complementary pairs of nitrogenous bases which are always as follows where: A with T, C with G. This is commonly known as the "Watson-Crick complementary rules.

Through various combinations of the four bases, DNA is able to store vast and complex genetic data of any living organisms [85]. For the four DNA bases, there would be a total of 24 possible types of combination which are as follows [84]:

Table 3.1 Twenty-four possible types of combinations of 4 DNA bases

CTAG	CTGA	CATG	CAGT	CGTA	CGAT
TCAG	TCGA	TACG	TAGC	TGAC	TGCA
ATCG	ATGC	ACTG	ACGT	AGCT	AGTC
GTAC	GTCA	GATC	GACT	GCTA	GCAT

For combinations which fulfil the Watson-Crick complementary rules, it is mentioned in [84] that there are 8 types which are:

Table 3.2 Eight types of combinations which fulfil Watson-Crick rules

CTAG	CATG	GTAC	GATC
TCGA	TGCA	ACGT	AGCT

DNA's primary role is to store and transmit genetic information. Specific segments of DNA, called genes, encode instructions for synthesizing proteins which function in structuring tissues, regulating bodily functions and catalysing biochemical reactions. The sequence of bases (A, C, T, G) acts as a code, with triplet of bases (codons) specifying individual amino acids, the building blocks of proteins.

Other than DNA, there exist another acid, ribonucleic acid known as RNA, whereby they are structurally similar except T is substituted with a base known as Uracil (U). RNA acts as an intermediary between DNA and protein, for instance, the genetic data of DNA is moved and translated to protein through messenger RNA (mRNA) [85].

3.2 DNA Encoding and Decoding Rules

In DNA cryptography, DNA bases are used as a medium for information exchange. Data can be encoded by mapping binary digits to corresponding nucleotides and to retrieve the original information, nucleotides are mapped to corresponding binary digits. These processes are known as DNA encoding and decoding rules. There are 8 possible DNA encoding and decoding rules which are shown in the table below [84].

Table 3.3 Eight rules for DNA encoding and decoding

	Rule 1	Rule 2	Rule 3	Rule 4	Rule 5	Rule 6	Rule 7	Rule 8
00	C	C	G	G	T	T	A	A
01	T	A	T	A	C	G	C	G
10	A	T	A	T	G	C	G	C
11	G	G	C	C	A	A	T	T

The encoding process typically follows these steps. Binary data (0s and 1s) is converted into DNA sequences using predefined mapping scheme as shown above. For example, based on Rule 7, 00 is mapped to A, 01 is mapped to C, 10 is mapped to G and 11 is mapped to T. This mapping ensures that every pair of binary digits corresponds to a specific nucleotide base, thus allowing digital data to be represented in the form of DNA sequence. Textual data can also be encoded into DNA by first converting characters into their binary representation and then applying the binary-to-DNA mapping. For example, the letter “A” in ASCII is 01000001 in binary form and by using Rule 7, “A” can be encoded as “CAAC”.

Decoding in DNA cryptography involves reversing the encoding process to retrieve the original message from the DNA sequence. The process can be done by converting DNA sequence back into binary data using the same mapping scheme used during the encoding process. For example, A → 00, C → 01, G → 10 and T → 11. The binary data is then converted back into textual form using ASCII.

3.3 Basics of Quasigroups

3.3.1 Binary Operation, Groupoids and Quasigroups

Definition 3.1. A binary operation on a nonempty set G is a function

$$\alpha: G \times G \to G.$$

That is, given any two elements a and b in G, the operation α assigns an element $\alpha(a, b)$ in G. When discussing general algebraic structures, it is often convenient to use a product notation $*$ such as writing $a * b$ in place of $\alpha(a, b)$.

Definition 3.2. A groupoid is a nonempty set G with binary operation $*$, which is denoted as $(G,*)$. The order of $(G,*)$ is the cardinality $|G|$ which means the number of elements in G. A groupoid is also said to be finite if $|G|$ is finite.

Definition 3.3. A quasigroup $(Q,*)$ is a groupoid which satisfies the following law:

For every $a, b \in Q$, there exist unique $x, y \in Q$ such that

$$a * x = b \text{ and } y * a = b.$$

For a finite set Q, the structure of a quasigroup $(Q,*)$ can be represented using a multiplication table. From Definition 3.3, due to the unique solvability

property, each element will appear exactly once in each row and each column of the multiplication table of $(Q,*)$. To construct a multiplication table of quasigroup, let Q be a finite set with n elements $\{a_1, a_2, \dots, a_n\}$. An $n \times n$ table is formed where the entry a_{ij} located in the i-th row and j-th column is the product of the element a_i and a_j. Each cell in the table is filled with the elements of the quasigroup without repetition in each row and column. Note that a given quasigroup can produce more than one multiplication table depending on the order of the elements formed at the border of the table.

Table 3.4 Multiplication table of a quasigroup

$*$	a_1	a_2	$\cdots$	a_n
a_1	a_{11}	a_{12}	$\cdots$	a_{1n}
a_2	a_{21}	a_{22}	$\cdots$	a_{2n}
$\vdots$	$\vdots$	$\vdots$	$\ddots$	$\vdots$
a_n	a_{n1}	a_{n2}	$\cdots$	a_{nn}

The following is an example of a multiplication table of a quasigroup $(Q,*)$ of order 4:

Table 3.5 Example of multiplication table of a quasigroup of order 4

$*$	1	2	3	4
1	2	3	1	4
2	4	1	3	2
3	3	4	2	1
4	1	2	4	3

Quasigroups are defined by a binary operation that ensures the Latin square property which states that, for each a and b in Q, there exist unique

elements x and y in Q such that $a * x = b$ and $y * a = b$ hold. This property is equivalent to the operation table of $*$ forming a Latin Square, where each element of Q appears exactly once in every row and column. The absence of repeated elements in rows or columns guarantees that solutions to the equations $a * x = b$ and $y * a = b$ are always unambiguous. Thus, $(Q,*)$ has the property of unique divisibility.

Definition 3.4. We define the left division operation, denoted $a\backslash b$ as:

$$a\backslash b = x \text{ if and only if } a * x = b$$

Similarly, we define the right division operation, denoted by b/a, as:

$$b/a = y \text{ if and only if } y * a = b$$

Due to the divisibility laws, quasigroup ensures that both the left and right cancellation laws hold. The cancellations laws are as follows:

(i) For $a, x, y \in Q$, $a * x = a * y$ implies that $x = y$ (left cancellation)

(ii) For a, x, $y \in Q$, $x * a = y * a$ implies that $x = y$ (right cancellation)

In algebra, a group is a mathematical structure which consists of a set paired with a binary operation that follows specific constraints: the operation must be associative, there must be an identity element, and every element has an inverse.

Definition 3.5. A group is an algebraic structure consisting of a set G together with a binary operation $*$ that satisfies the following four axioms:

(i) For all $a, b \in G$, the result of the operation $a * b$ is also in G.

(ii) For all $a, b, c \in G$, $(a * b) * c = a * (b * c)$.

(iii) There exists an element $e \in G$ (identity) such that for every element $a \in G$, $a * e = e * a = a$.

(iv) For every element $a \in G$, there exists an inverse element $a^{-1} \in G$ such that $a * a^{-1} = a^{-1} * a = e$.

From Definition 3.5, it follows that every group is a quasigroup. A quasigroup is a mathematical structure similar to a group but with less restrictive properties as they are not required to satisfy properties such as associativity or commutativity or having an identity element and this also means that its elements need not have inverses.

3.3.2 Parastrophes of Quasigroups

Definition 3.6. For each quasigroup operation '$*$', we can associate a new operation '$\circ$' on Q defined by:

$$x \circ y = z \text{ if and only if } x * z = y.$$

Definition 3.7. Each quasigroup $Q = (Q,*)$ forms five new quasigroup $Q_i = (Q,*_i)$ with operations $*_i$ defined as follows:

$$x *_1 y = z \leftrightarrow x * z = y \text{ (right division)}$$

$$x *_2 y = z \leftrightarrow z * y = x \text{ (left division)}$$

$$x *_3 y = z \leftrightarrow z * x = y \text{ (opposite multiplication)}$$

$$x *_4 y = z \leftrightarrow y * z = x \text{ (opposite right division)}$$

$$x *_5 y = z \leftrightarrow y * x = z \text{ (opposite left division)}$$

Quasigroups which are defined as such are known as parastrophes or conjugates of Q. It is worth noting that the operation $\circ$ in Definition 3.6 is actually the same operation $*_1$ in Definition 3.7. We single out this operation as it will be very useful in the decryption process in the next section.

The significance of parastrophes lies in their ability to provide alternative perspectives on the structure of quasigroup. In the study of quasigroups, one encounters not just the original binary operation but also a family of related operations known as parastrophes. Parastrophes (or conjugates) of a quasigroup are essentially just variations of the original quasigroup obtained by permuting the order of operations. Recall that a quasigroup is defined by the property that for any elements a and b in set Q, the equation $a * b = c$ has a unique solution for the unknown when any two of the three elements are fixed. The unique solvability property implies that one can "rearrange" the equation to define other operations. It is a well-established fact in quasigroup theory [86] that from any given quasigroup, one can define 6 conjugate quasigroups which are not necessarily distinct (including the original quasigroup).

It is also proven in [86] that the number of distinct parastrophes is always a divisor of 6, which are 1, 2, 3 or 6 and that for any $n \geq 4$, there exists a quasigroup of order n with $m = \{1, 2, 3, 6\}$ distinct conjugates. The number of distinct parastrophes does not depend on the cardinality of the quasigroup Q (the number of elements in Q) but rather on the structural properties.

In quasigroup theory, the parastrophes can either be pairwise distinct or pairwise equal. When the parastrophes are pairwise distinct, it means that each

of the five parastrophes exhibits different quasigroup structure. In other words, no two parastrophes are the same. However, when the parastrophes are described as pairwise equal, it means that some or all of the parastrophes coincide, resulting in quasigroup with identical structures. This occurs when the original quasigroup exhibits certain symmetries or special properties. Pairwise equality often occurs in specific types of quasigroups, such as commutative quasigroups or idempotent quasigroups. For instance, in a commutative quasigroup, the order of the elements does not affect the outcome, so the operations derived from switching the positions of the operands may end up being identical.

Table 3.6 Number of quasigroups of order $n \leq 11$

n	Q_n
1	1
2	2
3	12
4	576
5	161280
6	812851200
7	61479419904000
8	10877603245908295680
9	552475149615689284253122560
10	998243765821303987172506475692032000
11	776966836171770144107444346734230682311065600000

3.4 Encryption and Decryption Function

Let A be a finite set with elements $\{a_1, a_2, \dots, a_n\}$ and we construct nonempty finite strings $x_1 x_2 \cdots x_m$ of length m, from elements in A (i.e. $x_i \in A$ for all $i = 1, 2, \dots, m$). Let L be a leader chosen from the set A. Define a

quasigroup operation $*$ on the set A and the corresponding $\circ$ operation as mentioned in Definition 3.6. For the chosen $L \in A$, we define two functions as follows:

(i) Encryption function, $E(x_1x_2 \cdots x_m) = (y_1y_2 \cdots y_m)$, where $y_1 = L * x_1$ and $y_i = y_{i-1} * x_i$ for $i = 2, 3, \ldots, m$.

(ii) Decryption function, $D(y_1y_2 \cdots y_m) = (x_1x_2 \cdots x_m)$, where $x_1 = L \circ y_1$ and $x_i = y_{i-1} \circ y_i$.for $i = 2, 3, \ldots, m$.

3.4.1 Example of Application of Encryption and Decryption Functions

The applied encryption and decryption functions are demonstrated as follows:

We let Q be a quasigroup with operation $*$ and set of elements $\{0, 1, 2, 3, 4\}$. A quasigroup table is formed and shown as follows:

Table 3.7 Quasigroup table of order 5 with elements $\{0, 1, 2, 3, 4\}$

*	0	1	2	3	4
0	4	1	0	3	2
1	2	3	1	4	0
2	1	2	0	3	4
3	3	4	2	0	1
4	4	2	0	1	3

The quasigroup table acts as the key for encryption and decryption. At the start of encryption, a Leader, L which can be any element of the quasigroup table is chosen. To start the encryption process, the plaintext message is chosen to be $M = (x_1x_2 \cdots x_m) = (30412431)$. Then the encryption method is carried

out using: $E(x_1x_2\cdots x_m) = (y_1y_2\cdots y_m)$, where $y_1 = L * x_1$ and $y_i = y_{i-1} * x_i$. According to the plaintext message M, $x_1 = 3$, $x_2 = 0$, $x_3 = 4$ and so on. For this example, the Leader, $L = 0$ is chosen and based on the encryption formula and quasigroup table given above, the following result is obtained:

$$\begin{array}{cccc} y_1 = L * x_1 & y_2 = y_1 * x_2 & y_3 = y_2 * x_3 & y_4 = y_3 * x_4 \\ = 0 * 3 & = 3 * 0 & = 3 * 4 & = 1 * 1 \\ = 3 & = 3 & = 1 & = 3 \\ y_5 = y_4 * x_5 & y_6 = y_5 * x_6 & y_7 = y_6 * x_7 & y_8 = y_7 * x_8 \\ = 3 * 2 & = 2 * 4 & = 4 * 3 & = 1 * 1 \\ = 2 & = 4 & = 1 & = 3 \end{array}$$

Thus, the encrypted message, $E(M)$ = (33132413) is obtained.

Since there are various choices for choosing a leader, the encryption is made strong by choosing different leaders for each encryption. The quasigroup table can also be changed by permuting its rows and columns to produce different versions of quasigroup and this in turn increases the complexity of encryption scheme.

3.5 Proposed Scheme

The process of designing the system is mainly focused on combining the properties of DNA sequences with quasigroup operations. The algorithm has two phases of encryption as well as two phases of decryption, with the 1st phase involving quasigroup and the second phase involving its parastrophes. The first step of the process is to decide the order and elements of the quasigroup to be used. For the proposed model, a quasigroup, Q of order 5 which comprises of DNA bases $\{U, A, C, G, T\}$ as elements is selected. The number of Latin squares

of order n is known for small n. According to Table 3.7, for $n = 5$, the number of distinct Latin squares is 161280.

In this algorithm, base Uracil (U) which appears in Ribonucleic Acid (RNA) is used and will be considered as one of the elements to be used in the quasigroup table. Element 'U' enhances security by being absent in the plaintext DNA sequence but present in the ciphertext through quasigroup operations. Since 'U' does not appear in the original DNA bases (A, T, C, G), attackers may overlook its existence in the quasigroup and parastrophe table. By adding ‘U’ to the set of DNA bases used in the encryption process, the number of possible outputs for each operation can be exponentially increased, thus effectively increasing the complexity of the algorithm and resistance against statistical attacks. A larger key space also enhances security against brute force attacks by making them less computationally feasible. While 'U' naturally appears in RNA instead of DNA, its role in this cryptographic scheme is to increase security and differentiate the method from traditional DNA cryptography, which only uses four bases.

Our scheme which comprises of quasigroups and parastrophe transformation as well as DNA encoding with the introduction of element ‘U’ offers a novel approach not commonly explored in traditional cryptographic algorithms. Encryption begins by converting plaintext into DNA using a random encoding rule. In Phase I, a random leader is chosen from the set of elements $\{U, A, C, G, T\}$ and a random quasigroup table is generated. The plaintext was then encrypted using quasigroup operation $x * y = z$ to produce the first ciphertext. In Phase II, a new random leader and a random parastrophe table of the quasigroup used in Phase I are used to further encrypt to form the final

ciphertext. The two leaders, DNA encoding rule, quasigroup table and parastrophe table will act as the key and are transmitted through a secure channel to the receiver. The decryption process involves reversing the encryption steps using the inverse operation. The receiver retrieves the original plaintext by applying the decryption operation to the ciphertext with the key received.

3.5.1 Encryption Scheme

The encryption scheme is shown as follows:

1. A plaintext message M is chosen and is converted into hexadecimal form and subsequently binary form by referring to the ASCII table.
2. The message is then encoded into DNA bases to form a string of DNA sequence $(x_1 x_2 \cdots x_m)$ of length m using a randomly chosen encoding rule.
3. Once the DNA form of the message is obtained, the system enters into Phase I of encryption, where a Leader, L_1 is chosen randomly from a set of elements $\{A, C, G, T, U\}$. The quasigroup table for encryption is generated randomly using quasigroup operation $x * y = z$.
4. The function used for encryption Phase I is defined as $E_1(x_1 x_2 \cdots x_m) = (y_1 y_2 \cdots y_m)$, where $y_1 = L_1 * x_1$ and $y_i = y_{i-1} * x_i$.
5. The message is encrypted using the function E_1 and first ciphertext, $C_1 = (y_1 y_2 \cdots y_m)$, which is in DNA form is obtained.
6. In Phase II of encryption, a leader L_2 and a parastrophe operation $*_k$ are randomly chosen, and the corresponding parastrophe table is generated.

7. Encryption function for Phase II is defined as $E_2(C_1) = E_2(y_1 y_2 \cdots y_m) = (z_1 z_2 \cdots z_m)$, where $z_1 = L_2 *_k y_1$ and $z_i = z_{i-1} *_k y_i$.
8. The final ciphertext, $C_2 = (z_1 z_2 \ldots z_m)$ is obtained and sent to the receiver.

3.5.2 Decryption Scheme

The decryption scheme is shown as follows:

1. The ciphertext C_2 is received in DNA form.
2. In Phase I of decryption, a parastrophe table for $\circ_k$ is generated using the parastrophe table of $*_k$ such that $x \circ_k y = z$ if and only if $x *_k z = y$.
3. The function for decryption process is defined as $D_1(z_1 z_2 \cdots z_m) = (y_1 y_2 \cdots y_m)$, where $y_1 = L_2 \circ_k z_1$ and $y_i = z_{i-1} \circ_k z_i$.
4. The ciphertext, $C_1 = (y_1 y_2 \cdots y_m)$ is obtained by decrypting with the leader L_2 received and the quasigroup generated using decryption operation $\circ_k$.
5. For Phase II of decryption, the quasigroup table for $\circ$ is generated using the quasigroup table of $*$ in Phase I of encryption.
6. The decryption function is defined as $D_2(y_1 y_2 \cdots y_m) = (x_1 x_2 \cdots x_m)$, where $x_1 = L_1 \circ y_1$ and $x_i = y_{i-1} \circ y_i$.
7. The plaintext $(x_1 x_2 \cdots x_m)$ is obtained by decrypting the leader L_1 received and the quasigroup table for $\circ$. The decrypted message is converted into hexadecimal form and finally into the original message M by using corresponding DNA decoding rule and ASCII table.

CHAPTER 4: IMPLEMENTATION

4.1 Encryption Process

The implementation of the algorithm is demonstrated below:

Let the plaintext message, $M = \text{NOTTINGHAM}$.

Convert $M = \text{NOTTINGHAM}$ to Hexadecimal:

$$4E\ 4F\ 54\ 54\ 49\ 4E\ 47\ 48\ 41\ 4D$$

Convert $M = \text{NOTTINGHAM}$ to Binary:

$$0100\ 1110\ 0100\ 1111\ 0101\ 0100\ 0101\ 0100\ 0100\ 1001$$

$$0100\ 1110\ 01000111\ 0100\ 1000\ 0100\ 0001\ 0100\ 1101$$

We will be using DNA encoding Rule 7 in this demonstration.

$\text{N}(4E) \rightarrow 0100\ 1101 \rightarrow TCGA$ $\quad$ $\text{O}(4F) \rightarrow 0100\ 1111 \rightarrow TCGG$

$\text{T}(54) \rightarrow 0101\ 0100 \rightarrow TTTC$ $\quad$ $\text{T}(54) \rightarrow 0101\ 0100 \rightarrow TTTC$

$\text{I}(49) \rightarrow 0100\ 1001 \rightarrow TCAT$ $\quad$ $\text{N}(4E) \rightarrow 0100\ 1110 \rightarrow TCGA$

$\text{G}(47) \rightarrow 0100\ 0111 \rightarrow TCTG$ $\quad$ $\text{H}(48) \rightarrow 0100\ 1000 \rightarrow TCAC$

$\text{A}(41) \rightarrow 0100\ 0001 \rightarrow TCCT$ $\quad$ $\text{M}(4D) \rightarrow 0100\ 1101 \rightarrow TCGT$

The encoded plaintext M in DNA form is a string of length $m = 40$,

(*TCGA TCGG TTTC TTTC TCAT TCGA TCTG TCAC TCCT TCGT*) (1)

4.1.1 Phase I: Quasigroup

We choose a Leader, $L_1 = G$

Our quasigroup table is formed and shown as follows:

Table 4.1 Quasigroup table for Phase I of encryption

$*$	U	A	C	G	T
U	A	T	C	G	U
A	G	A	U	T	C
C	C	G	T	U	A
G	U	C	G	A	T
T	T	U	A	C	G

The function used for encryption is defined as:

$$E_1(x_1x_2\cdots x_m) = (y_1y_2\cdots y_m),$$

where n is the length of the plaintext M in DNA form, $y_1 = L_1 * x_1$ and $y_i = y_{i-1} * x_i$.

The encryption process for Phase I is carried out with leader $L_1 = G$ and $x_1 = T, x_2 = C, x_3 = G, \ldots$ from (1).

$$\begin{aligned}
y_1 &= L_1 * x_1 & y_2 &= y_1 * x_2 & y_3 &= y_2 * x_3 & y_4 &= y_3 * x_4\\
&= G * T & &= T * C & &= A * G & &= T * A\\
&= T & &= A & &= T & &= U\\
y_5 &= y_4 * x_5 & y_6 &= y_5 * x_6 & y_7 &= y_6 * x_7 & y_8 &= y_7 * x_8\\
&= U * T & &= U * C & &= C * G & &= U * G\\
&= U & &= C & &= U & &= G\\
y_9 &= y_8 * x_9 & y_{10} &= y_9 * x_{10} & y_{11} &= y_{10} * x_{11} & y_{12} &= y_{11} * x_{12}\\
&= G * T & &= T * T & &= G * T & &= T * C\\
&= T & &= G & &= T & &= A\\
y_{13} &= y_{12} * x_{13} & y_{14} &= y_{13} * x_{14} & y_{15} &= y_{14} * x_{15} & y_{16} &= y_{15} * x_{16}\\
&= A * T & &= C * T & &= A * T & &= C * C\\
&= C & &= A & &= C & &= T
\end{aligned}$$

$$
\begin{array}{llllllll}
y_{17} & = y_{16} * x_{17} & y_{18} & = y_{17} * x_{18} & y_{19} & = y_{18} * x_{19} & y_{20} & = y_{19} * x_{20} \\
 & = T * T & & = G * C & & = G * A & & = C * T \\
 & = G & & = G & & = C & & = A \\
y_{21} & = y_{20} * x_{21} & y_{22} & = y_{21} * x_{22} & y_{23} & = y_{22} * x_{23} & y_{24} & = y_{23} * x_{24} \\
 & = A * T & & = C * C & & = T * G & & = C * A \\
 & = C & & = T & & = C & & = G \\
y_{25} & = y_{24} * x_{25} & y_{26} & = y_{25} * x_{26} & y_{27} & = y_{26} * x_{27} & y_{28} & = y_{27} * x_{28} \\
 & = G * T & & = T * C & & = A * T & & = C * G \\
 & = T & & = A & & = C & & = U \\
y_{29} & = y_{28} * x_{29} & y_{30} & = y_{29} * x_{30} & y_{31} & = y_{30} * x_{31} & y_{32} & = y_{31} * x_{32} \\
 & = U * T & & = U * C & & = C * A & & = G * C \\
 & = U & & = C & & = G & & = G \\
y_{33} & = y_{32} * x_{33} & y_{34} & = y_{33} * x_{34} & y_{35} & = y_{34} * x_{35} & y_{36} & = y_{35} * x_{36} \\
 & = G * T & & = T * C & & = A * C & & = U * T \\
 & = T & & = A & & = U & & = U \\
y_{37} & = y_{36} * x_{37} & y_{38} & = y_{37} * x_{38} & y_{39} & = y_{38} * x_{39} & y_{40} & = y_{39} * x_{40} \\
 & = U * T & & = U * C & & = C * G & & = U * T \\
 & = U & & = C & & = U & & = U
\end{array}
$$

The encrypted message $C_1 = E_1(M)$ for the Phase I is

(*TATU UCUG TGTA CACT GGCA CTCG TACU UCGG TAUU UCUU*) (2)

4.1.2 Phase II: Parastrophe

There exist 5 distinct parastrophes (conjugates) for each quasigroup. The conjugates can form a quasigroup table by using their corresponding operations: The conjugates of quasigroup table are shown as follows:

$x *_1 y = z \Leftrightarrow x * z = y$

$*_1$	U	A	C	G	T
U	T	U	C	G	A
A	C	A	T	U	G
C	G	T	U	A	C
G	U	G	A	C	T
T	A	C	G	T	U

$x *_2 y = z \Leftrightarrow z * y = x$

$*_2$	U	A	C	G	T
U	G	T	A	C	U
A	U	A	T	G	C
C	C	G	U	T	A
G	A	C	G	U	T
T	T	U	C	A	G

$x *_3 y = z \Leftrightarrow z * x = y$

$*_3$	U	A	C	G	T
U	T	C	G	U	A
A	U	A	T	G	C
C	C	T	U	A	G
G	G	U	A	C	T
T	A	G	C	T	U

$x *_4 y = z \Leftrightarrow y * z = x$

$*_4$	U	A	C	G	T
U	G	U	C	A	T
A	T	A	G	C	U
C	A	T	U	G	C
G	C	G	T	U	A
T	U	C	A	T	G

$x *_5 y = z \Leftrightarrow y * x = z$

$*_5$	U	A	C	G	T
U	A	G	C	U	T
A	T	A	G	C	U
C	C	U	T	G	A
G	G	T	U	A	C
T	U	C	A	T	G

For the second phase of the encryption process, the algorithm is similar to the first phase. The only distinct part of the process would be the use of quasigroup table which is replaced with a random conjugate of quasigroup table.

To start the second phase, we choose a Leader, $L_2 = A$.

The random conjugate of quasigroup table for this phase is chosen to be:

Table 4.2 Conjugate of quasigroup table for Phase II of encryption

$*_4$	U	A	C	G	T
U	G	U	C	A	T
A	T	A	G	C	U
C	A	T	U	G	C
G	C	G	T	U	A
T	U	C	A	T	G

The function used for encryption is defined as:

$$E_2(y_1 y_2 \cdots y_m) = (z_1 z_2 \cdots z_m),$$

where $z_1 = L_2 *_4 y_1$ and $z_i = z_{i-1} *_4 y_i$.

The encryption process for Phase II is carried out with leader $L_2 = A$ and $y_1 = T$, $y_2 = A$, $y_3 = T$, … from (2).

$$\begin{aligned}
z_1 &= L_2 *_4 y_1 & z_2 &= z_1 *_4 y_2 & z_3 &= z_2 *_4 y_3 & z_4 &= z_3 *_4 y_4 \\
&= A *_4 T & &= U *_4 A & &= U *_4 T & &= T *_4 U \\
&= U & &= U & &= T & &= U \\
z_5 &= z_4 *_4 y_5 & z_6 &= z_5 *_4 y_6 & z_7 &= z_6 *_4 y_7 & z_8 &= z_7 *_4 y_8 \\
&= U *_4 U & &= G *_4 C & &= T *_4 U & &= U *_4 G \\
&= G & &= T & &= U & &= A \\
z_9 &= z_8 *_4 y_9 & z_{10} &= z_9 *_4 y_{10} & z_{11} &= z_{10} *_4 y_{11} & z_{12} &= z_{11} *_4 y_{12} \\
&= A *_4 T & &= U *_4 G & &= A *_4 T & &= U *_4 A \\
&= U & &= A & &= U & &= U \\
z_{13} &= z_{12} *_4 y_{13} & z_{14} &= z_{13} *_4 y_{14} & z_{15} &= z_{14} *_4 y_{15} & z_{16} &= z_{15} *_4 y_{16} \\
&= U *_4 C & &= C *_4 A & &= T *_4 C & &= A *_4 T \\
&= C & &= T & &= A & &= U \\
z_{17} &= z_{16} *_4 y_{17} & z_{18} &= z_{17} *_4 y_{18} & z_{19} &= z_{18} *_4 y_{19} & z_{20} &= z_{19} *_4 y_{20} \\
&= U *_4 G & &= A *_4 G & &= C *_4 C & &= U *_4 A \\
&= A & &= C & &= U & &= U
\end{aligned}$$

$$
\begin{aligned}
z_{21} &= z_{20} *_4 y_{21} & z_{22} &= z_{21} *_4 y_{22} & z_{23} &= z_{22} *_4 y_{23} & z_{24} &= z_{23} *_4 y_{24} \\
&= U *_4 C & &= C *_4 T & &= C *_4 C & &= U *_4 G \\
&= C & &= C & &= U & &= A \\
z_{25} &= z_{24} *_4 y_{25} & z_{26} &= z_{25} *_4 y_{26} & z_{27} &= z_{26} *_4 y_{27} & z_{28} &= z_{27} *_4 y_{28} \\
&= A *_4 T & &= U *_4 A & &= U *_4 C & &= C *_4 U \\
&= U & &= U & &= C & &= A \\
z_{29} &= z_{28} *_4 y_{29} & z_{30} &= z_{29} *_4 y_{30} & z_{31} &= z_{30} *_4 y_{31} & z_{32} &= z_{31} *_4 y_{32} \\
&= A *_4 U & &= T *_4 C & &= A *_4 G & &= C *_4 G \\
&= T & &= A & &= C & &= G \\
z_{33} &= z_{32} *_4 y_{33} & z_{34} &= z_{33} *_4 y_{34} & z_{35} &= z_{34} *_4 y_{35} & z_{36} &= z_{35} *_4 y_{36} \\
&= G *_4 T & &= A *_4 A & &= A *_4 U & &= T *_4 U \\
&= A & &= A & &= T & &= U \\
z_{37} &= z_{36} *_4 y_{37} & z_{38} &= z_{37} *_4 y_{38} & z_{39} &= z_{38} *_4 y_{39} & z_{40} &= z_{39} *_4 y_{40} \\
&= U *_4 U & &= G *_4 C & &= T *_4 U & &= U *_4 U \\
&= G & &= T & &= U & &= G
\end{aligned}
$$

Finally, we obtained our ciphertext to be:

$$E_2(C_1) = C_2 =$$

(*UUTU GTUA UAUU CTAU ACUU CCUA UUCA TACG AATU GTUG*).

4.2 Decryption Process

An encrypted message C_2 is received as shown below:

$$(UUTU\ GTUA\ UAUU\ CTAU\ ACUU\ CCUA\ UUCA\ TACG\ AATU\ GTUG) \quad (3)$$

The quasigroup table for decryption Phase I is shown as follows:

Table 4.3 Quasigroup table for Phase I of decryption

$\circ_4$	U	A	C	G	T
U	A	G	C	U	T
A	T	A	G	C	U
C	C	U	T	G	A
G	G	T	U	A	C
T	U	C	A	T	G

Phase I of the decryption is carried out as follows:

$$D_1(z_1 z_2 \cdots z_m) = (y_1 y_2 \cdots y_m),$$

where $y_1 = L_2 \circ_4 z_1$ and $y_i = z_{i-1} \circ_4 z_i$.

The decryption process for Phase I is carried out with leader $L_2 = A$ and $z_1 = U, z_2 = U, z_3 = T, \ldots$ from (3).

$$\begin{aligned}
y_1 &= L_2 \circ_4 z_1 = A \circ_4 U = T & y_2 &= z_1 \circ_4 z_2 = U \circ_4 U = A \\
y_3 &= z_2 \circ_4 z_3 = U \circ_4 T = T & y_4 &= z_3 \circ_4 z_4 = T \circ_4 U = U \\
y_5 &= z_4 \circ_4 z_5 = U \circ_4 G = U & y_6 &= z_5 \circ_4 z_6 = G \circ_4 T = C \\
y_7 &= z_6 \circ_4 z_7 = T \circ_4 U = U & y_8 &= z_7 \circ_4 z_8 = U \circ_4 A = G \\
y_9 &= z_8 \circ_4 z_9 = A \circ_4 U = T & y_{10} &= z_9 \circ_4 z_{10} = U \circ_4 A = G \\
y_{11} &= z_{10} \circ_4 z_{11} = A \circ_4 U = T & y_{12} &= z_{11} \circ_4 z_{12} = U \circ_4 U = A \\
y_{13} &= z_{12} \circ_4 z_{13} = U \circ_4 C = C & y_{14} &= z_{13} \circ_4 z_{14} = C \circ_4 T = A \\
y_{15} &= z_{14} \circ_4 z_{15} = T \circ_4 A = C & y_{16} &= z_{15} \circ_4 z_{16} = A \circ_4 U = T \\
y_{17} &= z_{16} \circ_4 z_{17} = U \circ_4 A = G & y_{18} &= z_{17} \circ_4 z_{18} = A \circ_4 C = G \\
y_{19} &= z_{18} \circ_4 z_{19} = C \circ_4 U = C & y_{20} &= z_{19} \circ_4 z_{20} = U \circ_4 U = A
\end{aligned}$$

$$
\begin{array}{llllllll}
y_{21} & = z_{20} \circ_4 z_{21} & y_{22} & = z_{21} \circ_4 z_{22} & y_{23} & = z_{22} \circ_4 z_{23} & y_{24} & = z_{23} \circ_4 z_{24} \\
& = U \circ_4 C & & = C \circ_4 C & & = C \circ_4 U & & = U \circ_4 A \\
& = C & & = T & & = C & & = G \\
y_{25} & = z_{24} \circ_4 z_{25} & y_{26} & = z_{25} \circ_4 z_{26} & y_{27} & = z_{26} \circ_4 z_{27} & y_{28} & = z_{27} \circ_4 z_{28} \\
& = A \circ_4 U & & = U \circ_4 U & & = U \circ_4 C & & = C \circ_4 A \\
& = T & & = A & & = C & & = U \\
y_{29} & = z_{28} \circ_4 z_{29} & y_{30} & = z_{29} \circ_4 z_{30} & y_{31} & = z_{30} \circ_4 z_{31} & y_{32} & = z_{31} \circ_4 z_{32} \\
& = A \circ_4 T & & = T \circ_4 A & & = A \circ_4 C & & = C \circ_4 G \\
& = U & & = C & & = G & & = G \\
y_{33} & = z_{32} \circ_4 z_{33} & y_{34} & = z_{33} \circ_4 z_{34} & y_{35} & = z_{34} \circ_4 z_{35} & y_{36} & = z_{35} \circ_4 z_{36} \\
& = G \circ_4 A & & = A \circ_4 A & & = A \circ_4 T & & = T \circ_4 U \\
& = T & & = A & & = U & & = U \\
y_{37} & = z_{36} \circ_4 z_{37} & y_{38} & = z_{37} \circ_4 z_{38} & y_{39} & = z_{38} \circ_4 z_{39} & y_{40} & = z_{39} \circ_4 z_{40} \\
& = U \circ_4 G & & = G \circ_4 T & & = T \circ_4 U & & = U \circ_4 G \\
& = U & & = C & & = U & & = U
\end{array}
$$

The decrypted message $D_1(C_2)$for Phase I is

$$D_1(C_2) = C_1 =$$

$(TATU\ UCUG\ TGTA\ CACT\ GGCA\ CTCG\ TACU\ UCGG\ TAUU\ UCUU)$.

The quasigroup table used for Phase II of decryption is as follows:

Table 4.4 Quasigroup table for Phase II of decryption

$\circ$	U	A	C	G	T
U	T	U	C	G	A
A	C	A	T	U	G
C	G	T	U	A	C
G	U	G	A	C	T
T	A	C	G	T	U

The Phase II of the decryption process is carried out with

$$D_2(y_1 y_2 \cdots y_m) = (x_1 x_2 \cdots x_m),$$

where $x_1 = L_1 \circ y_1$ and $x_i = y_{i-1} \circ y_i$.

Recall the leader $L_1 = G$ and $y_1 = T$, $y_2 = A$, $y_3 = T$, … from (4).

Second phase of the decryption process is carried out as shown below:

$$
\begin{array}{llll}
x_1 = L \circ y_1 & x_2 = y_1 \circ y_2 & x_3 = y_2 \circ y_3 & x_4 = y_3 \circ y_4 \\
= G \circ T & = T \circ A & = A \circ T & = T \circ U \\
= T & = C & = G & = A \\
x_5 = y_4 \circ y_5 & x_6 = y_5 \circ y_6 & x_7 = y_6 \circ y_7 & x_8 = y_7 \circ y_8 \\
= U \circ U & = U \circ C & = C \circ U & = U \circ G \\
= T & = C & = G & = G \\
x_9 = y_8 \circ y_9 & x_{10} = y_9 \circ y_{10} & x_{11} = y_{10} \circ y_{11} & x_{12} = y_{11} \circ y_{12} \\
= G \circ T & = T \circ G & = G \circ T & = T \circ A \\
= T & = T & = T & = C \\
x_{13} = y_{12} \circ y_{13} & x_{14} = y_{13} \circ y_{14} & x_{15} = y_{14} \circ y_{15} & x_{16} = y_{15} \circ y_{16} \\
= A \circ C & = C \circ A & = A \circ C & = C \circ T \\
= T & = T & = T & = C \\
x_{17} = y_{16} \circ y_{17} & x_{18} = y_{17} \circ y_{18} & x_{19} = y_{18} \circ y_{19} & x_{20} = y_{19} \circ y_{20} \\
= T \circ G & = G \circ G & = G \circ C & = C \circ A \\
= T & = C & = A & = T \\
x_{21} = y_{20} \circ y_{21} & x_{22} = y_{21} \circ y_{22} & x_{23} = y_{22} \circ y_{23} & x_{24} = y_{23} \circ y_{24} \\
= A \circ C & = C \circ T & = T \circ C & = G \circ G \\
= T & = C & = G & = A \\
x_{25} = y_{24} \circ y_{25} & x_{26} = y_{25} \circ y_{26} & x_{27} = y_{26} \circ y_{27} & x_{28} = y_{27} \circ y_{28} \\
= G \circ T & = T \circ A & = A \circ C & = C \circ U \\
= T & = C & = T & = G \\
x_{30} = y_{29} \circ y_{30} & x_{31} = y_{30} \circ y_{31} & x_{32} = y_{31} \circ y_{32} & x_{33} = y_{32} \circ y_{33} \\
= U \circ U & = U \circ C & = C \circ G & = G \circ G \\
= T & = C & = A & = C \\
x_{34} = y_{33} \circ y_{34} & x_{35} = y_{34} \circ y_{35} & x_{36} = y_{35} \circ y_{36} & x_{37} = y_{36} \circ y_{37} \\
= G \circ T & = T \circ A & = A \circ U & = U \circ U \\
= T & = C & = C & = T
\end{array}
$$

$$
\begin{array}{llllllll}
x_{37} & = y_{36} \circ y_{37} & x_{38} & = y_{37} \circ y_{38} & x_{39} & = y_{38} \circ y_{39} & x_{40} & = y_{39} \circ y_{40} \\
 & = U \circ U & & = U \circ C & & = C \circ U & & = U \circ U \\
 & = T & & = C & & = G & & = T
\end{array}
$$

After decryption, the original message is recovered as shown:

$$D_2(C_1) = M =$$

(*TCGA TCGG TTTC TTTC TCAT TCGA TCTG TCAC TCCT TCGT*)

From the obtained DNA sequence, the message is converted back into textual form based on DNA encoding rule and ASCII table.

The final form of the plaintext recovered is shown as follows:

$$
\begin{array}{ll}
TCGA \rightarrow 0100\ 1101 \rightarrow N(4E) & TCGG \rightarrow 0100\ 1111 \rightarrow O(4F) \\
TTTC \rightarrow 0101\ 0100 \rightarrow T(54) & TTTC \rightarrow 0101\ 0100 \rightarrow T(54) \\
TCAT \rightarrow 0100\ 1001 \rightarrow I(49) & TCGA \rightarrow 0100\ 1110 \rightarrow N(4E) \\
TCTG \rightarrow 0100\ 0111 \rightarrow G(47) & TCAC \rightarrow 0100\ 1000 \rightarrow H(48) \\
TCCT \rightarrow 0100\ 0001 \rightarrow A(41) & TCGT \rightarrow 0100\ 1101 \rightarrow M(4D)
\end{array}
$$

CHAPTER 5: RESULTS AND DISCUSSIONS

This chapter comprises of 2 main sections: security and efficiency analysis of the proposed method and a comparative study with two established methods: Padmapriya's method [76] and Markovski's method [57].

5.1 Security Analysis

The proposed scheme was simulated using Python 3.11 on a system with 2.38 GHz processor and 16GB RAM.

5.1.1 Brute Force Attack

The strength of any encryption system lies in its key space, which is the total number of unique keys that an attacker requires for a brute force attack. For the proposed system, the key space, K is determined by secret random parameters involved in the encryption process, which are the leaders, DNA encoding rule, quasigroup table and parastrophe table. The proposed encryption system applies a quasigroup operation at every step of the DNA-encoded message. As stated in Theorem 2 by [26], this means that an attacker trying to reverse the transformation must search through all possible sequences of quasigroup operations, even when the input and output are known. In this thesis, although the cryptographic system uses only one quasigroup table and one parastrophe table for the entire message, each symbol in the message is encrypted using a different pair of inputs (previous ciphertext and current plaintext). As a result, from the attacker's perspective, the actual operation applied at each step appears to be different and unknown. This justifies modelling the encryption system as a sequence $f_1, f_2, \cdots, f_L$ of unknown two-input functions, where L is the number of quasigroup operations performed,

each selected from a total of $C = 161{,}280 \times 6 = 967{,}680$ possible quasigroup and parastrophe combinations. The value of C represents the total number of distinct quasigroup operations that can be applied at each step of the encryption process. Based on Table 3.6, there are precisely $161{,}280$ quasigroups for $n = 5$. In the context of our algorithm, this corresponds to the number of possible quasigroup tables that can be randomly generated for Phase I of the encryption. As defined in Section 3.3.2, every quasigroup has 5 parastrophe, leading to a total of 6 related quasigroup (the original plus its five conjugates). While not all re always distinct, the maximum number of distinct parastrophes is 6. By applying the rule of product, the total number of unique combinations of a quasigroup and one of its parastrophes is:

C = number of quasigroups x number of parastrophes per quasigroup

= 161,280 x 6 = 967,680

Thus, applying the rule of product, the key space becomes

$$K = R \times L \times C^m$$

$$K = 8 \times 5^2 \times (967{,}680)^m$$

where:

- R: Number of DNA encoding rules
- L: Number of leader combinations
- C: Number of possible quasigroup and parastrophe combinations
- m: Length of the encoded message in DNA form

As a simple example, we use $m = 10$, in an exhaustive attack, the key space is:

$$K = 200 \times (967{,}680)^{10}$$

$$K = 1.44 \times 10^{62}$$

As can be observed, the key space is exponentially large even with just the length of encoded message being 10, this implies that it would be difficult for an attacker to break the system using brute force attack. Compared with two other cryptosystems' key space, specifically Umesh Kumar's quasigroup-based block cipher method which is $2^{128} \approx 3.4 \times 10^{38}$ [87] and Al-Ahmadi's DNA-based method which is $2^{80} \approx 1.21 \times 10^{24}$ [88], the key space of the proposed method is significantly bigger. The algorithm enhances security through the random parameters. Even with the knowledge of the plaintext and ciphertext, attackers would struggle to reconstruct the encryption process without the specific information of the key.

5.1.2 Known Plaintext Attack (KPA)

In the Known Plaintext Attack (KPA), the adversaries have access to both plaintext and its corresponding ciphertext. In this type of scenario, they aim to reverse engineer the encryption process by deducing and analysing any possible patterns or relations between the ciphertext and its corresponding plaintext. Randomness and unpredictability are the keys to protecting the system against this attack, they make it difficult for the attackers to infer and correlate any useful patterns that may expose the system.

In this section, we analyse how the proposed method is resistant against KPA. We use chi-square test to evaluate the uniformity of the DNA bases (A, C, T, G, U) distribution in the ciphertext. As mentioned in [95], chi-square test is used to check randomness of a string of numbers or symbols. The use of the Chi-square test to evaluate the uniformity of DNA base distribution in the ciphertext is a direct application of a fundamental principle in cryptography: a secure cipher must produce output that is statistically indistinguishable from random data [89]. If certain DNA bases appear more frequently than the others in the ciphertext, the adversary might be able to use these patterns to restructure the contents of the original plaintext. The application of the Chi-square test for cryptographic purpose is well-documented in both general cryptographic literature and in the specific field of DNA cryptography [11], [90], [91],. By applying the Chi-square test and obtaining high p-values for our ciphertext across different plaintext lengths, we are able to provide quantifiable, standards-based evidence that our proposed algorithm successfully eliminates statistical biases.

The chi-square statistics is numerically represented as:

$$\chi^2 = \sum_{i=1}^{n} \frac{(O_i - E_i)^2}{E_i}$$

Where:

- n is the number of distinct DNA bases (A, C, T, G, U)
- O_i is the observed frequency of the i^{th} base
- $E_i = \frac{N}{n}$ is the expected frequency where N is the total number of characters in the ciphertext.

The resulting chi-square statistic is then used to calculate the p-value. The closer the chi-square statistic is to 0 and the higher the p-value (usually more than 0.05), the more uniform the distribution.

Table 5.1 Chi-Square statistic and p-value for plaintext length of 500, 2000, 3500 and 5000

Plaintext Length	Chi-Square Statistic	p-value
500	0.5600	0.9674
2000	0.3944	0.9829
3500	2.1219	0.7133
5000	4.2240	0.3765

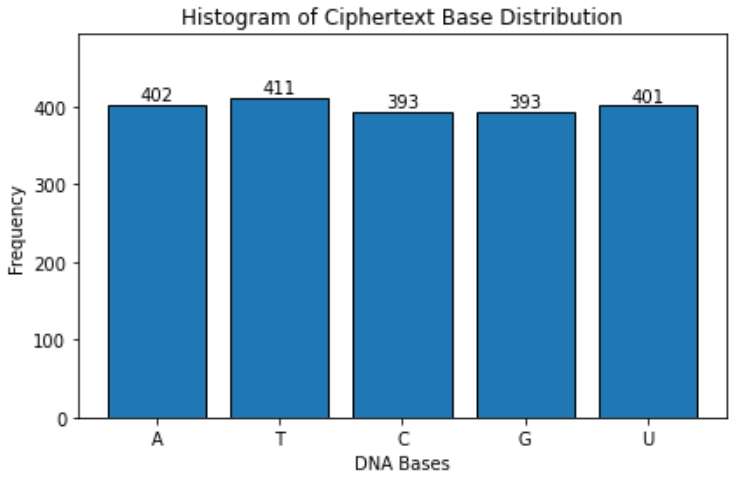

Figure 5.1 Histogram of Ciphertext Base Distribution for Plaintext of 500 Character Lengths

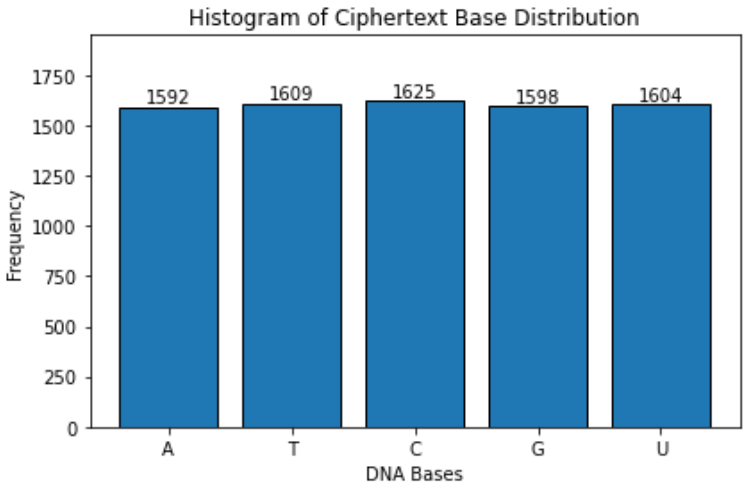

Figure 5.2 Histogram of Ciphertext Base Distribution for Plaintext of 2000 Character Lengths

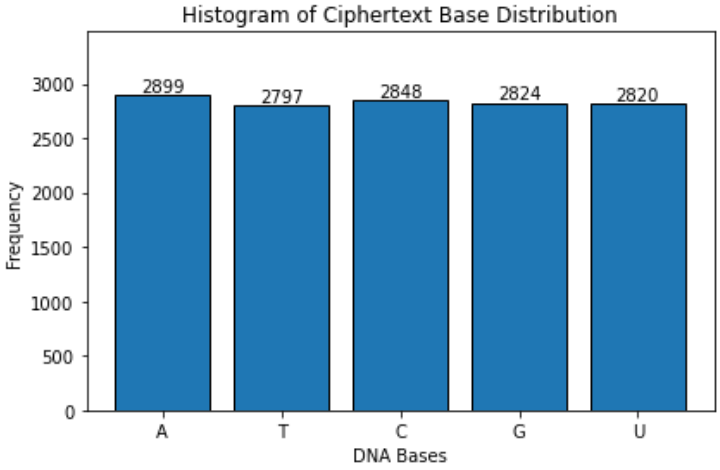

Figure 5.3 Histogram of Ciphertext Base Distribution for Plaintext of 3500 Character Lengths

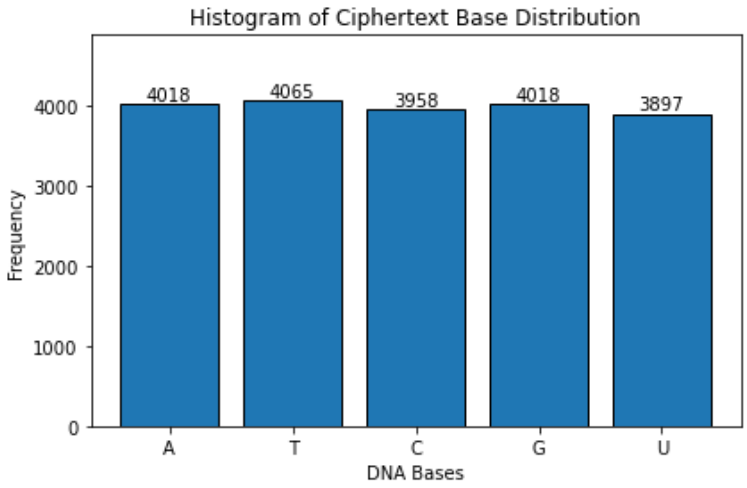

Figure 5.4 Histogram of Ciphertext Base Distribution for Plaintext of 5000 Character Lengths

All p-values are above the 0.05 threshold and all histograms of ciphertext base distribution for plaintext of 500, 2000, 3500 and 5000 are uniformly distributed and shows no sign of bias. The consistency in the distribution of each figure proves to be hard for the adversary to find any type of pattern or relation in the proposed system. Thus, the system is resistant against KPA.

5.1.3 Complexity Analysis

Complexity analysis determines the efficiency of an algorithm. According to the approximations from complexity theory, the smallest possible class of functions is used to express the growing rate of algorithm's runtime. For instance, if the number of operations is $1 + 2n$, then the complexity would be $O(n)$ and if the number of operations is $4 + n + n^3$, then the complexity would be $O(n^3)$.

Time complexity is defined as the time required to execute an algorithm. The runtime of an algorithm is defined as the sum of all operations. The time

required to convert plaintext to ciphertext is referred to as encryption time. The time complexity of an encryption scheme is the sum of the time required at phase 1 and phase 2.

For the proposed algorithm, the time complexity of the operations will be measured based on the order of the quasigroup. For the proposed method, the quasigroup has 5 elements $\{A, C, G, T, U\}$. The lookup and operation on the quasigroup table are constant time operations $O(1)$. The larger the quasigroup, the more elements will need to be processed in terms of memory $O(n^2)$, where n is the order of quasigroups, due to the need to store the quasigroup multiplication table. Since we are using two layers of encryption, where the first layer uses the original quasigroup and the second layer uses its conjugate, The time complexity for a single layer of encryption is $O(m)$, where m is the length of the message being encrypted, since each element of the message goes through the quasigroup multiplication operation once. With two layers, this results in $O(2m)$ time complexity for encryption and decryption, which simplifies to $O(m)$, as the constant factor can be disregarded in Big-O notation. An $O(m)$ algorithm performs a number of operations proportional to the size of the input n. For example, if n doubles, the time taken also doubles. Searching through an unsorted list of n elements for a specific item typically takes $O(m)$ time, as it requires examining each element once. This is generally faster and more efficient for large datasets than $O(n^2)$. Thus, since the encryption and decryption process operate with a linear time complexity of $O(m)$, the algorithm's performance remains efficient even as the input size increases.

Space complexity is defined as the amount of memory the algorithm uses as the input size grows. The quasigroup table has n^2 elements, where n is the number of elements in quasigroup, resulting in $O(n^2)$ space complexity for the table itself. This means that as the number of elements in the quasigroup increases, the space required to store the table increases significantly.

When encrypting a message, the ciphertext must be stored. The size of the ciphertext is directly proportional to the length of the input message, m. The DNA sequence requires space proportional to the length of input message m, meaning the total space complexity is $O(m)$ for the storage of the ciphertext. This means that the length of the message is directly proportional to the storage requirement for the ciphertext. When considering both the storage required for the quasigroup multiplication table and the storage for the ciphertext, the overall space complexity can be expressed as the sum of the two individual complexities. Thus, overall space complexity would be $O(m + n^2)$ due to the message size and quasigroup storage, which indicates that the storage requirements grow with both the message length and the size of the quasigroup.

The linear growth in time complexity with message length $O(m)$ ensures the system is suitable for long plaintexts, as the computational cost is proportional to input size. However, the quadratic growth in space complexity due to the quasigroup table $O(n^2)$ highlights a trade-off: increasing the quasigroup size enhances security but demands significantly more memory.

The cryptosystem achieves a balance between computational efficiency and storage demands, with its linear time complexity being a standout feature for practical applications. However, as the quasigroup size increases to enhance

security, the associated storage requirements must be carefully managed to avoid excessive resource consumption. This balance makes the cryptosystem particularly suitable for environments with moderate storage constraints and a need for fast encryption and decryption processes. In future work, we may consider looking into the possibility of developing algorithm which can reduce the space complexity while only marginally increase the time complexity.

5.1.4 Shannon's Entropy

In the field of cryptography, Shannon entropy is used to assess the strength of encryption systems. Shannon's entropy, developed by Claude Shannon, is a measure of uncertainty or randomness in a set of data, such as a sequence of text or encoded information. Higher entropy values indicate more randomness and unpredictability, while lower values imply more regularity or predictability. For instance, a repetitive sequence, such as "AAAA", has low entropy because there is little surprise in each new character, the next character is likely "a" again whilst a sequence like "AGCTGTCA", where each character is less predictable, has high entropy because each new character introduces more "surprise". An encryption system with high entropy means it is harder for attackers to reverse engineer the system, making the encryption system more secure.

Definition 5.1. For a random variable α with n possible values $\alpha_1, \dots, \alpha_n$ such that $P[\alpha = \alpha_i] = p_i$, we define its Shannon's entropy as

$$H(\alpha) \coloneqq \sum_{i=1}^{n} p_i \log_2 \frac{1}{p_i}$$

It is measured in bits.

Where:

- p_i represents the probability of each unique event i
- The $\log_2$ calculates the "information" of each event in bits.

If an event occurs with probability 1, it has 0 entropy (it's entirely predictable). The maximum entropy occurs when all events are equally probably, maximising uncertainty.

Suppose you have a simple text string like "AABBCCDD":

The probabilities for each character are $[0.25, 0.25, 0.25, 0.25]$ for "A", "B", "C" and "D".

Shannon's entropy calculation:

$$H = \sum_{1}^{4} 0.25 \cdot \log_2(4) = 2$$

Normalised Shannon entropy is a scaled version of Shannon entropy that adjusts for the size of the alphabet used in the data. It provides a value between 0 and 1, making it easier to compare entropy across datasets with different character sets or symbol sizes. Raw Shannon entropy values can vary significantly depending on the alphabet size. Normalised entropy scales these values to a consistent range (0 to 1).

Normalised entropy would compare this to the maximum entropy possible for a system with four unique characters:

$$\text{Max entropy} = \log_2(4) = 2$$

Giving a normalised entropy of 1, indicating maximal randomness for a sequence with four equally probable outcomes.

The maximum possible entropy $H_{max} = \log_2(n)$

$$\text{Normalised entropy} = \frac{H}{H_{max}}$$

In this thesis, we obtained the data through python coding for maximum possible entropy, Shannon Entropy and Normalised Shannon Entropy for plaintext and ciphertext of the proposed method. The obtained data are listed in the table shown below:

Table 5.2 Proposed Method's Shannon Entropy for Plaintext

Plaintext Length	Maximum Possible Entropy	Shannon Entropy	Normalised Shannon Entropy
500	5.209453	4.406603	0.845886
1000	5.285402	4.351083	0.823226
1500	5.523562	4.381499	0.793238
2000	5.523562	4.346915	0.786977
2500	5.554589	4.364680	0.785779
3000	5.672425	4.373745	0.771054

Table 5.3 Proposed Method's Shannon Entropy for Ciphertext

Ciphertext Length	Maximum Possible Entropy	Shannon Entropy	Normalised Shannon Entropy
500	2.321928	2.320869	0.999544
1000	2.321928	2.320915	0.999564
1500	2.321928	2.321286	0.999723
2000	2.321928	2.321455	0.999796
2500	2.321928	2.321513	0.999821
3000	2.321928	2.321871	0.999975

As can be observed, the normalised Shannon's entropy for ciphertext of length 500 to 3000 are all relatively close to one, showing high unpredictability of the system.

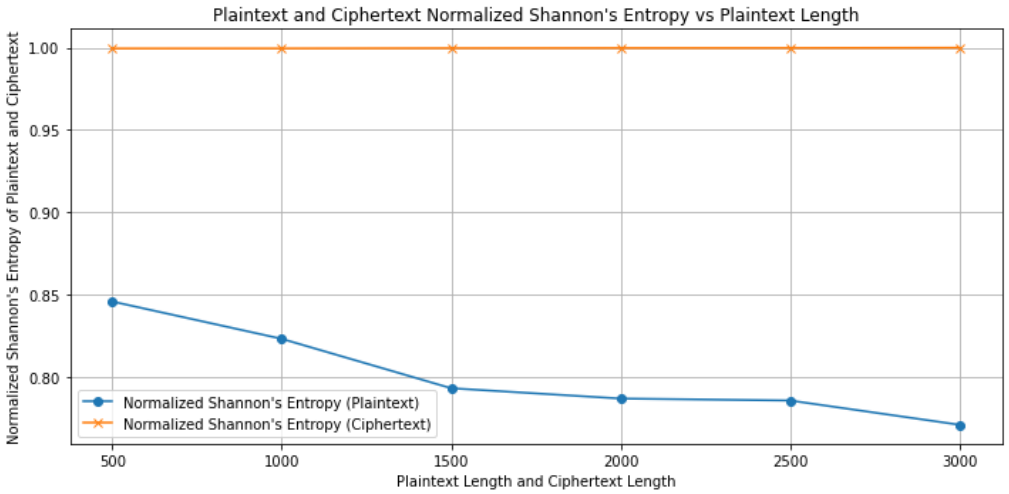

Figure 5.5 Graph of Normalised Shannon's Entropy of Plaintext and Ciphertext of the Proposed Method against Plaintext Length

Figure 5.5 shows the graphs of Shannon's Entropy and Normalised Shannon's Entropy of the Proposed Method against Plaintext Length and Ciphertext Length respectively. The blue line represents the normalised Shannon's Entropy of plaintext while the orange line represents normalised Shannon's entropy of ciphertext. For Shannon's entropy analysis, the plaintext starts with a low Shannon entropy (approximately 4.3 to 4.4), indicating a high degree of predictability and redundancy in the data. After encryption, the ciphertext entropy rises to approximately 2.32, which is close to the maximum possible entropy for the given plaintext size. This increase also suggests that the encryption process has effectively removed patterns and made the data more random.

For the analysis of Normalised Shannon's Entropy, the normalised entropy for plaintext begins near 0.77. After encryption, the normalised entropy is approximately 0.999, approaching to 1, which is the theoretical maximum for randomness. This indicates that the ciphertext has high unpredictability and randomness, which is pivotal property for a cryptosystem. The result of the normalised Shannon entropy of the ciphertext would not be affected and different even if the plaintext came from different domains as the transformation operations of the plaintext depends not on the context of the plaintext but the length of said plaintext.

The increase in both Shannon entropy and normalised Shannon entropy demonstrates the cryptosystem's ability to obscure patterns in the plaintext, making it resistant to statistical and frequency-based attacks. The graph has also demonstrated the effectiveness of the cryptosystem in transforming plaintext into ciphertext with significantly increased randomness, as indicated by both the result of Shannon's entropy and normalised Shannon's entropy. Overall, these results confirm that the cryptosystem is both effective and secure, as it achieves high levels of randomness and entropy in the ciphertext regardless of the plaintext length.

5.1.5 Provable Security in DNA and Quasigroup-based Cryptosystems: Current State and Limitations

There is limited literature discussing theoretical and provable security analysis for DNA-based and quasigroup-based cryptosystem and existing work remains primarily empirical or descriptive. For DNA cryptosystems, most studies focus on improving randomness and efficiency using DNA coding combined with

other mathematical models, with analysis including randomness test like NIST suite and resistance to common attacks like brute force attacks. For quasigroup-based cryptography, some work adopts provable security frameworks for quasigroup-based cryptosystems, for instance, symmetric encryption scheme based on quasigroups (SEBQ) has been proven to achieve IND-CPA security [92]. Nevertheless, while provable security frameworks are emerging for specific quasigroup-based encryption, overall coverage in literature remains less extensive compared to classical cryptosystems. Overall, while empirical and descriptive security analysis dominate in these areas, there are ongoing efforts toward integrating provable security frameworks particularly for quasigroup-based cryptosystems. DNA cryptosystems are generally at an earlier stage where provable security remains a research challenge due to novelty and complexity of biological encoding models. While a full provable security analysis is beyond the scope of this thesis, it represents a vital and recommended direction for future work.

5.2 Efficiency Analysis

5.2.1 Encryption and Decryption time

The results for encryption time of the proposed method for plaintext of different lengths, from 10 characters up to 500 characters are obtained and shown in the tables below:

Table 5.4 Encryption and decryption time for the proposed method

Plaintext Length	Encryption time (ms)	Decryption time (ms)
10	0.4479	0.1200
20	0.5256	0.2619
40	0.5820	0.4291
80	0.6747	0.8275
100	0.7610	1.0465
500	2.5488	5.3951

As shown in the table, the encryption and decryption time grows gradually as the plaintext length increases. For very short plaintexts (10 – 100 characters), the encryption and decryption time remains below 1ms, indicating that the overhead of the encryption and decryption procedure is minimal for small data. Even for 500 characters, the encryption and decryption time are only about 2.5488 ms and 5.3951 ms respectively.

These results indicate the efficiency of the encryption and decryption process of the proposed method. For typical applications, an encryption time of less than a millisecond (for up to 100 characters) is practically negligible. The linear increase in encryption time and decryption time with respect to the plaintext length also confirms that the proposed method has a time complexity of $O(m)$, as predicted in Section 5.1.3. Different character sets of the same length would produce very similar timing results because the algorithm's performance is fundamentally determined by the quantity of data (number of DNA bases to process) rather than the specific content of that data.

5.3 Comparative Analysis

5.3.1 Efficiency

In this section, the proposed method will be compared to two cryptographic methods which are Padmapriya's method [38] that involves DNA cryptography and Markovski's method [39] that involves quasigroup-based cryptography. These three methods are compared in terms of security (Shannon's entropy) and efficiency (encryption and decryption time). I implemented Padmapriya's and Markovski's method from scratch based on the description in [78] and [79] respectively. The results were not taken directly from the published papers but came from my own implementations of all three methods and exactly the same plaintext sets for all three methods to ensure fair comparison.

Padmapriya's scheme [38] is a two-phase symmetric key stream cipher which combines a DNA-derived One-Time Pad (OTP) with a frequency-based (Huffman style) prefix code to produce storage optimised ciphertext whilst Markovski's method [39] introduces BCMPQ, a symmetric block cipher that employs quasigroups of order 4 in a compact matrix form.

The proposed scheme is a hybrid that integrates two distinct concepts: DNA cryptography and quasigroup-based transformations. To properly assess the contribution of the proposed scheme in terms of security and efficiency, it is necessary to compare against a benchmark that represents each domain of the hybrid which are DNA and quasigroups. Thus, the comparative study focuses on two representative schemes, that is, Padmapriya's DNA-based method and Markovski's quasigroup-based method as they are directly comparable to the

proposed hybrid scheme. By comparing a pure DNA method and a pure quasigroup method, we can effectively demonstrate the synergistic advantages of the proposed hybrid approach. If the proposed scheme outperforms or matches both in key metrics, it strongly validates the hybrid design principle. Padmapriya's and Markovski's methods were chosen specifically because they are well-documented, implementable, and directly relevant to the core innovations of this thesis, thereby enabling a clear and interpretable comparative analysis.

The encryption time and decryption time for Padmapriya, Markovski and proposed method for different plaintext lengths are shown below:

Table 5.5 Encryption time of three methods for different plaintext lengths

Plaintext Length	Encryption time (ms)		
	Padmapriya	Markovski	Proposed
10	2	0.2402	0.4479
20	4	0.2899	0.5256
40	3	0.5242	0.5820
80	5	0.8386	0.6747
100	5	1.4333	0.7610
500	8	4.4841	2.5488

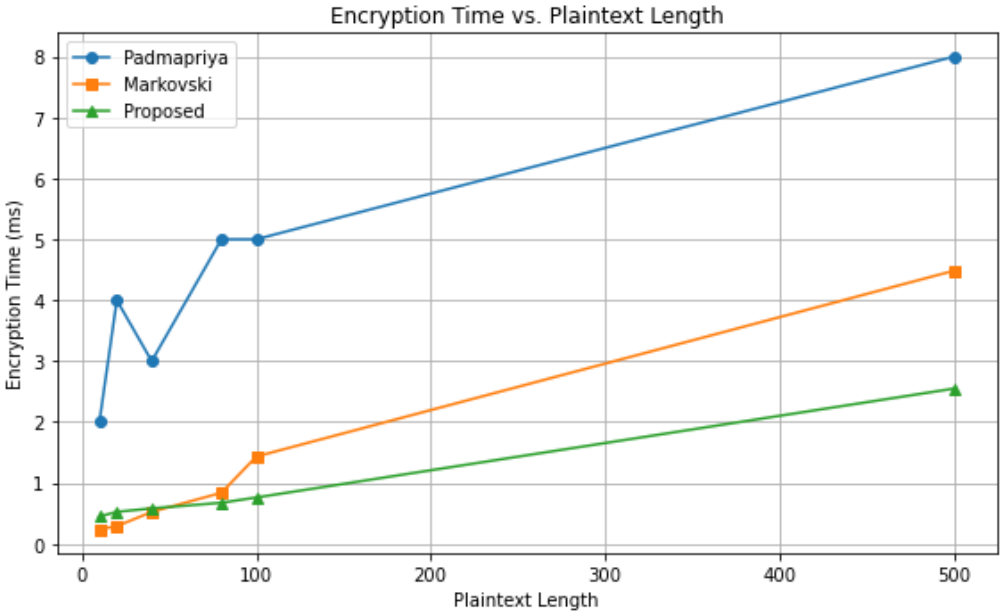

Figure 5.6 Graph of encryption time for the three methods against plaintext length

Table 5.6 Decryption time of three methods for different plaintext lengths

Plaintext Length	Decryption time (ms)		
	Padmapriya	Markovski	Proposed
10	2	0.1739	0.1200
20	2	0.2510	0.2619
40	3	0.4756	0.4291
80	3	0.8060	0.8275
100	3	1.3891	1.0465
500	7	7.6478	5.3951

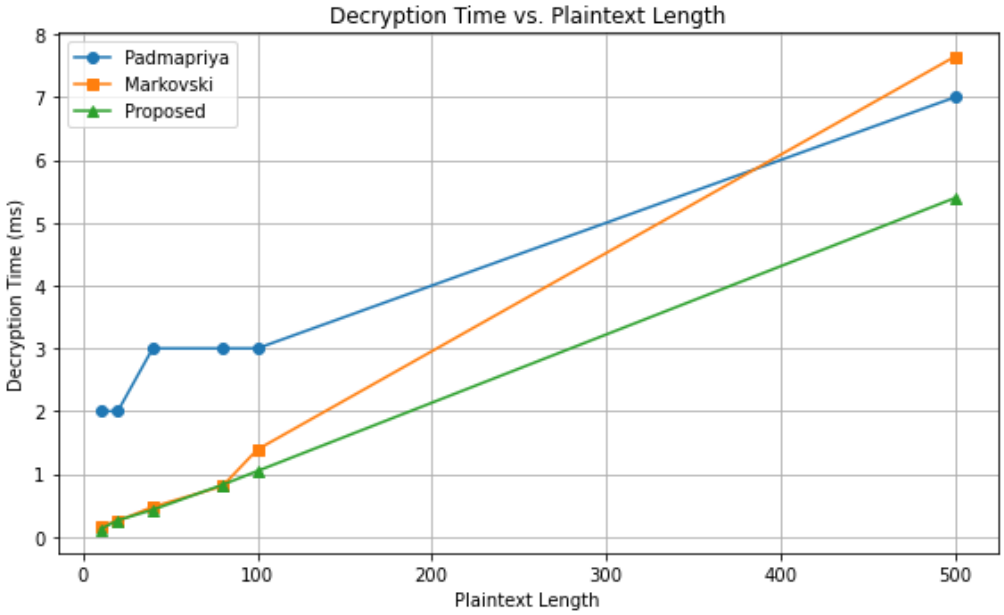

Figure 5.7 Graph of decryption time for the three methods against plaintext length

It can be observed that Padmapriya's method exhibits moderate but the slowest encryption times among the three methods, ranging from 2 ms (10 characters) to 8 ms (500 characters), thus not the most efficient compared to the other two methods. Markovski's method achieves the fastest encryption time for small plaintexts (10 – 20 characters), with time as low as 0.2402 ms for 10 characters. However, its encryption time surpasses the proposed method's time at larger input sizes like 500 characters with 4.4841 ms. Overall, Markovski's method leads for small plaintext lengths, while the proposed method shows superior performance at larger plaintext sizes.

The same cannot be said for the decryption times as well. Padmapriya's method shows much higher decryption time than the other two methods for plaintext lengths between 10 (2 ms) to 100 (3 ms). The decryption time for Markovski's method is longest among three methods when it comes to large plaintext like 500 characters. The proposed method, however, delivers the

fastest decryption times at all measure plaintext lengths, from 0.1200 ms (10 characters) to 5.3951 ms (500 characters), which indicates minimal overhead for both small and large messages.

Overall, for small plaintext sizes, Markovski's method encrypts the fastest while the proposed method decrypts the fastest and Padmapriya's method is generally in the slowest range. For larger plaintext sizes, the proposed method outperforms all the other methods for both encryption and decryption.

The rationale as to why Markovski's method leads in performance for small plaintext lengths while the proposed method shows superior performance at larger plaintext sizes stems from their respective algorithmic characteristics and scalability. Markovski's method uses quasigroups of order 4 and operates on fixed 64-bit blocks with compact matrix operations, which incurs minimal overhead on small data sizes. This design leads to very low overhead per block, and thus results in faster encryption time, making it highly efficient for small plaintext lengths because lightweight operations and small block handling dominate efficiency at this scale. However, as plaintext size grows, the overhead accumulates as more blocks are processed, and the time complexity is less favourable on large inputs, leading to slower performance as size increases. Conversely, the proposed method processes plaintext linearly as the operation on the quasigroup table is constant-time, and the encryption has a linear time complexity $O(m)$ with respect to plaintext length m. Although there is some overhead from the more complex transformations and two-phase structure, this overhead becomes negligible as plaintext length grows. This linear scaling ensures that the performance of the proposed method becomes increasingly

efficient for larger plaintext sizes and eventually outperforms Markovski's method as input size becomes substantial.

Thus, the comparative analysis for efficiency has shown that the proposed method is the most balanced choice, delivery high encryption speed and the lowest decryption time.

5.3.2 Security

The securities of the three methods are compared using normalised Shannon's entropy for ciphertext. All the data of the three methods for normalised Shannon's entropy are listed in the table as shown below:

Table 5.7 Normalised Shannon's entropy for ciphertext of all three methods

Plaintext Length	Normalised Shannon Entropy		
	Padmapriya	Markovski	Proposed
500	0.860536	0.972287	0.999544
1000	0.864609	0.979534	0.999564
1500	0.861672	0.982073	0.999723
2000	0.858049	0.987947	0.999796
2500	0.858435	0.988624	0.999821
3000	0.863743	0.991953	0.999975

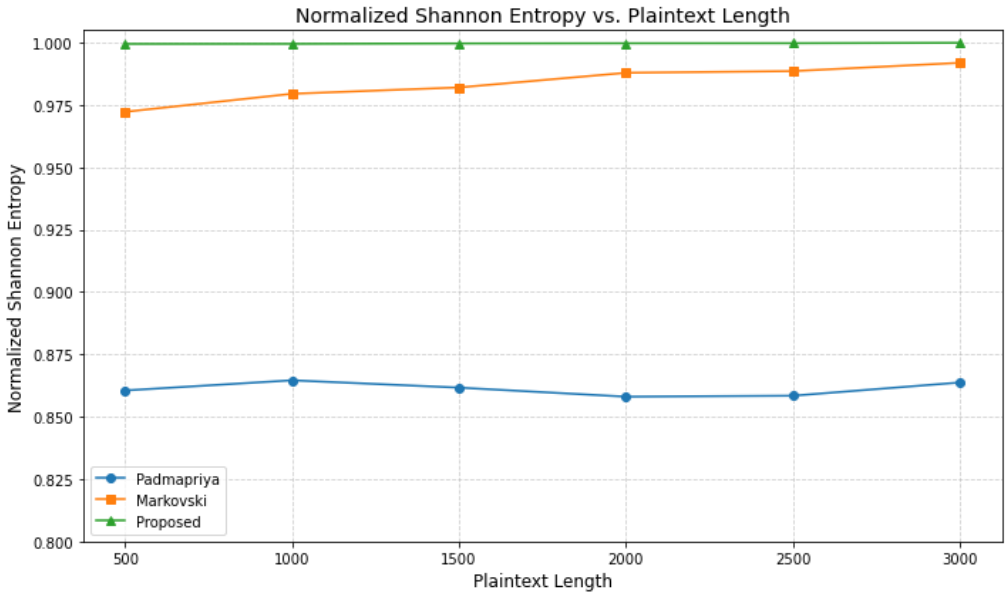

Figure 5.8 Graph of Normalised Shannon's entropy of all three methods against plaintext length

Padmapriya's method exhibits low normalised Shannon's entropy, ranging from 0.858049 to 0.864609. Markovski's method's value ranges from 0.972287 to 0.991953. The proposed method has highest value with 0.999544 at 500 characters and up to 0.999975 at 3000 characters. The proposed method achieves marginally highest values indicating that ciphertext distribution are highly uniform. This suggests strong randomness and minimal exploitable patterns in the ciphertext. High normalised Shannon's entropy is generally desirable as it reduces the likelihood of successful statistical attacks. The proposed method's value suggests it is more resistant to such attacks.

Padmapriya's scheme may offer storage efficiency, but it falls short in operational flexibility and speed. Each encryption run of Padmapriya's method must retrieve a random DNA sequence from public database GenBank and this introduces operational dependencies on Padmapriya's end. The proposed method does not rely on fetching large DNA sequences for each message,

thereby eliminating external key-retrieval overhead and still attaining high randomness which is comparable to Padmapriya's. Markovski's quasigroup matrix block cipher may be a lightweight block cipher whose entire public parameter set fits in 160 bytes and whose operations reduce to fixed-size Boolean matrix transforms which is ideal for resource-constrained hardware, yet it operates on fixed 64-bit blocks with quasigroups of order 4, which limits its security under modern block cipher standards and it yields only moderate randomness. In comparison, the proposed method uses higher order quasigroup and yields higher entropy.

5.4 Trade Off

Additional experiments were conducted to examine how encryption time and decryption time scale with larger quasigroup sizes and more phases.

Table 5.8 Encryption and decryption time of the proposed method with different quasigroup sizes

Quasigroup Size	Encryption Time (ms)	Decryption Time (ms)
4	0.202320	0.049240
5	0.301180	0.049940
6	0.474340	0.050080
7	0.736340	0.051020
8	1.330600	0.068320
9	1.815320	0.056280
10	2.320840	0.058460
11	3.430860	0.056280
12	4.767060	0.060100
13	7.151600	0.064280
14	9.155140	0.064840
15	10.776880	0.064580

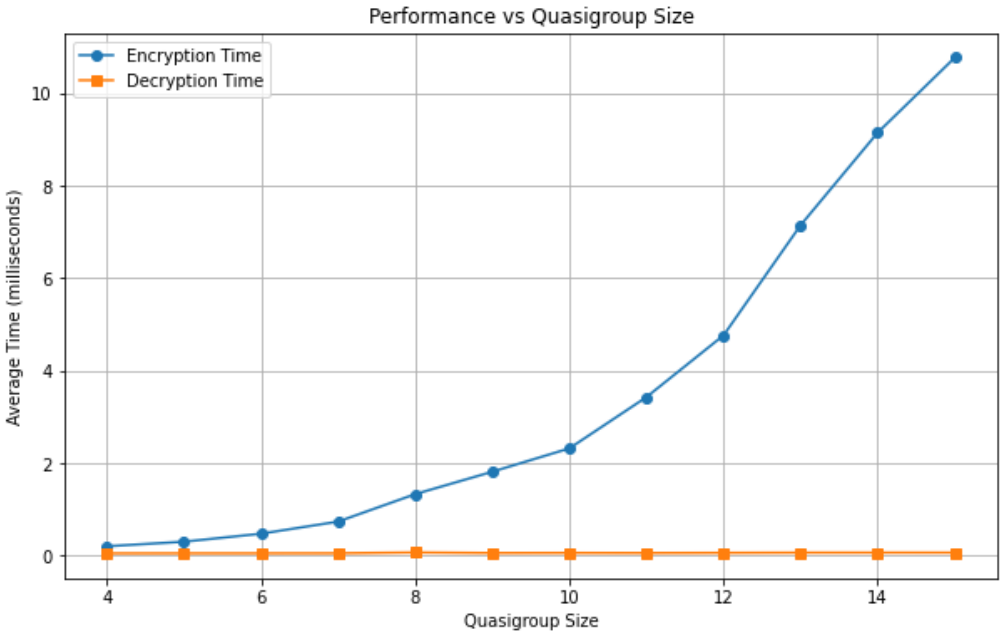

Figure 5.9 Graph of encryption and decryption time against different quasigroup size

According to Table 5.8 and Figure 5.9, the encryption time increases quite drastically from about 0.189980 ms for quasigroup of size 5 to 15.168740 ms for quasigroup of size 15. This rapid rise indicates that larger quasigroups demand significantly more computation in the encryption phase, making the method less practical for real-time or resource constrained environments. The decryption time, however, does not increase as drastically as the encryption time. This is due to the fact that decryption requires no random generation or table construction as it merely applies the already established inverse operations with known keys and tables.

It is true that a larger quasigroup size theoretically expands the key space and can enhance security. However, once a certain point is reached, the slowdown in performance outweighs the security boost gained, making it less practical for applications. Quasigroup of size 5 strikes a practical balance as it

still manages to offer large numbers of possible quasigroups while keeping the encryption time under 1ms for small and moderate input sizes.

Table 5.9 shows the encryption and decryption time of the proposed method with different number of phases ranging from 2 to 10. Each additional phase adds another transformation step, although it aims to improve complexity in the ciphertext, it also increases computational overhead.

Table 5.9 Encryption time and decryption time of the proposed method for different numbers of phases

Number of Phases	Encryption Time (ms)	Decryption Time (ms)
2	0.401020	0.053100
4	0.577720	0.087640
6	0.878320	0.126000
8	1.186460	0.165060
10	1.596760	0.202080

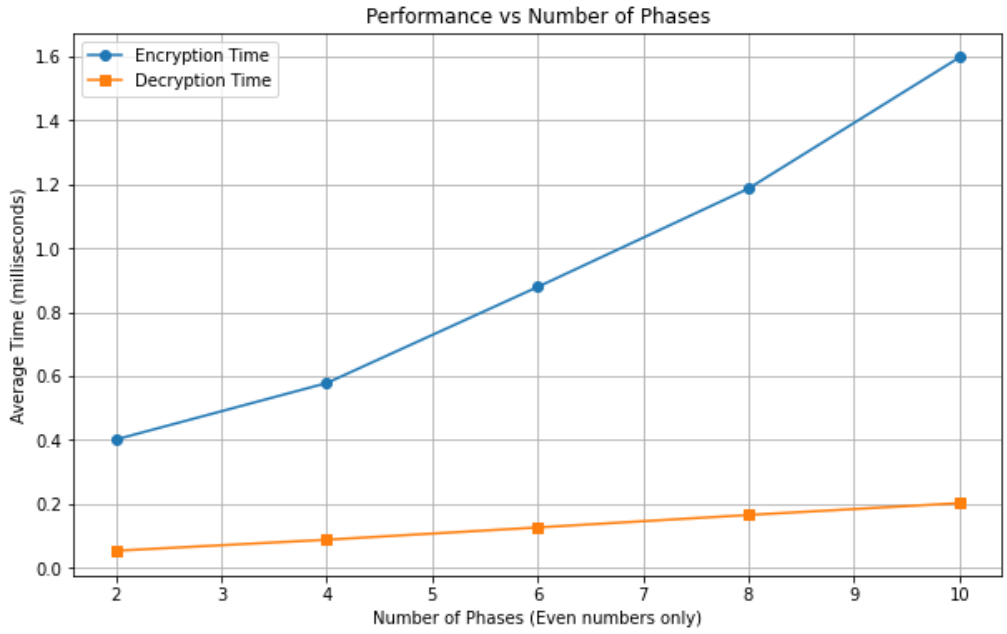

Figure 5.10 Graph of encryption and decryption time against number of phases

Table 5.10 Normalised Shannon Entropy of Plaintext and Ciphertext of the proposed method for different quasigroup

Quasigroup Size	Normalised Shannon Entropy of Plaintext	Normalised Shannon Entropy of Ciphertext
4	0.4855	0.9995
5	0.6284	0.9996
6	0.5645	0.9995
7	0.6736	0.9992
8	0.6304	0.9991
9	0.5966	0.9990
10	0.5693	0.9993
11	0.5466	0.9986
12	0.5275	0.9990
13	0.5867	0.9986
14	0.5702	0.9988
15	0.5557	0.9987

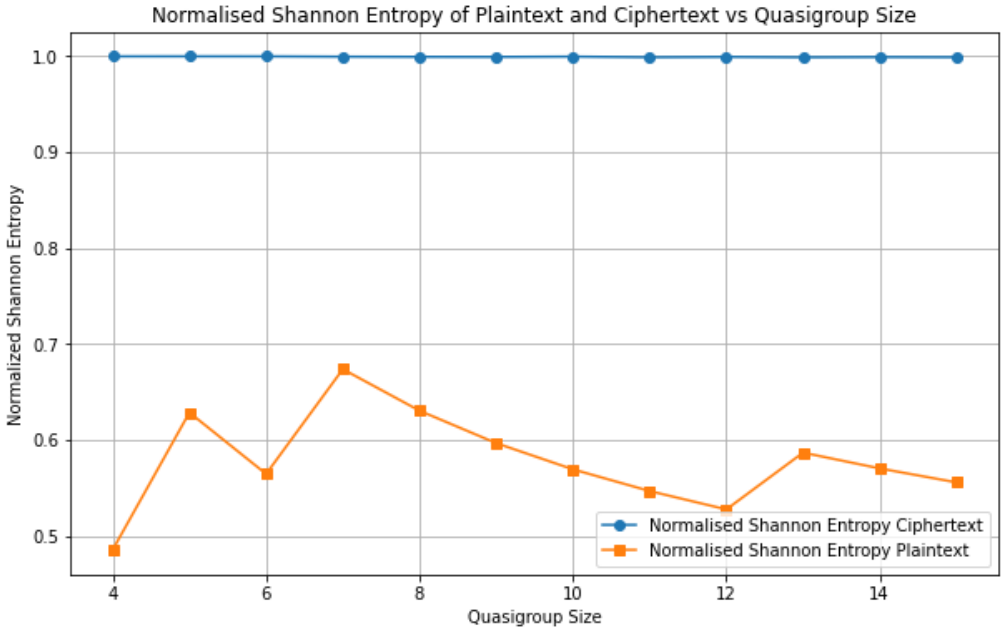

Figure 5.11 Graph of normalised Shannon entropy of plaintext and ciphertext against quasigroup sizes

Table 5.11 Normalised Shannon Entropy of Plaintext and Ciphertext of the proposed method for different numbers of phases

Number of Phases	Encryption Time (ms)	Decryption Time (ms)
2	0.4182	0.9992
4	0.4182	0.9993
6	0.4182	0.9994
8	0.4182	0.9992
10	0.4182	0.9995

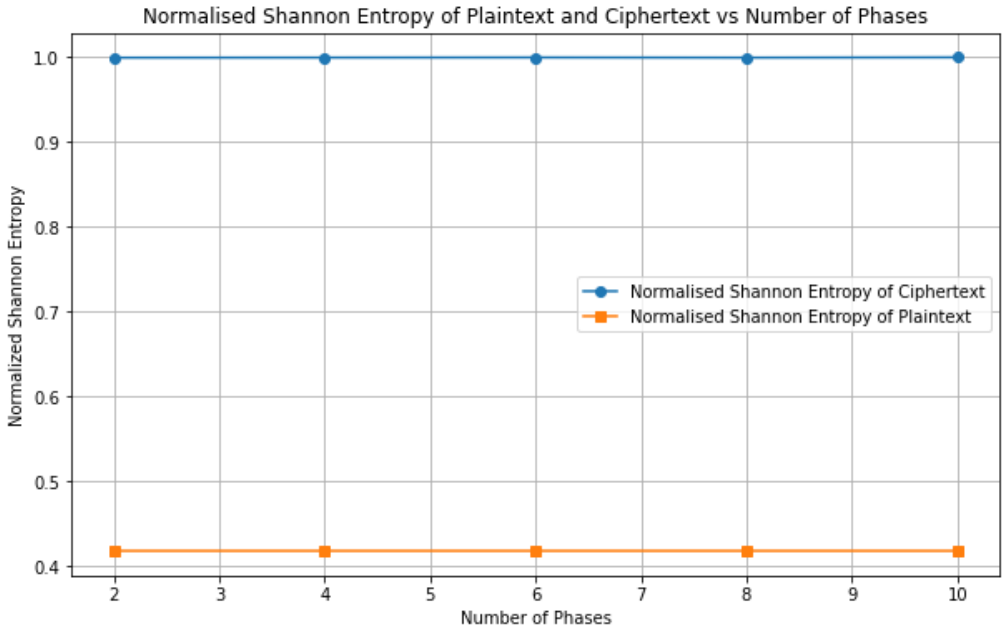

Figure 5.12 Graph of normalised Shannon entropy of plaintext and ciphertext against number of phases

The proposed encryption method operates entirely on DNA symbols, where the input plaintexts are transformed into DNA bases before applying quasigroup and parastrophic transformations that produce essentially random-like ciphertexts. The randomness and uniformity analysis in Chapter 5 has shown that the ciphertext distributions remain consistent across varying plaintext

lengths, indicating robust statistical obfuscation that should generalise across different plaintext domain types. However, while the method generally achieves excellent diffusion and near uniform distribution with normalised entropy $>$ 0.99 in most cases, extremely low entropy domains, for instance, highly repetitive plaintext such as long strings of identical characters can result in slightly reduced ciphertext entropy. The proposed method has been tested with low entropy plaintext AAAAAAAAAA of length 10 which yields normalised Shannon entropy of ciphertext of 0.816376, comparatively lower than the random plaintext of length 10 which yields normalised Shannon entropy of ciphertext of 0.963329. The combination of a large key space, random initial parameters and multiple quasigroup phases ensure that any domain specific characteristics are effectively masked. For typical plaintext domains, the ciphertext is expected to maintain high entropy and exhibit uniform symbol distribution.

From Table 5.9 and Figure 5.10, it can be seen that the encryption time grows from about 0.286640 ms at 2 phases to 1.203000 ms at 10 phases. The decryption time increases proportionally from 0.062660 ms to 0.283780 ms. As mentioned before, each phase adds an additional transformation layer, theoretically enhancing security. However, beyond 2 phases, each additional layer adds only a small boost in security compared to the extra work it requires. For practicality, 2 phases of quasigroup-based DNA encryption already provide robust complexity, making additional phases unnecessary and redundant. While more phases could be beneficial for extremely sensitive data, real-world systems typically require a balance between encryption strength and latency. Doubling or tripling the encryption time to add extra layers may not be

worthwhile if 2 phases are already sufficient to strike a balance between security and efficiency.

Based on Table 5.10 and Figure 5.11, all ciphertexts achieve entropy near 1, which is the theoretical maximum for a uniformly random sequence. It is observed that larger quasigroups do not produce more random ciphertexts. Increasing quasigroup size greatly increases computational cost but does not effectively increase the normalised Shannon entropy of ciphertext. Therefore, the overhead is not justified from the perspective of Shannon entropy-based security.

According to Table 5.11 and Figure 5.12, there is no noticeable gain in normalised Shannon entropy when it comes to the number of phases. Increasing the number of phases to 4, 6, 8 or 10 does not produce any significant increase in the normalised Shannon entropy. Thus, adding more than 2 phases significantly increases the computational overhead without any measurable improvement in the normalised Shannon entropy of ciphertext. Minimal configurations already achieve the theoretical randomness bound, making further expansion is deemed unnecessary.

In short, while it is technically feasible to increase the quasigroup size or the number of phases, these results show that doing so significantly impacts performance without providing a reasonable improvement in security for real-world applications.

CHAPTER 6: CONCLUSION

The research sets out to explore and develop a novel cryptographic algorithm by integrating the unique properties of DNA with nonassociative transformations provided by quasigroups. The proposed method distinguishes itself from traditional cryptosystems by directly mapping plaintext into DNA form using a randomly chosen DNA encoding rule and then applying a two-phase encryption process that leverages both a random quasigroup table and one of its random parastrophes. One of the key innovations in the proposed method is the introduction of element Uracil (U). The design choice not only increases the key space but also obfuscates statistical patterns, thereby enhancing the system's resistance against known plaintext and statistical attacks. The two-phase encryption scheme significantly increases the complexity and randomness of the ciphertext, as evidenced by the high normalised Shannon entropy value which approaches 1. The extensive key space demonstrates that even for a small encoded message length of 10, the key space reaches an exponential size (approximately 1.44×10^{62}). The enormous key space ensures that an exhaustive search attack would be computationally infeasible, thereby reinforcing the cryptosystem's security.

In addition to its strong security, the proposed method also demonstrates high efficiency. Experimental results show that both the encryption and decryption times scale linearly with the plaintext length. Comparative analysis with existing DNA cryptographic schemes such as Padmapriya's and Markovski's methods indicate that while all methods achieve high levels of randomness, the proposed method offers a balanced approach with faster encryption and decryption time, especially for larger plaintexts.

Furthermore, additional experiments were conducted to assess the trade-offs associated with using larger quasigroup sizes and increasing the number of encryption phases. These investigations revealed that although increasing these parameters could theoretically enhance security by expanding the key space and adding more layers of transformation, the practical impact on performance is significant. In particular, encryption times grow rapidly with larger quasigroup sizes and additional phases, while security gains remain marginal beyond the chosen configuration. This finding validates the design choices made in this thesis, ensuring that the system achieves robust security without compromising efficiency.

Despite the promising results, the current implementation is limited to text file encryption. Future work could focus on extending the method to handle multimedia data such as images and audio, as well as exploring further optimisations in key management and transformation efficiency. On the whole, the proposed method represents a significant step forward in the application of quasigroup in DNA cryptography.

REFERENCES

[1] J. F. Dooley, *A brief history of cryptology and cryptographic algorithms*, vol. 21. New York: Springer, 2013. doi: 10.5860/choice.51-4489.

[2] Dave, "Understanding Classical cryptography," Coded Insights.

[3] D. Davies, "A brief history of cryptography," *Information Security Technical Report*, vol. 2, no. 2, pp. 14–17, 1997.

[4] W. Trappe and L. C. Washington, *Introduction to cryptography with Coding Theory 3rd*. 2007.

[5] B. Preneel, C. Paar, and J. Pelzl, *Understanding Cryptography: A Textbook for Students and Practitioners*, vol. 2009, no. April. 2009.

[6] D. E. Standard, "Data encryption standard," *Federal Information Processing Standards Publication*, vol. 112, no. 3, 1999.

[7] NIST, "FIPS 197 Advanced encryption standard," 2001.

[8] D. R. Stinson and M. B. Paterson, *Cryptography Theory and Practice 4th Edition*, vol. 1, no. 1. 2018.

[9] "What Are Cryptographic Attacks?: The Complete Guide." Accessed: Sep. 20, 2024. [Online]. Available: https://www.goallsecure.com/blog/cryptographic-attacks-complete-guide/

[10] T. Hanoymak, "On provable security of cryptographic schemes," *International Journal of Information Security Science*, vol. 2, no. 2, pp. 44–56, 2013.

[11] A. J. Menezes, P. C. Van Oorschot, and S. A. Vanstone, *Handbook of applied cryptography*. 1996. doi: 10.2307/2589608.

[12] S. Ramakrishnan, *Cryptographic and Information Security Approaches for Images and Videos*. CRC Press, 2018.

[13] A. Biryukov, "Chosen plaintext attack," in *Encyclopedia of Cryptography, Security and Privacy*, Boston, MA: Springer US, 2011, pp. 205–206.

[14] Simon Burge, "8 Types of Attack in Cryptography," ISJ INTERNATIONAL SECURITY JOURNAL. Accessed: Sep. 20, 2024. [Online]. Available: https://internationalsecurityjournal.com/types-of-attack-in-cryptography/#:~:text=Cryptography%20attacks%20are%20malicious%20attempts,and%20availability%20of%20encrypted%20data.

[15] "Cryptography: Theory and practice," *Computers & Mathematics with Applications*, vol. 30, no. 9, 1995, doi: 10.1016/0898-1221(95)90225-2.

[16] L. Chu, Y. Su, X. Yao, P. Xu, and W. Liu, "A review of DNA cryptography," *Intelligent Computing*, vol. 4, p. 0106, 2025.

[17] Z. Zhang and Z. Zhang, "DNA Information Storage and Cryptography System," *Academic Journal of Science and Technology*, vol. 10, no. 1, pp. 243–249, 2024.

[18] Sénat & Assemblée nationale, "Science and Technology Briefings: Briefing 29 – DNA data storage," 2021. Accessed: Jul. 15, 2025. [Online].

Available: https://www.assemblee-nationale.fr/commissions/opecst-index.asp

[19] Y. Zhang and L. H. Bochen Fu, “Research on DNA Cryptography,” in *Applied Cryptography and Network Security*, vol. 357, Springer, 2012, pp. 357–376. doi: 10.5772/34510.

[20] L. N. de Castro, *Fundamentals of natural computing: Basic concepts, algorithms, and applications*. 2006.

[21] O. Tornea, “Contributions to DNA Cryptography: Applications to Text and Image Secure Transmission,” University of Nice Sophia Antipolis, 2013.

[22] L. M. Adleman, “Molecular computation of solutions to combinatorial problems,” *Science (1979)*, vol. 266, no. 5187, 1994, doi: 10.1126/science.7973651.

[23] M. Mondal and K. S. Ray, “Review on DNA cryptography,” *arXiv preprint*, 2019.

[24] S. KK*, “DNA Cryptography an Area of DNA Computing,” *Bioinformatics & Proteomics Open Access Journal*, vol. 1, no. 1, 2017, doi: 10.23880/bpoj-16000103.

[25] A. Mileva, “New developments in quasigroup-based cryptography,” in *Multidisciplinary Perspectives in Cryptology and Information Security*, 2014. doi: 10.4018/978-1-4666-5808-0.ch012.

[26] S. Markovski, D. Gligoroski, and V. Bakeva, “Quasigroup string processing. Part, 1,” pp. 1–2, 1999.

[27] S. K. Tiwari, A. Awasthi, S. Chkrabarti, and S. Yadav, “INRU: A Quasigroup Based Lightweight Block Cipher,” *arXiv preprint arXiv:2112.07411*, 2021.

[28] D. Gligoroski, S. Markovski, and S. J. Knapskog, “A public key block cipher based on multivariate quadratic quasigroups,” *arXiv preprint*, 2008.

[29] D. Gligoroski *et al.*, “MQQ-SIG: An ultra-fast and provably CMA resistant digital signature scheme,” in *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 2012. doi: 10.1007/978-3-642-32298-3_13.

[30] G. Teşeleanu, “The Security of Quasigroups Based Substitution Permutation Networks,” in *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 2023. doi: 10.1007/978-3-031-32636-3_18.

[31] M. Battey and A. Parakh, “Efficient quasigroup block cipher for sensor networks,” in *2012 21st International Conference on Computer Communications and Networks, ICCCN 2012 - Proceedings*, 2012. doi: 10.1109/ICCCN.2012.6289294.

[32] M. Battey and A. Parakh, “An efficient quasigroup block cipher,” *Wirel Pers Commun*, vol. 73, pp. 63–76, 2013.

[33] D. Chauhan, I. Gupta, P. R. Mishra, and R. Verma, “Construction of cryptographically strong S-boxes from ternary quasigroups of order 4,” *Cryptologia*, vol. 46, no. 6, 2022, doi: 10.1080/01611194.2021.1934915.

[34] D. Nager, “Xifrat-Compact Public-Key Cryptosystems based on Quasigroups,” *Cryptology*, 2021.

[35] M. El-Hadedy, D. Gligoroski, and S. J. Knapskog, “High performance implementation of a public key block cipher - MQQ, for FPGA platforms,” in *Proceedings - 2008 International Conference on Reconfigurable Computing and FPGAs, ReConFig 2008*, 2008. doi: 10.1109/ReConFig.2008.11.

[36] H. Mihajloska, T. Yalcin, and D. Gligoroski, “How lightweight is the hardware implementation of quasigroup S-boxes,” in *Advances in Intelligent Systems and Computing*, 2013. doi: 10.1007/978-3-642-37169-1_12.

[37] M. Matsumoto, M. Saito, T. Nishimura, and M. Hagita, “A fast stream cipher with huge state space and quasigroup filter for software,” in *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 2007. doi: 10.1007/978-3-540-77360-3_16.

[38] M. K. Padmapriya and P. V. Eric, “A Technique of Data Security using DNA Cryptography with Optimized Data Storage,” *Journal of System and Management Sciences*, vol. 12, no. 4, 2022, doi: 10.33168/JSMS.2022.0425.

[39] S. Markovski, V. Dimitrova, Z. Trajcheska, M. Petkovska, M. Kostadinoski, and D. Buhov, “Block cipher defined by matrix presentation of quasigroups,” *Cryptology*, 2021.

[40] Sidhpurwala H., “A Brief History of Cryptography,” Red Hat Customer Portal. Accessed: Sep. 20, 2024. [Online]. Available: https://www.redhat.com/en/blog/brief-history-cryptography

[41] P. Bajpai, “Contribution of Ciphering machines in world wars-a review,” *Anusandhaan-Vigyaan Shodh Patrika*, vol. 7, no. 01, pp. 87–91, 2019.

[42] D. E. Standard, “Data encryption standard,” *Federal Information Processing Standards Publication*, vol. 112, no. 3, 1999.

[43] J. S. Revathy, M. S. Prakash, K. Lingeshwaran, J. D. Dhinakaran, and V. Harish, “A Comprehensive Study of the Advanced Encryption Standard (AES) for Secure Communications,” in *2025 3rd IEEE International Conference on Industrial Electronics: Developments & Applications (ICIDeA)*, IEEE, 2025, pp. 1–6.

[44] Leonard M. Adleman, “Computing with DNA,” *Sci Am*, vol. 279, no. 2, pp. 54–61, Aug. 1998.

[45] D. Boneh, C. Dunworth, and R. Lipton, “Breaking DES using a molecular computer,” 1996. doi: 10.1090/dimacs/027/04.

[46] Q. Ouyang, P. D. Kaplan, S. Liu, and A. Libchaber, “DNA solution of the maximal clique problem,” *Science (1979)*, vol. 278, no. 5337, 1997, doi: 10.1126/science.278.5337.446.

[47] J. Chen, “A DNA-based, biomolecular cryptography design,” in *Proceedings - IEEE International Symposium on Circuits and Systems*, 2003. doi: 10.1109/iscas.2003.1205146.

[48] Ashish Gehani, LaBean Thomas, and John Reif, “DNA-based cryptography,” *Springer Berlin Heidelberg*, pp. 167–188, 2004.

[49] S. T. Amin, M. Saeb, and S. El-Gindi, “A DNA-based implementation of yaea encryption algorithm,” in *Proceedings of the 2nd IASTED International Conference on Computational Intelligence, CI 2006*, 2006.

[50] M. X. Lu, X. J. Lai, G. Z. Xiao, and L. Qin, “Symmetric-key cryptosystem with DNA technology,” *Science in China, Series F: Information Sciences*, vol. 50, no. 3, 2007, doi: 10.1007/s11432-007-0025-6.

[51] G. Cui, L. Qin, Y. Wang, and X. Zhang, “An encryption scheme using DNA technology,” in *2008 3rd International Conference on Bio-Inspired Computing: Theories and Applications*, IEEE, 2008, pp. 37–42.

[52] X. J. Lai, M. X. Lu, L. Qin, J. S. Han, and X. W. Fang, “Asymmetric encryption and signature method with DNA technology,” *Science in China, Series F: Information Sciences*, vol. 53, no. 3, 2010, doi: 10.1007/s11432-010-0063-3.

[53] M. Sabry, M. Hashem, T. Nazmy, and M. E. Khalifa, “A DNA and Amino Acids-Based Implementation of Playfair Cipher,” 2010. [Online]. Available: https://www.researchgate.net/publication/45198045

[54] H. J. Shiu, K. L. Ng, J. F. Fang, R. C. T. Lee, and C. H. Huang, “Data hiding methods based upon DNA sequences,” *Inf Sci (N Y)*, vol. 180, no. 11, 2010, doi: 10.1016/j.ins.2010.01.030.

[55] Y. Zhang, B. Fu, and X. Zhang, “DNA cryptography based on DNA fragment assembly,” in *Proceedings - ICIDT 2012, 8th International Conference on Information Science and Digital Content Technology*, IEEE, 2012, pp. 179–182.

[56] O. Tornea and M. E. Borda, “Security and complexity of a DNA-based cipher,” in *Proceedings - RoEduNet IEEE International Conference*, 2013. doi: 10.1109/RoEduNet.2013.6511755.

[57] E. M. S. Hossain, K. M. R. Alam, M. R. Biswas, and Y. Morimoto, “A DNA cryptographic technique based on dynamic DNA sequence table,” in *2016 19th International Conference on Computer and Information Technology (ICCIT)*, IEEE, 2016, pp. 270–275. doi: 10.1109/ICCITECHN.2016.7860208.

[58] M. Karimi, M. A. Jinnah, U. Karachi, and P. W. Haider, “Cryptography using DNA Nucleotides,” 2017. [Online]. Available: www.ijcaonline.org

[59] N. S. Kolte, K. V Kulhalli, and S. C. Shinde, “DNA Cryptography using Index-Based Symmetric DNA Encryption Algorithm.” [Online]. Available: http://www.irphouse.com

[60] X. Zhang, Z. Zhou, and Y. Niu, “An Image Encryption Method Based on the Feistel Network and Dynamic DNA Encoding,” *IEEE Photonics J*, vol. 10, no. 4, 2018, doi: 10.1109/JPHOT.2018.2859257.

[61] S. S. Nafea and M. K. Ibrahem, “Cryptographic Algorithm based on DNA and RNA Properties,” *International Journal of Advanced Research in Computer Engineering & Technology (IJARCET)*, vol. 7, no. 11, pp. 804–811, 2018.

[62] B. D. Patnala and R. Kiran Kumar, “A Novel Level-Based DNA Security Algorithm Using DNA Codons,” in *SpringerBriefs in Applied Sciences and Technology*, Springer Verlag, 2019, pp. 1–13. doi: 10.1007/978-981-13-0544-3_1.

[63] O. F. Rashid, “Text Encryption Based on DNA Cryptography , RNA , and Amino Acid,” *E- Proceedings of The 5th International Multi-Conference on Artificial Intelligence Technology (MCAIT 2021) Artificial Intelligence in the 4th Industrial Revolution*, no. 2017, 2021.

[64] L. Euler, “Recherches sur une espece de carrés magiques,” *Commentationes Arithmeticae Collectae*, vol. 2, pp. 302–361, 1849.

[65] R. Moufang, “Zur Struktur von Alternativkörpern,” *Math Ann*, vol. 110, no. 1, 1935, doi: 10.1007/BF01448037.

[66] V. Shcherbacov, *Elements of quasigroup theory and applications*. 2017. doi: 10.1201/9781315120058.

[67] A. D. Keedwell and J. Dénes, *Latin Squares and their Applications: Second Edition*. 2015. doi: 10.1016/C2014-0-03412-0.

[68] R. Schauffler, “Eine Anwendung zykligcher Permutationen und ihre theorie,” Philipps-Universitat zu Marburg, 1948.

[69] C. Kościelny, “A method of constructing quasigroup-based stream-ciphers,” 1996.

[70] S. Markovski, D. Gligoroski, and S. Andova, “Using quasigroups for one-one secure encoding,” *Proc. VIII Conf. Logic and Computer Science ’LIRA*, vol. 97, pp. 157–162, 1997.

[71] Ritter T., “Latin squares: a literature survey,” Research comments from Ciphers By Ritter. Accessed: Jul. 15, 2025. [Online]. Available: http://www.ciphersbyritter.com/RES/LATSQ.HTM#Bose84

[72] E. Ochodková and V. Snášel, “Using quasigroups for secure encoding of file system,” *In Proceedings of the International Scientific NATO PfP/PWP Conference Security and Information Protection*, pp. 175–181, 2001.

[73] S. Markovski and V. Kusakatov, “Quasigroup String Processing: Part 2,” *Contributions, Section of Natural, Mathematical and Biotechnical Sciences*, vol. 21, no. 1–2, pp. 15–32, 2000.

[74] S. Markovski and V. Kusakatov, “Quasigroup String Processing: Part 3,” *Contributions, Section of Natural, Mathematical and Biotechnical Sciences*, vol. 24, no. 1–2, pp. 7–27, 2003.

[75] S. Markovski and V. Bakeva, “Quasigroup String Processing: Part 4,” *Contributions, Section of Natural, Mathematical and Biotechnical Sciences*, vol. 27, no. 1–2, 2007, doi: 10.20903/csnmbs.masa.2006.27.1-2.5.

[76] S. I. Marnas, L. Angelis, and G. L. Bleris, “All-Or-Nothing Transforms Using Quasigroups ,” *In Proc. 1st Balkan Conference in Informatics*, pp. 183–191, 2003.

[77] Y. Xu, “A cryptography application of conjugate quasigroups,” in *Proceedings - 2010 International Conference on Web Information Systems and Mining, WISM 2010*, 2010, pp. 63–65. doi: 10.1109/WISM.2010.15.

[78] Bakeva, Verica, Vesna Dimitrova, and Aleksandra Popovska-Mitrovikj, “Parastrophic quasigroup string processing,” 2011.

[79] A. Petrescu, “n-QUASIGROUP CRYPTOGRAPHIC PRIMITIVES: STREAM CIPHERS,” *Studia Universitatis Babes-Bolyai, Informatica, 55(2)*, vol. 55, 2010.

[80] S. Chakrabarti, S. K. Pal, and S. Gangopadhyay, “An improved 3-quasigroup based encryption scheme,” *ICT Innovations*, vol. 173, 2012.

[81] S. Markovski, “Design of crypto primitives based on quasigroups,” *Quasigroups and Related Systems*, vol. 23, no. 1, 2015.

[82] O. Bonham-Carter, A. Parakh, and D. Bastola, “SEncrypt: An encryption algorithm inspired from biological processes,” in *Proceedings - 12th IEEE International Conference on Trust, Security and Privacy in Computing and Communications, TrustCom 2013*, 2013. doi: 10.1109/TrustCom.2013.43.

[83] D. Chauhan, I. Gupta, P. R. Mishra, and R. Verma, “An ultra-lightweight block cipher with string transformations,” *Cryptologia*, 2023, doi: 10.1080/01611194.2023.2224107.

[84] N. A. N. Abdullah *et al.*, “A THEORETICAL COMPARATIVE ANALYSIS OF DNA TECHNIQUES USED IN DNA BASED CRYPTOGRAPHY,” *J Sustain Sci Manag*, vol. 17, no. 5, pp. 165–178, May 2022, doi: 10.46754/jssm.2022.05.014.

[85] A. P. Thiruthuvadoss, “Comparison and Performance Evaluation of Modern Cryptography and DNA Cryptography,” Royal Institute of Technology, 2012.

[86] C. C. Lindner and D. Steedley, “On the number of conjugates of a quasigroup,” *Algebra Universalis*, vol. 5, no. 1, 1975, doi: 10.1007/BF02485252.

[87] U. Kumar and V. C. Venkaiah, “A Family of Block Ciphers Based on Multiple Quasigroups,” *Cryptology*, 2022.

[88] W. A. E. Al-Ahmadi, A. O. Aljahdali, F. Thabit, and A. Munshi, “A secure fingerprint hiding technique based on DNA sequence and mathematical function,” *PeerJ Comput Sci*, vol. 10, p. e1847, 2024.

[89] J. Katz and Y. Lindell, *Introduction to modern cryptography: principles and protocols*. Chapman and hall/CRC, 2007.

[90] D. Sharma, “Implementing Chi-Square method and even mirroring for cryptography of speech signal using Matlab,” in *Proceedings on 2015 1st International Conference on Next Generation Computing Technologies, NGCT 2015*, 2016. doi: 10.1109/NGCT.2015.7375148.

[91] N. H. Munshi, P. Das, and S. Maitra, “Chi-Squared Test Analysis on Hybrid Cryptosystem,” *Micro and Nanosystems*, vol. 14, no. 1, 2021, doi: 10.2174/1876402913666210508235706.

[92] S. Kumar, H. Singh, I. Gupta, and A. J. Gupta, “Symmetric Encryption Scheme Based on Quasigroup Using Chained Mode of Operation,” 2024.

[93] M. A. Abdelaal, A. I. Moustafa, H. Kasban, H. Saleh, H. A. Abdallah, and M. Y. I. Afifi, “DNA-Inspired Lightweight Cryptographic Algorithm for Secure and Efficient Image Encryption,” *Sensors*, vol. 25, no. 7, p. 2322, 2025.

[94] I. Aisah, E. Djauhari, and A. Singgih, “Dihedral Group in The Ancient Genetic,” *Jurnal Matematika Integratif*, vol. 16, no. 1, 2020, doi: 10.24198/jmi.v16i1.26646.

[95] K. A. Ameen, W. khalid Abdulwahab, and Y. N. A. Taher, “Encryption Technique Using a Mixture of Hill Cipher and Modified DNA for Secure Data Transmission,” *International Journal of Computing*, vol. 17, no. 1, pp. 1–9, 2025.

[96] Anne-Sophie Boutaud, “Data storage: the DNA revolution,” CNRS News.

[97] G. Bhoi, R. Bhavsar, P. Prajapati, and P. Shah, “A review of recent trends on DNA based cryptography,” in *Proceedings of the 3rd International Conference on Intelligent Sustainable Systems, ICISS 2020*, 2020. doi: 10.1109/ICISS49785.2020.9316013.

[98] Boaz Barak, *An Intensive Introduction to Cryptography*. 2021.

[99] M. U. Bokhari, S. Afzal, I. Khan, and M. Z. Khan, “Securing IoT Communications: A Novel Lightweight Stream Cipher Using DNA Cryptography and Grain-80 Cipher,” *SN Comput Sci*, vol. 6, no. 2, p. 88, 2025.

[100] A. Bopalkar, “Sustainable Computation: Harnessing DNA for Solving NP Problems, Secure Cryptography, and High-Density Data Storage,”

Vidhyayana-An International Multidisciplinary Peer-Reviewed E-Journal-ISSN 2454-8596, vol. 10, no. si4, 2025.

[101] K. J. Brakas and M. Alanezi, "A Dynamic DNA Cryptosystem for Secure File Sharing," *Mesopotamian Journal of CyberSecurity*, vol. 5, no. 2, pp. 424–435, 2025.

[102] G. M. Church, Y. Gao, and S. Kosuri, "Next-generation digital information storage in DNA," 2012. doi: 10.1126/science.1226355.

[103] L. D. Dang, D. V. Lu, and N. T. Bac, "Quantum-Enhanced DNA Cryptography: A Revolutionary Intersection of Quantum Computing and Biological Security," in *International conference on WorldS4*, Singapore: Springer Singapore, 2024, pp. 441–456.

[104] A. Das, S. K. Sarma, and S. Deka, "Data security with DNA cryptography," in *Transactions on Engineering Technologies: World Congress on Engineering 2019*, Springer Singapore, 2020, pp. 159–173.

[105] W. A. Dudek, "Parastrophes of quasigroups," *Quasigroups and Related Systems*, vol. 23, no. 2, 2015.

[106] A. Kairi, T. Bhadra, S. K. Pandey, A. Sinha, and A. Nag, "Adaptive DNA Cryptography With Intelligent Machine Learning for Cloud Data Defense," *Engineering Reports*, vol. 7, no. 6, p. e70223, 2025.

[107] S. Kumar, I. Gupta, and A. J. Gupta, "Quantum secure digital signature scheme based on multivariate quadratic quasigroups (MQQ)," *Advances in Mathematics of Communications*, 2024.

[108] C. Lound and R. M. Sirkin, “Statistics for the Social Sciences.,” *J R Stat Soc Ser A Stat Soc*, vol. 159, no. 3, 1996, doi: 10.2307/2983353.

[109] D. Mechkaroska, A. Popovska-Mitrovikj, and V. Bakeva, *Cryptocoding Based on Quasigroups*. Springer, 2024.

[110] P. Mukherjee, C. Pradhan, R. K. Barik, and H. Dubey, “Emerging DNA cryptography-based encryption schemes: a review,” *International Journal of Information and Computer Security*, vol. 20, no. 1–2, 2023, doi: 10.1504/ijics.2023.128000.

[111] A. Parakh, W. Mahoney, L. Gerlock, and M. Battey, “Quasigroup-Based encryption for low-powered devices,” in *Security, Privacy and Reliability in Computer Communications and Networks*, 2017. doi: 10.1201/9781003339410-9.

[112] S. Rawther and S. Sivaji, “Protecting Cloud Computing Environments from Malicious Attacks using Multi-factor Authentication and Modified DNA Cryptography,” *Recent Patents on Engineering*, vol. 19, no. 1, 2025, doi: 10.2174/1872212118666230905141926.

[113] M. V. K. Reddy, R. R. Reddy, E. P. Latha, S. Alamanda, and P. V. S. Srinivas, “Introduction of DNA Computing in Cryptography,” *Artificial Intelligence-Enabled Blockchain Technology and Digital Twin for Smart Hospitals*, pp. 39–60, 2024.

[114] Renda Zhang, “Information Theory Series: 1 — Entropy and Shannon Entropy,” Medium. Accessed: Jul. 15, 2025. [Online]. Available: https://rendazhang.medium.com/information-theory-series-1-entropy-and-shannon-entropy-a20a2101108e

[115] T. A. Taj and M. I. Hossain, “A multi-level random key cryptosystem based on DNA encoding and state-changing mealy machine,” *Journal of Information Security and Applications*, vol. 83, p. 103760, 2024.

[116] X. , W. Xie *et al.*, “DNA computers: advances in storage, cryptography and logic circuits,” *ChemBioChem*, vol. 26, no. 1, p. e202400670, 2025.

[117] M. Ţălu, “DNA-based Cryptography for Internet of Things Security: Concepts, Methods, Applications, and Emerging Trends,” *Buletin Ilmiah Sarjana Teknik Elektro*, vol. 7, no. 2, pp. 68–94, 2025.

APPENDICES

Proposed_Method.py

```
import random
import time
from math import log2

# ----- DNA Bases -----
DNA_BASES = ['A', 'T', 'C', 'G', 'U']

# ----- DNA Encoding/Decoding Rules -----
ENCODING_RULES = {
    "Rule 1": {'00': 'C', '01': 'T', '10': 'A', '11': 'G'},
    "Rule 2": {'00': 'C', '01': 'A', '10': 'T', '11': 'G'},
    "Rule 3": {'00': 'G', '01': 'T', '10': 'A', '11': 'C'},
    "Rule 4": {'00': 'G', '01': 'A', '10': 'T', '11': 'C'},
    "Rule 5": {'00': 'T', '01': 'C', '10': 'G', '11': 'A'},
    "Rule 6": {'00': 'T', '01': 'G', '10': 'C', '11': 'A'},
    "Rule 7": {'00': 'A', '01': 'C', '10': 'G', '11': 'T'},
    "Rule 8": {'00': 'A', '01': 'G', '10': 'C', '11': 'T'}
}
# Compute the inverse for each rule
DECODING_RULES = {
    rule: {v: k for k, v in mapping.items()}
    for rule, mapping in ENCODING_RULES.items()
}

# ----- Conversion Functions -----
def text_to_binary(text):
    #Convert ASCII text to binary string.
    return ''.join(format(ord(ch), '08b') for ch in text)
```

```
def binary_to_text(binary_str):
    #Convert binary string back to ASCII text.
    return ''.join(
        chr(int(binary_str[i:i+8], 2))
        for i in range(0, len(binary_str), 8)
    )

def binary_to_DNA(binary_str, mapping):
    # Map each pair of bits to a DNA base using 'mapping'.
    # Pads with '0' if necessary to complete the last pair.
    if len(binary_str) % 2 != 0:
        binary_str += '0'
    dna = ''
    for i in range(0, len(binary_str), 2):
        dna += mapping[binary_str[i:i+2]]
    return dna

def DNA_to_binary(dna, mapping):
    # Map each DNA base back to its binary string using 'mapping'.
    return ''.join(mapping[base] for base in dna)

def text_to_DNA(text, encoding_mapping):
    # Convert text to binary to DNA (using chosen encoding)
    return binary_to_DNA(text_to_binary(text), encoding_mapping)

def DNA_to_text(dna, decoding_mapping):
    # Convert DNA to binary to text (using chosen decoding)
    return binary_to_text(DNA_to_binary(dna, decoding_mapping))

# ----- Quasigroup Table Generation -----
def generate_latin_square(bases):
```

```
    # Construct a random Quasigroup table of size n×n over 'bases' by backtracking.
    # Ensures each row and column is a permutation of 'bases'.
    n = len(bases)
    square = [[None]*n for _ in range(n)]

    def valid(r, c, v):
        # Check that 'v' does not appear in row r or column c yet
        return (v not in square[r]
                and all(square[i][c] != v for i in range(n)))

    def backtrack(idx=0):
        # Fill cells one by one; if we reach n*n, we're done
        if idx == n*n:
            return True
        r, c = divmod(idx, n)
        # Try each base in random order
        for v in random.sample(bases, n):
            if valid(r, c, v):
                square[r][c] = v
                if backtrack(idx+1):
                    return True
        # Backtrack
        square[r][c] = None
        return False

    if backtrack():
        return square
    else:
        raise ValueError("Failed to generate Latin square")

def convert_square_to_table(square, bases):
```

```
    # Convert a 2D list 'square' into a dict-of-dicts table: table[a][b] = result of
a * b
    return {
        bases[i]: {
            bases[j]: square[i][j]
            for j in range(len(bases))
        }
        for i in range(len(bases))
    }

def generate_random_quasigroup_table(bases):
    # Generate and convert one random quasigroup table.
    square = generate_latin_square(bases)
    return convert_square_to_table(square, bases)

# ----- Parastrophe Table Generation -----
def compute_parastrophes(Q):
    # Given a quasigroup table Q, compute its 5 parastrophe tables.
    bases = list(Q.keys())
    P = [{b: {} for b in bases} for _ in range(5)]

    for x in bases:
        for y in bases:
            # 1) x *_1 y = z  <=>  x * z = y
            z1 = next(z for z in bases if Q[x][z] == y)
            # 2) x *_2 y = z  <=>  z * y = x
            z2 = next(z for z in bases if Q[z][y] == x)
            # 3) x *_3 y = z  <=>  z * x = y
            z3 = next(z for z in bases if Q[z][x] == y)
            # 4) x *_4 y = z  <=>  y * z = x
            z4 = next(z for z in bases if Q[y][z] == x)
            # 5) x *_5 y = z  <=>  y * x = z
```

```
            z5 = Q[y][x]

            # Store in the corresponding table
            P[0][x][y] = z1
            P[1][x][y] = z2
            P[2][x][y] = z3
            P[3][x][y] = z4
            P[4][x][y] = z5

    return P

def generate_random_parastrophe_table(quasigroup_table):
    # Choose one of the five parastrophes at random for Phase II.
    parastrophes = compute_parastrophes(quasigroup_table)
    return random.choice(parastrophes)

def print_table(table, title="Table"):
    print(f"\n--- {title} ---")
    headers = list(table.keys())
    print("    " + " ".join(f"{h:>3}" for h in headers))
    for a in headers:
        row = "".join(f"{table[a][b]:>4}" for b in headers)
        print(f"{a:>3}:{row}")

# ----- Quasigroup Encryption/Decryption -----
def encrypt_phase(dna_input, leader, table):
    prev = leader
    out = ''
    for sym in dna_input:
        c = table[prev][sym]
        out += c
        prev = c
```

```
    return out

def left_divide(a, c, table):
    return next(m for m in table[a] if table[a][m] == c)

def decrypt_phase(dna_cipher, leader, table):
    prev = leader
    out = ''
    for c in dna_cipher:
        m = left_divide(prev, c, table)
        out += m
        prev = c
    return out

# ----- Full Encryption/Decryption Process -----
def encrypt_method(plaintext):
    # 1) Choose a random DNA-encoding rule
    rule = random.choice(list(ENCODING_RULES.keys()))
    enc_map = ENCODING_RULES[rule]
    dec_map = DECODING_RULES[rule]

    # 2) Convert plaintext to DNA string
    dna_plain = text_to_DNA(plaintext, enc_map)

    # 3) Phase I: quasigroup encryption
    leader1 = random.choice(DNA_BASES)
    Q = generate_random_quasigroup_table(DNA_BASES)
    phase1 = encrypt_phase(dna_plain, leader1, Q)

    # 4) Phase II: parastrophe encryption
    leader2 = random.choice(DNA_BASES)
    P = generate_random_parastrophe_table(Q)
```

```
    final = encrypt_phase(phase1, leader2, P)

    # Store all keys needed for decryption
    keys = {
        "leader1": leader1, "Q": Q,
        "leader2": leader2, "P": P,
        "rule": rule, "enc_map": enc_map, "dec_map": dec_map
    }
    return final, keys

def decrypt_method(final_cipher, keys):
    # Reverse Phase II (parastrophe)
    phase1 = decrypt_phase(final_cipher, keys["leader2"], keys["P"])
    # Reverse Phase I (original Q)
    dna_plain = decrypt_phase(phase1, keys["leader1"], keys["Q"])
    # Convert DNA to text
    return DNA_to_text(dna_plain, keys["dec_map"])

# --- Entropy Calculation ---
def shannon_entropy(s):
    from collections import Counter
    total = len(s)
    if total == 0:
        return 0.0
    freqs = Counter(s)
    return -sum((count/total) * log2(count/total) for count in freqs.values())

def compute_entropy_metrics(s, allowed_alphabet=None):
    H = shannon_entropy(s)
    if allowed_alphabet is None:
        allowed_alphabet = set(s)
    max_H = log2(len(allowed_alphabet)) if allowed_alphabet else 0
```

```
    norm_H = H / max_H if max_H > 0 else 0
    return H, max_H, norm_H

# ----- Main Routine -----
if __name__ == "__main__":
    plaintext = input("Enter plaintext: ")

    # Encrypt and measure time
    start = time.perf_counter()
    cipher, keys = encrypt_method(plaintext)
    enc_time = (time.perf_counter() - start)*1000

    print("\nEncryption Complete")
    print("Cipher DNA:", cipher)
    print("Leader1:", keys["leader1"],
          "Leader2:", keys["leader2"],
          "Encoding Rule:", keys["rule"])
    print_table(keys["Q"], "Quasigroup Table")
    print_table(keys["P"], "Parastrophe Table")
    print(f"Encryption time: {enc_time:.4f}ms")

    # Decrypt and measure time
    start = time.perf_counter()
    decrypted = decrypt_method(cipher, keys)
    dec_time = (time.perf_counter() - start)*1000

    print("\nDecryption Complete")
    print("Decrypted Text:", decrypted)
    print(f"Decryption time: {dec_time:.4f}ms")

    # Entropy metrics
    pt_H, pt_maxH, pt_norm = compute_entropy_metrics(plaintext)
```

```
    ct_H, ct_maxH, ct_norm = compute_entropy_metrics(cipher,
allowed_alphabet=DNA_BASES)


    print("\n--- Entropy Metrics ---")
    print("Plaintext Shannon Entropy: {:.6f}".format(pt_H))
    print("Plaintext Max Entropy: {:.6f}".format(pt_maxH))
    print("Plaintext Normalized Entropy: {:.6f}".format(pt_norm))

    print("Ciphertext Shannon Entropy: {:.6f}".format(ct_H))
    print("Ciphertext Max Entropy: {:.6f}".format(ct_maxH))
    print("Ciphertext Normalized Entropy: {:.6f}".format(ct_norm))
```

Markovski_Method.py

```
import random
import time
from math import log2

# ---- GF(2) & Matrix Utilities ----

def bits_to_vec(x):
    return [(x >> 1) & 1, x & 1]

def vec_to_bits(v):
    return (v[0] << 1) | v[1]

def random_inv_2x2():
    while True:
        M = [[random.randint(0,1) for _ in range(2)] for __ in range(2)]
        if (M[0][0]*M[1][1] ^ M[0][1]*M[1][0]) == 1:
            return M

# ---- Matrix-Based Quasigroup Class (Used only for table generation) ----

class MGQuasigroup:
    def __init__(self, m, A, B, C=None):
        self.m = m  # 2-bit constant
        self.A = A  # 2x2 GF(2) matrix
        self.B = B  # 2x2 GF(2) matrix
        self.C = C or [[1,1],[1,1]]
    def mul(self, x, y):
        xv, yv = bits_to_vec(x), bits_to_vec(y)
        Ax = [self.A[i][0]*xv[0] ^ self.A[i][1]*xv[1] for i in (0,1)]
        By = [self.B[i][0]*yv[0] ^ self.B[i][1]*yv[1] for i in (0,1)]
```

```
        CAx = [self.C[i][0]*Ax[0] ^ self.C[i][1]*Ax[1] for i in (0,1)]
        CBy = [self.C[i][0]*By[0] ^ self.C[i][1]*By[1] for i in (0,1)]
        dot = (CAx[0] & CBy[0]) ^ (CAx[1] & CBy[1])
        res = [self.m[i] ^ Ax[i] ^ By[i] for i in (0,1)]
        res[1] ^= dot
        return vec_to_bits(res)

# ---- Table Generation (Forward & Inverse) ----

def make_forward_table(Q: MGQuasigroup):
    table = [[0]*4 for _ in range(4)]
    for x in range(4):
        for y in range(4):
            table[x][y] = Q.mul(x, y)
    return table

def make_inv_table(Q: MGQuasigroup):
    inv = [[None]*4 for _ in range(4)]
    for x in range(4):
        for y in range(4):
            z = Q.mul(x, y)
            inv[x][z] = y
    return inv

# ---- Public Quasigroup Tables Generator ----

def generate_public_quasigroups(n=128):
    QT_forward, QT_inverse = [], []
    for _ in range(n):
        m = [random.randint(0,1) for _ in range(2)]
        A, B = random_inv_2x2(), random_inv_2x2()
        Q = MGQuasigroup(m, A, B)
```

```
        QT_forward.append(make_forward_table(Q))
        QT_inverse.append(make_inv_table(Q))
    return QT_forward, QT_inverse

# ---- Transformations (e/d) with Table Lookups ----

def e_transform(seq, leader, QT):
    b = [QT[leader][seq[0]]]
    for i in range(1, len(seq)):
        b.append(QT[b[i-1]][seq[i]])
    return b

def d_transform(seq, leader, QT_inv):
    a = [QT_inv[leader][seq[0]]]
    for i in range(1, len(seq)):
        a.append(QT_inv[seq[i-1]][seq[i]])
    return a

# ----- Utility Functions for Bit/Element Conversions -----
def bytes_to_elements(block_bytes):
    bit_str = ''.join(format(b, '08b') for b in block_bytes)
    if len(bit_str) % 2 != 0:
        bit_str += '0'
    return [int(bit_str[i:i+2], 2) for i in range(0, len(bit_str), 2)]

def elements_to_bytes(elements):
    bit_str = ''.join(format(e, '02b') for e in elements)
    b = bytearray()
    for i in range(0, len(bit_str), 8):
        b.append(int(bit_str[i:i+8], 2))
    return bytes(b)
```

```
# ----- Key Schedule Extraction -----
def extract_key_parts(key_bytes):
    if len(key_bytes) != 16:
        raise ValueError("Key must be 16 bytes (128 bits).")
    key_bits = ''.join(format(b, '08b') for b in key_bytes)
    leaders = [int(key_bits[i*2:i*2+2], 2) for i in range(8)]
    start = 16
    Q_indices = [int(key_bits[start + i*7 : start + i*7 + 7], 2) for i in range(8)]
    start += 56
    T_indices = [int(key_bits[start + i*7 : start + i*7 + 7], 2) for i in range(8)]
    return leaders, Q_indices, T_indices

# ----- Padding Functions -----
BLOCK_BYTE_SIZE = 8  # 64 bits = 8 bytes

def pad(plaintext):
    pad_len = BLOCK_BYTE_SIZE - (len(plaintext) %
BLOCK_BYTE_SIZE)
    return plaintext + bytes([pad_len] * pad_len)

def unpad(padded):
    pad_len = padded[-1]
    return padded[:-pad_len]

# ----- Encryption/Decryption per Block (Optimized with Tables) -----
def encrypt_block(block_bytes, leaders, Q_indices, T_indices,
public_QT_forward, public_QT_inverse):
    elems = bytes_to_elements(block_bytes)

    # Step 1: Process mini-blocks
    mini_blocks = [elems[i*4:(i+1)*4] for i in range(8)]
    for i in range(8):
        Q_index = Q_indices[i] % len(public_QT_forward)
```

```
        QT = public_QT_forward[Q_index]
        l = leaders[i]
        mini_blocks[i] = e_transform(mini_blocks[i], l, QT)
    X = []
    for mini in mini_blocks:
        X.extend(mini)

    # Step 2: Full block transformation
    for i in range(8):
        T_index = T_indices[i] % len(public_QT_forward)
        QT = public_QT_forward[T_index]
        l = leaders[i]
        if i % 2 == 0:
            X = e_transform(X, l, QT)
        else:
            X_rev = list(reversed(X))
            X_rev = e_transform(X_rev, l, QT)
            X = list(reversed(X_rev))
    return elements_to_bytes(X)

def decrypt_block(block_bytes, leaders, Q_indices, T_indices,
public_QT_forward, public_QT_inverse):
    X = bytes_to_elements(block_bytes)

    # Reverse Step 2
    for i in reversed(range(8)):
        T_index = T_indices[i] % len(public_QT_inverse)
        T_inv = public_QT_inverse[T_index]
        l = leaders[i]
        if i % 2 == 0:
            X = d_transform(X, l, T_inv)
        else:
```

```
            X_rev = list(reversed(X))
            X_rev = d_transform(X_rev, l, T_inv)
            X = list(reversed(X_rev))

    # Reverse Step 1
    mini_blocks = [X[i*4:(i+1)*4] for i in range(8)]
    for i in range(8):
        Q_index = Q_indices[i] % len(public_QT_inverse)
        Q_inv = public_QT_inverse[Q_index]
        l = leaders[i]
        mini_blocks[i] = d_transform(mini_blocks[i], l, Q_inv)
    elems = []
    for mini in mini_blocks:
        elems.extend(mini)
    return elements_to_bytes(elems)


# ----- Full Message Encryption/Decryption -----
def encrypt_message(plaintext, key_bytes, public_QT_forward,
public_QT_inverse):
    padded = pad(plaintext)
    ciphertext = bytearray()
    leaders, Q_indices, T_indices = extract_key_parts(key_bytes)
    for i in range(0, len(padded), BLOCK_BYTE_SIZE):
        block = padded[i:i+BLOCK_BYTE_SIZE]
        cipher_block = encrypt_block(block, leaders, Q_indices, T_indices,
public_QT_forward, public_QT_inverse)
        ciphertext.extend(cipher_block)
    return bytes(ciphertext)


def decrypt_message(ciphertext, key_bytes, public_QT_forward,
public_QT_inverse):
    plaintext = bytearray()
    leaders, Q_indices, T_indices = extract_key_parts(key_bytes)
```

```
    for i in range(0, len(ciphertext), BLOCK_BYTE_SIZE):
        block = ciphertext[i:i+BLOCK_BYTE_SIZE]
        plain_block = decrypt_block(block, leaders, Q_indices, T_indices,
public_QT_forward, public_QT_inverse)
        plaintext.extend(plain_block)
    return unpad(plaintext)


# --- Entropy Functions ---
def shannon_entropy(data):
    # Compute Shannon entropy (in bits per symbol) for data.
    freqs = {}
    for symbol in data:
        freqs[symbol] = freqs.get(symbol, 0) + 1
    total = len(data)
    H = 0.0
    for count in freqs.values():
        p = count / total
        H -= p * log2(p)
    return H


def compute_entropy_metrics(data, allowed_alphabet=None):
    H = shannon_entropy(data)
    if allowed_alphabet is None:
        allowed_alphabet = set(data)
    max_H = log2(len(allowed_alphabet)) if allowed_alphabet else 0
    norm_H = H / max_H if max_H > 0 else 0
    return H, max_H, norm_H


# ----- Main Routine -----
def main():
    user_text = input("Enter plaintext: ")
    plaintext = user_text.encode('utf-8')
```

```
    key_bytes = bytes(random.getrandbits(8) for _ in range(16))
    public_QT_forward, public_QT_inverse =
generate_public_quasigroups(128)


    start_enc = time.perf_counter()
    ciphertext = encrypt_message(plaintext, key_bytes, public_QT_forward,
public_QT_inverse)
    end_enc = time.perf_counter()
    enc_time = (end_enc - start_enc) * 1000


    start_dec = time.perf_counter()
    decrypted = decrypt_message(ciphertext, key_bytes, public_QT_forward,
public_QT_inverse)
    end_dec = time.perf_counter()
    dec_time = (end_dec - start_dec) * 1000


    # Compute entropy metrics for plaintext
    pt_H, pt_max_H, pt_norm_H = compute_entropy_metrics(plaintext,
allowed_alphabet=set(plaintext))


    # Compute entropy metrics for ciphertext
    ct_H, ct_max_H, ct_norm_H = compute_entropy_metrics(ciphertext,
allowed_alphabet=set(ciphertext))




    print("\nCiphertext (hex):")
    print(ciphertext.hex())
    try:
        decrypted_text = decrypted.decode('utf-8')
    except UnicodeDecodeError:
        decrypted_text = str(decrypted)
    print("\nDecrypted Plaintext:")
    print(decrypted_text)
```

```
    print("\nEncryption Time: {:.6f} ms".format(enc_time))
    print("Decryption Time: {:.6f} ms".format(dec_time))

    if decrypted == plaintext:
        print("\nSuccess: Decrypted text matches original plaintext.")
    else:
        print("\nError: Decrypted text does not match original plaintext.")



    print("\n--- Entropy Metrics ---")
    print("Plaintext Shannon Entropy: {:.6f}".format(pt_H))
    print("Plaintext Max Entropy: {:.6f}".format(pt_max_H))
    print("Plaintext Normalized Entropy: {:.6f}".format(pt_norm_H))

    print("Ciphertext Shannon Entropy: {:.6f}".format(ct_H))
    print("Ciphertext Max Entropy: {:.6f}".format(ct_max_H))
    print("Ciphertext Normalized Entropy: {:.6f}".format(ct_norm_H))



if __name__ == '__main__':
    main()
```

Padmapriya_Method.py

```
import heapq
import collections
import time
import random
from datetime import datetime

# DNA <-> Binary mapping
BASES = ['A', 'C', 'G', 'T']
BASE_TO_BIN = {'A': [0, 0], 'C': [0, 1], 'G': [1, 0], 'T': [1, 1]}
BIN_TO_BASE = {'00': 'A', '01': 'C', '10': 'G', '11': 'T'}

# Huffman Tree
class Node:
    def __init__(self, freq, symbol, left=None, right=None):
        self.freq = freq
        self.symbol = symbol
        self.left = left
        self.right = right
    def __lt__(self, other): return self.freq < other.freq
def build_huffman_tree(freq):
    heap = [Node(freq[s], s) for s in freq if freq[s] > 0]
    heapq.heapify(heap)
    while len(heap) > 1:
        left = heapq.heappop(heap)
        right = heapq.heappop(heap)
        merged = Node(left.freq + right.freq, None, left, right)
        heapq.heappush(heap, merged)
    return heap[0]
def get_huffman_codes(root):
    codes = {}
```

```
    def traverse(node, cur=''):
        if node.symbol is not None:
            codes[node.symbol] = cur or '0'
            return
        if node.left: traverse(node.left, cur + '0')
        if node.right: traverse(node.right, cur + '1')
    traverse(root)
    return codes

# Key generation
def generate_dna_key(bit_length):
    bases_needed = (bit_length + 1) // 2
    return ''.join(random.choice(BASES) for _ in range(bases_needed))

# File I/O helpers
def write_key_to_file(dna_key, filename="dna_key.txt"):
    with open(filename, "w", encoding="utf-8") as f:
        f.write(dna_key)
def read_key_from_file(filename="dna_key.txt"):
    with open(filename, "r", encoding="utf-8") as f:
        return f.read().strip()
def log(message, log_file="encryption_log.txt"):
    timestamp = datetime.now().strftime("%Y-%m-%d %H:%M:%S.%f")[:-3]
    line = f"[{timestamp}] {message}"
    print(line)
    with open(log_file, "a", encoding="utf-8") as f:
        f.write(line + "\n")

# ENCRYPTION
def encrypt(plaintext):
    start_total = time.perf_counter()
    # 1. plaintext → binary
```

```
    bin_plain = [int(b) for c in plaintext for b in f'{ord(c):08b}']
    bit_len = len(bin_plain)
    # 2. DNA key
    dna_key = generate_dna_key(bit_len)
    bin_key = [bit for b in dna_key for bit in BASE_TO_BIN[b]]
    if len(bin_key) < bit_len:
        bin_key += [0] * (bit_len - len(bin_key))
    # 3. XOR (DNA OTP)
    bin_cipher = [a ^ b for a, b in zip(bin_plain, bin_key)]
    # 4. binary → DNA
    dna_cipher = ''.join(BIN_TO_BASE[''.join(map(str, bin_cipher[i:i+2]))]
                         for i in range(0, len(bin_cipher), 2))
    # 5. Huffman compression
    freq = collections.Counter(dna_cipher)
    root = build_huffman_tree(freq)
    codes = get_huffman_codes(root)
    encoded = ''.join(codes[b] for b in dna_cipher)
    # ---- file writes ----
    write_key_to_file(dna_key)
    with open("ciphertext.bin", "w", encoding="utf-8") as f:
        f.write(encoded)

    total_ms = (time.perf_counter() - start_total) * 1000
    return encoded, root, dna_key, total_ms

# DECRYPTION
def decrypt(encoded, root, dna_key, expected_bit_len):
    start_total = time.perf_counter()
    # 1. Huffman → DNA (Z)
    Z = []
    x = root
    for bit in encoded:
```

```
        if bit == '0':
            if x.left:
                x = x.left
            else:
                raise ValueError("Invalid bit '0' - no left child")
        else:
            if x.right:
                x = x.right
            else:
                raise ValueError("Invalid bit '1' - no right child")
        if x.left is None and x.right is None:
            Z.append(x.symbol)
            x = root
    # 2. DNA → binary (C as list of 0/1)
    C = []
    for Zi in Z:
        if Zi == 'A':
            C.append(0)
            C.append(0)
        elif Zi == 'C':  # Fixed paper typo: use elif to avoid extra appends
            C.append(0)
            C.append(1)
        elif Zi == 'G':
            C.append(1)
            C.append(0)
        else:  # 'T'
            C.append(1)
            C.append(1)
    # 3. key → binary
    bin_key = [bit for b in dna_key for bit in BASE_TO_BIN[b]]
    bin_key = bin_key + [0] * (expected_bit_len - len(bin_key)) if len(bin_key)
< expected_bit_len else bin_key[:expected_bit_len]
```

```
    # 4. XOR back (M)
    M = [Ci ^ Keyi for Ci, Keyi in zip(C, bin_key)]
    # 5. binary → text (Convert M to ASCII)
    plaintext = ''
    for i in range(0, len(M), 8):
        chunk = M[i:i+8]
        byte_val = sum(bit * (1 << (7 - j)) for j, bit in enumerate(chunk))
        plaintext += chr(byte_val)
    # ---- log ----
    log(f'DECRYPT: Success -> '{plaintext}'')
    total_ms = (time.perf_counter() - start_total) * 1000
    return plaintext, total_ms
# ------------------------------------------------------------
# MAIN
def main():
        print("\n" + "-"*60)
        plaintext = input("Enter plaintext: ").strip()
        print(f"Encrypting: \"{plaintext}\" ({len(plaintext)} chars)")
        # ---- ENCRYPT ----
        encoded, root, key, enc_ms = encrypt(plaintext)
        # ---- DECRYPT ----
        decrypted, dec_ms = decrypt(encoded, root, key, len(plaintext)*8)
        # ---- RESULTS ----
        print("\n" + "="*62)
        print(" RESULT")
        print("="*62)
        print(f"Encryption Time : {enc_ms:8.3f} ms")
        print(f"Decryption Time : {dec_ms:8.3f} ms")
        print("="*62)
        print("\nFiles created: dna_key.txt, ciphertext.bin, encryption_log.txt")
if __name__ == "__main__":
    main()
```

Quasigroup_Size_Phase_Number.py

```
import random
import time
import matplotlib.pyplot as plt
import string
import math
from collections import Counter

# ----- DNA Encoding/Decoding Rules -----
ENCODING_RULES = {
    "Rule 1": {'00': 'C', '01': 'T', '10': 'A', '11': 'G'},
    "Rule 2": {'00': 'C', '01': 'A', '10': 'T', '11': 'G'},
    "Rule 3": {'00': 'G', '01': 'T', '10': 'A', '11': 'C'},
    "Rule 4": {'00': 'G', '01': 'A', '10': 'T', '11': 'C'},
    "Rule 5": {'00': 'T', '01': 'C', '10': 'G', '11': 'A'},
    "Rule 6": {'00': 'T', '01': 'G', '10': 'C', '11': 'A'},
    "Rule 7": {'00': 'A', '01': 'C', '10': 'G', '11': 'T'},
    "Rule 8": {'00': 'A', '01': 'G', '10': 'C', '11': 'T'}
}
DECODING_RULES = { rule: {v: k for k, v in mapping.items()}
                   for rule, mapping in ENCODING_RULES.items() }

# ----- Conversion Functions -----
def text_to_binary(text):
    return ''.join(format(ord(ch), '08b') for ch in text)

def binary_to_text(binary_str):
    return ''.join(chr(int(binary_str[i:i+8], 2)) for i in range(0, len(binary_str),
8))

def binary_to_DNA(binary_str, mapping):
    if len(binary_str) % 2 != 0:
```

```
        binary_str += '0'
    dna = ''
    for i in range(0, len(binary_str), 2):
        dna += mapping[binary_str[i:i+2]]
    return dna

def DNA_to_binary(dna, mapping):
    return ''.join(mapping[base] for base in dna)

def text_to_DNA(text, encoding_mapping):
    return binary_to_DNA(text_to_binary(text), encoding_mapping)

def DNA_to_text(dna, decoding_mapping):
    return binary_to_text(DNA_to_binary(dna, decoding_mapping))

# ----- Latin Square Generation -----
def generate_latin_square(bases):
    n = len(bases)
    square = [[None] * n for _ in range(n)]

    def is_valid(row, col, value):
        for j in range(n):
            if square[row][j] == value:
                return False
        for i in range(n):
            if square[i][col] == value:
                return False
        return True

    def backtrack(cell=0):
        if cell == n * n:
            return True
```

```
        row, col = divmod(cell, n)
        for value in random.sample(bases, len(bases)):
            if is_valid(row, col, value):
                square[row][col] = value
                if backtrack(cell + 1):
                    return True
                square[row][col] = None
        return False

    if backtrack():
        return square
    else:
        raise ValueError("Failed to generate Latin square.")

def convert_square_to_table(square, bases):
    table = {}
    for i, row_label in enumerate(bases):
        table[row_label] = {}
        for j, col_label in enumerate(bases):
            table[row_label][col_label] = square[i][j]
    return table

def generate_random_quasigroup_table(bases):
    return convert_square_to_table(generate_latin_square(bases), bases)

# ----- Parastrophe Table Generation -----
def compute_parastrophes(Q):
    bases = list(Q.keys())
    P = [{x: {} for x in bases} for _ in range(5)]

    for x in bases:
        for y in bases:
```

```
            # 1) x *_1 y = z  <=>  x * z = y
            z1 = next(z for z in bases if Q[x][z] == y)
            # 2) x *_2 y = z  <=>  z * y = x
            z2 = next(z for z in bases if Q[z][y] == x)
            # 3) x *_3 y = z  <=>  z * x = y
            z3 = next(z for z in bases if Q[z][x] == y)
            # 4) x *_4 y = z  <=>  y * z = x
            z4 = next(z for z in bases if Q[y][z] == x)
            # 5) x *_5 y = z  <=>  y * x = z
            z5 = Q[y][x]

            P[0][x][y] = z1
            P[1][x][y] = z2
            P[2][x][y] = z3
            P[3][x][y] = z4
            P[4][x][y] = z5

    return P

def generate_random_parastrophe_table(Q):
    # Pick one of the parastrophes at random.
    parastrophes = compute_parastrophes(Q)
    return random.choice(parastrophes)

# ----- Encryption/Decryption Functions -----
def encrypt_phase(dna_input, leader, table):
    result = ""
    prev = leader
    for symbol in dna_input:
        c = table[prev][symbol]
        result += c
        prev = c
```

```
    return result

def left_divide(a, c, table):
    for m in table[a]:
        if table[a][m] == c:
            return m
    raise ValueError("No valid division found.")

def decrypt_phase(dna_cipher, leader, table):
    result = ""
    prev = leader
    for c in dna_cipher:
        m = left_divide(prev, c, table)
        result += m
        prev = c
    return result

# ----- Normalized Shannon Entropy -----
def normalized_shannon_entropy(sequence, alphabet):
    """
    Returns normalized Shannon entropy in [0,1] for `sequence` over
`alphabet`.
    Normalization is by log2(|alphabet|).
    """
    if len(sequence) == 0:
        return 0.0
    counts = Counter(sequence)
    total = sum(counts[a] for a in alphabet if a in counts)
    # If total==0 (none of alphabet chars present), return 0
    if total == 0:
        return 0.0
    H = 0.0
```

```
    for a in alphabet:
        p = counts.get(a, 0) / total
        if p > 0:
            H -= p * math.log2(p)
    max_H = math.log2(len(alphabet)) if len(alphabet) > 1 else 1.0
    return H / max_H


# ----- Encrypt/Decrypt Methods with Even Phases -----
def encrypt_method(plaintext, dna_bases, num_phases=2):
    # Encrypts the plaintext using an even number of phases.
    # Each pair of phases starts with encryption using a random quasigroup table, and ends with encryption using the corresponding parastrophe table.
    if num_phases % 2 != 0:
        raise ValueError("Number of phases must be an even number.")

    chosen_rule = random.choice(list(ENCODING_RULES.keys()))
    encoding_mapping = ENCODING_RULES[chosen_rule]
    dna_plaintext = text_to_DNA(plaintext, encoding_mapping)

    phase_leaders = []
    phase_tables = []
    current_cipher = dna_plaintext

    for _ in range(num_phases // 2):
        # Phase 1: encryption using a random quasigroup table.
        leader1 = random.choice(dna_bases)
        table = generate_random_quasigroup_table(dna_bases)
        current_cipher = encrypt_phase(current_cipher, leader1, table)
        phase_leaders.append(leader1)
        phase_tables.append(table)

        # Phase 2: encryption using the parastrophe of the quasigroup table.
```

```
        leader2 = random.choice(dna_bases)
        parastrophe_table = generate_random_parastrophe_table(table)
        current_cipher = encrypt_phase(current_cipher, leader2,
parastrophe_table)
        phase_leaders.append(leader2)
        phase_tables.append(parastrophe_table)

    keys = {
        "dna_bases": dna_bases,
        "phase_leaders": phase_leaders,
        "phase_tables": phase_tables,
        "encoding_rule": chosen_rule,
        "decoding_mapping": DECODING_RULES[chosen_rule]
    }
    return current_cipher, keys

def decrypt_method(final_cipher, keys):
    current_dna = final_cipher
    for leader, table in zip(reversed(keys["phase_leaders"]),
reversed(keys["phase_tables"])):
        current_dna = decrypt_phase(current_dna, leader, table)
    return DNA_to_text(current_dna, keys["decoding_mapping"])

# ----- Main Routine -----
if __name__ == "__main__":
    user_plaintext = input("Enter your plaintext: ")

    def measure_quasigroup_performance_custom(plaintext):
        sizes = list(range(4, 16))  # Quasigroup sizes from 4 to 15
        num_runs = 5
        results = {}

        for size in sizes:
```

```
        if size == 4:
            dna_bases = ['A', 'T', 'C', 'G']
        else:
            extras = [ch for ch in string.ascii_uppercase if ch not in ['A', 'T',
'C', 'G']]
            dna_bases = ['A', 'T', 'C', 'G'] + extras[:size-4]

        enc_times = []
        dec_times = []
        entropies_cipher = []
        entropies_plain = []

        for _ in range(num_runs):
            start = time.perf_counter()
            cipher, keys = encrypt_method(plaintext, dna_bases, 2)
            enc_time = (time.perf_counter() - start) * 1000  # milliseconds

            # compute entropies (normalized)
            h_plain = normalized_shannon_entropy(plaintext, dna_bases)
            h_cipher = normalized_shannon_entropy(cipher, dna_bases)

            start = time.perf_counter()
            decrypt_method(cipher, keys)
            dec_time = (time.perf_counter() - start) * 1000  # milliseconds

            enc_times.append(enc_time)
            dec_times.append(dec_time)
            entropies_plain.append(h_plain)
            entropies_cipher.append(h_cipher)

        results[size] = {
            'enc_avg': sum(enc_times) / num_runs,
```

```
            'dec_avg': sum(dec_times) / num_runs,
            'plain_entropy_avg': sum(entropies_plain) / num_runs,
            'cipher_entropy_avg': sum(entropies_cipher) / num_runs
        }
        # Display average times only
        print(f"Quasigroup Size {size}: Encryption Avg = {results[size]['enc_avg']:.6f} ms, "
              f"Decryption Avg = {results[size]['dec_avg']:.6f} ms"
              f"Normalised Shannon Entropy Plaintext = {results[size]['plain_entropy_avg']:.4f}, "
              f"Normalised Shannon Entropy Ciphertext = {results[size]['cipher_entropy_avg']:.4f}")

    # Plotting
    sizes_list = list(results.keys())
    enc_avgs = [results[s]['enc_avg'] for s in sizes_list]
    dec_avgs = [results[s]['dec_avg'] for s in sizes_list]
    cipher_ent_avgs = [results[s]['cipher_entropy_avg'] for s in sizes_list]
    plain_ent_avgs = [results[s]['plain_entropy_avg'] for s in sizes_list]

    plt.figure(figsize=(10, 6))
    plt.plot(sizes_list, enc_avgs, marker='o', label='Encryption Time')
    plt.plot(sizes_list, dec_avgs, marker='s', label='Decryption Time')
    plt.xlabel('Quasigroup Size')
    plt.ylabel('Average Time (milliseconds)')
    plt.title(f'Performance vs Quasigroup Size')
    plt.legend()
    plt.grid(True)
    plt.show()

    plt.figure(figsize=(10, 6))
    plt.plot(sizes_list, cipher_ent_avgs, marker='o', label='Normalised Shannon Entropy Ciphertext')
```

```
        plt.plot(sizes_list, plain_ent_avgs, marker='s', label='Normalised Shannon Entropy Plaintext')
        plt.xlabel('Quasigroup Size')
        plt.ylabel('Normalized Shannon Entropy')
        plt.title('Normalised Shannon Entropy of Plaintext and Ciphertext vs Quasigroup Size')
        plt.legend()
        plt.grid(True)
        plt.show()

        return results

    def measure_phases_performance_custom(plaintext):
        # Use even number of phases only: 2, 4, 6, 8, 10
        phases = list(range(2, 11, 2))
        num_runs = 5
        results = {}
        dna_bases = ['A', 'T', 'C', 'G', 'U']

        for num_phase in phases:
            enc_times = []
            dec_times = []
            entropies_cipher = []
            entropies_plain = []

            for _ in range(num_runs):
                start = time.perf_counter()
                cipher, keys = encrypt_method(plaintext, dna_bases, num_phase)
                enc_time = (time.perf_counter() - start) * 1000  # milliseconds

                start = time.perf_counter()
                decrypt_method(cipher, keys)
                dec_time = (time.perf_counter() - start) * 1000  # milliseconds
```

```
            # entropy calculation (normalized)
            h_plain = normalized_shannon_entropy(plaintext, dna_bases)
            h_cipher = normalized_shannon_entropy(cipher, dna_bases)

            enc_times.append(enc_time)
            dec_times.append(dec_time)
            entropies_plain.append(h_plain)
            entropies_cipher.append(h_cipher)

        results[num_phase] = {
            'enc_avg': sum(enc_times) / num_runs,
            'dec_avg': sum(dec_times) / num_runs,
            'plain_entropy_avg': sum(entropies_plain) / num_runs,
            'cipher_entropy_avg': sum(entropies_cipher) / num_runs
        }
        # Display average times only
        print(f"Phases {num_phase}: Encryption Avg =
{results[num_phase]['enc_avg']:.6f} ms, "
              f"Decryption Avg = {results[num_phase]['dec_avg']:.6f} ms"
              f"Normalised Shannon Entropy of Plaintext =
{results[num_phase]['plain_entropy_avg']:.4f}, "
              f"Normalised Shannon Entropy of Ciphertext =
{results[num_phase]['cipher_entropy_avg']:.4f}")

    # Plotting
    phase_list = list(results.keys())
    enc_avgs = [results[p]['enc_avg'] for p in phase_list]
    dec_avgs = [results[p]['dec_avg'] for p in phase_list]
    cipher_ent_avgs = [results[p]['cipher_entropy_avg'] for p in phase_list]
    plain_ent_avgs = [results[p]['plain_entropy_avg'] for p in phase_list]

    plt.figure(figsize=(10, 6))
```

```
        plt.plot(phase_list, enc_avgs, marker='o', label='Encryption Time')
        plt.plot(phase_list, dec_avgs, marker='s', label='Decryption Time')
        plt.xlabel('Number of Phases (Even numbers only)')
        plt.ylabel('Average Time (milliseconds)')
        plt.title(f'Performance vs Number of Phases')
        plt.legend()
        plt.grid(True)
        plt.show()

        plt.figure(figsize=(10, 6))
        plt.plot(phase_list, cipher_ent_avgs, marker='o', label='Normalised
Shannon Entropy of Ciphertext')
        plt.plot(phase_list, plain_ent_avgs, marker='s', label='Normalised
Shannon Entropy of Plaintext')
        plt.xlabel('Number of Phases')
        plt.ylabel('Normalized Shannon Entropy')
        plt.title('Normalised Shannon Entropy of Plaintext and Ciphertext vs
Number of Phases')
        plt.legend()
        plt.grid(True)
        plt.show()

        return results

    print("\n=== Measuring Performance vs Quasigroup Size ===")
    qs_results = measure_quasigroup_performance_custom(user_plaintext)

    print("\n=== Measuring Performance vs Number of Phases ===")
    phase_results = measure_phases_performance_custom(user_plaintext)
```

Plot_Graph.py

```
import matplotlib.pyplot as plt


# Graph of Normalised Shannon’s Entropy of Plaintext and Ciphertext of the Proposed Method against Plaintext Length

# Data from Tables 5.2 and Table 5.3

plaintext_lengths = [500, 1000, 1500, 2000, 2500, 3000]

normalised_entropy_plaintext = [0.845886, 0.823226, 0.793238, 0.786977, 0.785779, 0.771054]

normalised_entropy_ciphertext = [0.999544, 0.999564, 0.999723, 0.999796, 0.999821, 0.999975]


plt.figure(figsize=(10, 5))

plt.plot(plaintext_lengths, normalised_entropy_plaintext, marker='o', label="Normalized Shannon's Entropy (Plaintext)")

plt.plot(plaintext_lengths, normalised_entropy_ciphertext, marker='x', label="Normalized Shannon's Entropy (Ciphertext)")


plt.title("Plaintext and Ciphertext Normalised Shannon's Entropy vs Plaintext Length")

plt.xlabel("Plaintext Length and Ciphertext Length")

plt.ylabel("Normalised Shannon's Entropy of Plaintext and Ciphertext")

plt.legend()

plt.grid(True)

plt.tight_layout()

plt.show()


# Graph of encryption time for the three methods against plaintext length

# Data from Table 5.5

plaintext_lengths = [10, 20, 40, 80, 100, 500]

padmapriya_times = [2, 4, 3, 5, 5, 8]

markovski_times = [0.2402, 0.2899, 0.5242, 0.8386, 1.4333, 4.4841]

proposed_times = [0.4479, 0.5256, 0.5820, 0.6747, 0.7610, 2.5488]
```

```
plt.figure(figsize=(8, 5))
plt.plot(plaintext_lengths, padmapriya_times, marker='o', label='Padmapriya')
plt.plot(plaintext_lengths, markovski_times, marker='s', label='Markovski')
plt.plot(plaintext_lengths, proposed_times, marker='^', label='Proposed')

plt.title('Encryption Time vs. Plaintext Length')
plt.xlabel('Plaintext Length')
plt.ylabel('Encryption Time (ms)')
plt.grid(True)
plt.legend()
plt.tight_layout()
plt.show()

# Graph of decryption time for the three methods against plaintext length
# Data from Table 5.6
plaintext_lengths = [10, 20, 40, 80, 100, 500]
padmapriya_times = [2, 2, 3, 3, 3, 7]
markovski_times = [0.1739, 0.2510, 0.4756, 0.8060, 1.3891, 7.6478]
proposed_times = [0.1200, 0.2619, 0.4291, 0.8275, 1.0465, 5.3951]

plt.figure(figsize=(8, 5))
plt.plot(plaintext_lengths, padmapriya_times, marker='o', label='Padmapriya')
plt.plot(plaintext_lengths, markovski_times, marker='s', label='Markovski')
plt.plot(plaintext_lengths, proposed_times, marker='^', label='Proposed')

plt.title('Decryption Time vs. Plaintext Length')
plt.xlabel('Plaintext Length')
plt.ylabel('Decryption Time (ms)')
plt.grid(True)
plt.legend()
plt.tight_layout()
```

```
plt.show()


# Graph of Normalised Shannon’s entropy of all three methods against plaintext length
# Data from Table 5.7
plaintext_lengths = [500, 1000, 1500, 2000, 2500, 3000]
padmapriya = [0.860536, 0.864609, 0.861672, 0.858049, 0.858435, 0.863743]
markovski = [0.972287, 0.979534, 0.982073, 0.987947, 0.988624, 0.991953]
proposed = [0.999544, 0.999564, 0.999723, 0.999796, 0.999821, 0.999975]

plt.figure(figsize=(10, 6))

plt.plot(plaintext_lengths, padmapriya, marker='o', label='Padmapriya', linestyle='-')
plt.plot(plaintext_lengths, markovski, marker='s', label='Markovski', linestyle='-')
plt.plot(plaintext_lengths, proposed, marker='^', label='Proposed', linestyle='-')

plt.title("Normalised Shannon Entropy vs. Plaintext Length", fontsize=14)
plt.xlabel("Plaintext Length", fontsize=12)
plt.ylabel("Normalised Shannon Entropy", fontsize=12)
plt.xticks(plaintext_lengths)
plt.ylim(0.80, 1.005)
plt.grid(True, linestyle='--', alpha=0.6)

plt.legend(loc='lower left', fontsize=10)

plt.tight_layout()
plt.show()
```

Chi_Square_Test.py

```
import random
import matplotlib.pyplot as plt

# DNA Bases and Encoding Rules
DNA_BASES = ['A', 'T', 'C', 'G', 'U']

ENCODING_RULES = {
    "Rule 1": {'00': 'C', '01': 'T', '10': 'A', '11': 'G'},
    "Rule 2": {'00': 'C', '01': 'A', '10': 'T', '11': 'G'},
    "Rule 3": {'00': 'G', '01': 'T', '10': 'A', '11': 'C'},
    "Rule 4": {'00': 'G', '01': 'A', '10': 'T', '11': 'C'},
    "Rule 5": {'00': 'T', '01': 'C', '10': 'G', '11': 'A'},
    "Rule 6": {'00': 'T', '01': 'G', '10': 'C', '11': 'A'},
    "Rule 7": {'00': 'A', '01': 'C', '10': 'G', '11': 'T'},
    "Rule 8": {'00': 'A', '01': 'G', '10': 'C', '11': 'T'}
}
DECODING_RULES = {
    rule: {v: k for k, v in mapping.items()}
    for rule, mapping in ENCODING_RULES.items()
}

# --- Conversion Functions ---
def text_to_binary(text):
    return ''.join(format(ord(ch), '08b') for ch in text)

def binary_to_text(binary_str):
    return ''.join(chr(int(binary_str[i:i+8], 2)) for i in range(0, len(binary_str),
8))

def binary_to_DNA(binary_str, mapping):
    if len(binary_str) % 2 != 0:
```

```
        binary_str += '0'
    return ''.join(mapping[binary_str[i:i+2]] for i in range(0, len(binary_str), 2))

def DNA_to_binary(dna, mapping):
    return ''.join(mapping[base] for base in dna)

def text_to_DNA(text, mapping):
    return binary_to_DNA(text_to_binary(text), mapping)

def DNA_to_text(dna, mapping):
    return binary_to_text(DNA_to_binary(dna, mapping))

# --- Quasigroup and Parastrophe Generation ---
def generate_latin_square(bases):
    n = len(bases)
    square = [[None]*n for _ in range(n)]

    def valid(r, c, v):
        return v not in square[r] and all(square[i][c] != v for i in range(n))

    def backtrack(idx=0):
        if idx == n*n:
            return True
        r, c = divmod(idx, n)
        for v in random.sample(bases, n):
            if valid(r, c, v):
                square[r][c] = v
                if backtrack(idx+1):
                    return True
        square[r][c] = None
        return False
```

```
    if not backtrack():
        raise ValueError("Failed to generate Latin square")
    return square

def convert_square_to_table(square, bases):
    return {bases[i]: {bases[j]: square[i][j] for j in range(len(bases))} for i in range(len(bases))}

def generate_random_quasigroup_table(bases):
    return convert_square_to_table(generate_latin_square(bases), bases)

def compute_parastrophes(Q):
    bases = list(Q.keys())
    P = [{b: {} for b in bases} for _ in range(5)]
    for x in bases:
        for y in bases:
            z1 = next(z for z in bases if Q[x][z] == y)
            z2 = next(z for z in bases if Q[z][y] == x)
            z3 = next(z for z in bases if Q[z][x] == y)
            z4 = next(z for z in bases if Q[y][z] == x)
            z5 = Q[y][x]
            P[0][x][y] = z1
            P[1][x][y] = z2
            P[2][x][y] = z3
            P[3][x][y] = z4
            P[4][x][y] = z5
    return P

def generate_random_parastrophe_table(Q):
    return random.choice(compute_parastrophes(Q))

# --- Quasigroup Encryption/Decryption ---
```

```
def encrypt_phase(dna_input, leader, table):
    prev, out = leader, ''
    for sym in dna_input:
        c = table[prev][sym]
        out += c
        prev = c
    return out

def left_divide(a, c, table):
    return next(m for m in table[a] if table[a][m] == c)

def decrypt_phase(dna_cipher, leader, table):
    prev, out = leader, ''
    for c in dna_cipher:
        m = left_divide(prev, c, table)
        out += m
        prev = c
    return out

from collections import Counter
from scipy.stats import chisquare

def chi_square_uniform_test(ciphertext, bases):
    observed = [ciphertext.count(b) for b in bases]
    expected = [len(ciphertext) / len(bases)] * len(bases)
    chi_stat, p_value = chisquare(f_obs=observed, f_exp=expected)
    return chi_stat, p_value

def plot_ciphertext_histogram(ciphertext, bases):
    freq = Counter(ciphertext)
    counts = [freq[b] for b in bases]
```

```
    plt.figure(figsize=(6, 4))
    plt.bar(bases, counts, color='skyblue', edgecolor='black')
    plt.xlabel('DNA Bases')
    plt.ylabel('Frequency')
    plt.title('Histogram of Ciphertext Base Distribution')
    plt.ylim(0, max(counts) * 1.2)  # Add some padding
    for i, count in enumerate(counts):
        plt.text(i, count + 1, str(count), ha='center', va='bottom')
    plt.tight_layout()
    plt.grid(axis='y', linestyle='--', alpha=0.7)
    plt.show()

# --- Main with Metrics ---
if __name__ == "__main__":
    plaintext = input("Enter plaintext: ")
    rule = random.choice(list(ENCODING_RULES.keys()))
    enc_map = ENCODING_RULES[rule]
    dec_map = DECODING_RULES[rule]

    # Encode to DNA
    dna_plain = text_to_DNA(plaintext, enc_map)

    # Phase I encryption
    leader1 = random.choice(DNA_BASES)
    Q = generate_random_quasigroup_table(DNA_BASES)
    phase1 = encrypt_phase(dna_plain, leader1, Q)

    # Phase II encryption (parastrophe)
    leader2 = random.choice(DNA_BASES)
    P = generate_random_parastrophe_table(Q)
    final_cipher = encrypt_phase(phase1, leader2, P)
    # Simulate encryption
```

```
_ = encrypt_phase(phase1, leader2, P)


# Decryption
phase1_dec = decrypt_phase(final_cipher, leader2, P)
dna_dec = decrypt_phase(phase1_dec, leader1, Q)
decrypted_text = DNA_to_text(dna_dec, dec_map)

# Output encryption/decryption results
print("\n--- Encryption ---")
print("Cipher DNA:", final_cipher)
print("Leader1:", leader1, " Leader2:", leader2, " Rule:", rule)

print("\n--- Decryption ---")
print("Decrypted Text:", decrypted_text)

chi_stat, p_value = chi_square_uniform_test(final_cipher, DNA_BASES)
print("\n--- Chi-Square Uniformity Test ---")
print(f"Chi-Square Statistic: {chi_stat:.4f}")
print(f"P-Value: {p_value:.4f}")
if p_value > 0.05:
    print("Result: Pass (Ciphertext appears uniformly distributed)")
else:
    print("Result: Fail (Ciphertext may not be uniformly distributed)")

plot_ciphertext_histogram(final_cipher, DNA_BASES)
```

Plaintexts for Encryption

Plaintext for encryption of different lengths was generated by ChatGPT.

Plaintext Length	Plaintext
10	Hello, Bob
20	Hello there, matey!!
40	In art and truth, beauty endures always.
80	Knowledge grows when curiosity meets dedication; each lesson shapes true wisdom.
100	In the silent hours of dawn, gentle light awakens dreams and kindles hope in every heart, so healing
500	In the quiet embrace of nature, the gentle murmur of a distant brook creates a symphony that soothes the soul. Each ray of sunlight filters through the leaves, casting a mosaic of shadows on the forest floor. The whispering wind carries tales of ancient times, while vibrant blossoms add splashes of color to the landscape. In this serene moment, every heartbeat echoes the promise of renewal and timeless beauty.Soft echoes of nature remind us that every ending births a new beginning. And hold on!!
1000	Success is not final, failure is not fatal. It is the courage to continue that counts. Life is a journey filled with ups and downs. What matters most is how we respond to challenges. Stay focused on your goals, work hard, and never give up. Every obstacle is an opportunity to learn. Surround yourself with positive people who inspire you. Time is precious, use it wisely. Develop good habits and

	maintain discipline. Reading expands knowledge, while action brings results. Small consistent efforts lead to big changes. Be kind, stay humble, and help others when possible. True happiness comes from within, not from material possessions. Appreciate the little things in life. Laughter is the best medicine, enjoy moments of joy. Travel, explore, and experience new cultures. Keep an open mind and embrace diversity. Mistakes are lessons in disguise, learn from them. Stay patient, as great things take time. Believe in your potential and keep pushing forward. The future belongs to those who are prepared.
1500	Communication is the foundation of strong relationships. Whether in personal or professional life, clear and respectful dialogue builds trust. Listening is just as important as speaking, understand before being understood. Technology has transformed how we connect, but face-to-face interactions remain invaluable. Empathy allows us to see things from others' perspectives. Honesty fosters deeper connections, even when the truth is difficult. Conflict is natural, but resolution requires patience and compromise. Words have power, use them wisely to inspire, not hurt. A simple thank you or I appreciate you can brighten someone's day. Teamwork achieves more than individual effort alone. Collaboration brings diverse ideas together for innovation. Leadership is about guiding, not controlling, empower others to grow. Time management increases productivity

	and reduces stress. Prioritize tasks based on importance and urgency. Breaks are essential for maintaining focus and creativity. Health is wealth, exercise regularly, eat well, and sleep sufficiently. Mental well-being is equally crucial, practice mindfulness and self-care. Financial discipline ensures long-term security, save and invest wisely. Lifelong learning keeps the mind sharp, read books, take courses, and seek new skills. Adaptability is key in a fast-changing world. Stay curious and open to new experiences. Gratitude turns what we have into enough. Positivity attracts opportunities, maintain an optimistic outlook. Keep at it.
2000	Education is the most powerful weapon which you can use to change the world. Knowledge empowers individuals and transforms societies. A well rounded education includes not only academics but also emotional and social learning. Critical thinking enables us to analyze information objectively. Creativity fuels innovation and problem solving. Curiosity drives discovery, never stop asking questions. Reading broadens perspectives and enhances imagination. Writing clarifies thoughts and improves communication. Mathematics teaches logic and precision. Science explains the wonders of the universe. History provides lessons from the past to shape a better future. Art and music express emotions beyond words. Physical education promotes health and teamwork. Technology is

	a tool, use it responsibly and ethically. Digital literacy is essential in the modern world. Respect for diversity fosters inclusive communities. Kindness costs nothing but means everything. Volunteering strengthens empathy and social bonds. Environmental awareness ensures a sustainable planet, reduce waste, recycle, and conserve resources. Small eco friendly habits make a big difference. Financial literacy helps manage money wisely, budgeting, saving, and avoiding debt. Entrepreneurship encourages innovation and self reliance. Hard work beats talent when talent does not work hard. Discipline turns goals into achievements. Time is non renewable, spend it on what truly matters. Failure is feedback, not defeat, learn and improve. Resilience helps bounce back from setbacks. Patience yields long term rewards. Self confidence comes from preparation and practice. Humility keeps us grounded despite success. Integrity means doing the right thing even when no one is watching. Honesty builds trust in relationships. Courage is taking action despite fear. Persistence turns dreams into reality. Gratitude brings contentment and joy. Positivity attracts opportunities and happiness. Laughter is universal medicine for the soul. Smiles.
2500	The journey of a thousand miles begins with a single step. Setting clear goals provides direction and motivation. Break big ambitions into smaller manageable tasks. Consistency is more effective than

	occasional intensity. Progress may be slow but perseverance ensures success. Self discipline is choosing what you want most over what you want now. Time management maximizes productivity, focus on priorities first. Distractions are everywhere, stay committed to your objectives. Learning from mistakes turns failures into stepping stones. Feedback helps refine skills and strategies. Adaptability is crucial in an ever changing world. Embrace challenges as opportunities to grow. Resilience means bouncing back stronger after setbacks. A positive mindset attracts solutions not problems. Gratitude shifts focus from what is lacking to what is abundant. Happiness comes from within not external validation. Kindness creates ripples of positivity, small acts matter. Respect differences, diversity enriches perspectives. Effective communication prevents misunderstandings, listen actively. Body language conveys unspoken messages, be mindful of it. Emotional intelligence fosters better relationships. Patience avoids rushed decisions with long term consequences. Integrity builds trust and credibility. Honesty even when difficult strengthens character. Accountability means owning actions and their outcomes. Teamwork achieves collective success, value each member and their contribution. Leadership inspires others through vision and example. Mentorship shares knowledge and accelerates growth. Financial literacy ensures wise money management,

	save invest avoid debt. Health is true wealth, exercise eat well sleep sufficiently. Mental well being requires self care and stress management. Lifelong learning keeps the mind sharp, read explore stay curious. Creativity solves problems in innovative ways. Critical thinking evaluates information objectively. Technology should enhance life not control it. Digital detoxes maintain balance in a connected world. Nature rejuvenates the spirit, spend time outdoors. Sustainability protects the planet for future generations. Reduce waste recycle and support eco friendly practices. Volunteering gives back to the community. Family bonds provide unconditional love and support. Friendships enrich life with shared experiences. Travel broadens horizons and fosters cultural appreciation. Laughter relieves stress and strengthens connections. The world awaits.
3000	Growth is a continuous process that requires patience, discipline, and consistent effort. No achievement is born overnight. Progress may be slow at times, but it is steady dedication that brings lasting success. Small steps taken daily lead to major milestones. Believe in your journey, even when the path seems unclear. Stay committed to your goals, and trust that your hard work will pay off. Challenges are opportunities in disguise. They teach us to be strong, to adapt, and to evolve. Difficult moments are often the ones that shape our character. Stay focused on

	what matters, and let go of what you cannot control. The most powerful thing you can do is take responsibility for your actions. Accountability brings growth. Learn from each mistake, reflect on each failure, and use them as stepping stones. Every setback carries a lesson. Your response determines your future. Attitude shapes experience. Approach life with optimism and gratitude. Appreciate the little things, and celebrate progress. Kindness and humility open doors. Treat others with respect, even when you disagree. Empathy builds understanding and reduces conflict. Communication is the bridge between confusion and clarity. Listen actively and speak with intention. Words carry weight, so use them to uplift. Honesty nurtures trust, and trust strengthens relationships. In any team, collaboration leads to innovation. Diverse ideas lead to better solutions. Everyone has a role to play. Great leaders inspire, support, and serve. Leadership is not about power, but about guidance. Encourage others to grow, and you will grow too. Time is a limited resource. Use it wisely. Organize your day, prioritize your tasks, and rest when needed. Productivity is not about doing more, but about doing what matters. Avoid distractions that steal your focus. Discipline is the key to freedom. Healthy routines build resilience. A balanced life includes physical, mental, and emotional well-being. Take care of your body through exercise and rest. Eat nourishing food. Sleep deeply.

	Maintain mental health through mindfulness and reflection. Make space for silence, nature, and stillness. These moments refresh your soul. Let go of comparison. Your journey is unique. Measure your progress by your own growth. Stay true to your values. Integrity is a compass that guides you in difficult times. Even when no one is watching, do what is right. Be brave enough to start. Be patient enough to keep going. Believe in your ability to change and adapt. Stay curious, and continue learning. Knowledge expands our understanding and unlocks potential. Read, ask questions, and welcome feedback. Feedback is a gift that helps us improve. Do not fear failure, for it is part of learning. What matters is that you rise each time you fall. Keep your vision clear and your heart strong. Support those around you. Build communities of encouragement and respect. Share your lights, and it will multiply. Practice gratitude every day. It challenges you.
3500	Success is not final, and failure is not fatal. What truly matters is the courage to continue when the path gets hard. Life is a journey filled with ups and downs, and every experience shapes who we become. Challenges are not meant to break us, but to help us discover our strengths. With perseverance, obstacles become opportunities. Stay focused on your goals, work hard, and never give up. Every small step forward matters. Consistency beats intensity. The

	people we surround ourselves with influence our mindset. Choose positivity and kindness. Encourage others, and they will uplift you in return. Listening is a powerful skill, often more impactful than speaking. Understand before trying to be understood. In communication, clarity and respect build lasting trust. Honesty creates deep connections, even when the truth is difficult. Be truthful, but always gentle. Empathy allows us to see the world through another person's eyes. It softens conflict and builds bridges. Mistakes are part of growth. Learn from them, reflect, and improve. Progress is not always visible, but each effort builds momentum. Time is our most precious resource. Use it with intention. Prioritize tasks that align with your values and goals. Breaks are necessary for sustained creativity and energy. Rest is not a reward, but a requirement. Good habits are the foundation of long term success. Discipline brings freedom. Self control lets you make better choices. Confidence comes from preparation and action. Face your fears, and they lose their power. Resilience is built in adversity. We grow when we adapt and endure. Adaptability allows us to thrive in a changing world. Keep an open mind, be willing to learn, and welcome new ideas. Curiosity leads to discovery. Reading expands our knowledge and imagination. Writing helps us organize thoughts and communicate clearly. Math teaches logic and precision. Science reveals the wonders of the universe. History offers lessons to guide our future. Art

	and music express what words cannot. Physical activity strengthens body and mind. Nutrition fuels performance. Sleep restores us. Mental health is equally important. Practice mindfulness and self compassion. Reach out when you need support. Financial literacy builds stability. Save regularly, spend wisely, and invest in what matters. Simplicity often leads to clarity. Let go of what no longer serves you. Gratitude turns what we have into enough. Celebrate progress, no matter how small. Positivity attracts new opportunities. A smile can change someone's day. Laughter lightens heavy moments. Acts of kindness create ripples of joy. Be generous not only with resources, but with your time and attention. Leadership is not about control, but about inspiration and service. Great leaders empower others. Collaboration brings diverse strengths together. Teamwork achieves more than individual effort. Conflict is natural, but resolution requires empathy and patience. Seek solutions, not blame. Everyone makes mistakes. Forgiveness sets us free. The past cannot be changed, but the future is unwritten. Dream big, but act with purpose. Set intentions and take consistent steps. Stay true to your values. Integrity is doing the right thing when no one is watching. Your actions define your character. Never underestimate the impact of your presence. The world needs your unique voice and talents. Let go of perfection and embrace progress. Life is not a race. It is a dance of

	moments, a song of connection, a canvas for your story. You are enough. Keep going.
5000	In the heart of a distant land, where golden hills roll into the horizon and the skies remain painted in eternal hues of lavender and gold, there lies a village untouched by time. It is said that the people of this village live in harmony, their lives intertwined not by force or necessity, but by a deep unspoken understanding passed down through generations. Every morning, as the sun rises behind the snow dusted peaks of Mount Aeloria, the villagers awaken to the sound of chimes hanging in doorways, each one uniquely tuned to the family that lives within. The soft music floats through the air like a hymn of unity, a reminder of the rhythm that binds all who dwell there. Among the villagers is a young woman named Elira, a weaver of exceptional talent. Her tapestries tell stories more vividly than words, depicting events not yet occurred and memories long forgotten. Some say she was born with the gift of foresight, though she claims her visions come only through listening deeply to the wind and watching how the leaves fall on her loom. Eliras days are spent in her sunlit studio at the edge of the forest, where birds gather on the windowsill and sing as she works. Her most prized creation, a tapestry titled The Thread of Truth, is kept hidden beneath layers of linen, shown only to those who truly seek it. It tells the story of a child born under a crimson moon, destined to unite the broken realms. One evening, as twilight

	began to settle over the village, a stranger arrived on horseback. Clad in a cloak woven from the night itself, the stranger spoke in riddles and carried a map inked in silver. He asked for Elira, for he had heard of her gift, and he believed she alone could interpret the hidden path that the stars had laid before him. Elira welcomed him into her studio, offering tea brewed with petals of the dreaming rose, a flower known to calm the mind and open the heart. As the fire crackled in the hearth, the stranger unrolled his map, revealing constellations unfamiliar to any known chart. Elira studied them in silence, her fingers tracing the lines with reverence. Then she spoke. This path, she whispered, is not one of direction but of transformation. You will not find your way by walking, you must become the road itself. Puzzled but compelled, the stranger stayed in the village for many moons, learning its rhythms, helping in the fields, listening to the stories told by the elders beside the sacred flame. Slowly, he changed. The weight he carried in his shoulders eased, the sharpness in his gaze softened, and laughter found its way into his voice. Elira watched him with quiet pride. She knew the journey was not about reaching a place, but about becoming someone new. And so, when the time came for him to leave, he did not take the map, for he no longer needed it. He carried the stars within him now. Years passed, and tales of a peacekeeper began to spread beyond the valleys and rivers, of a man who could unite feuding clans with a

	single story, who healed wounds by listening, and who bore on his cloak the threads of a tapestry never seen. Elira, older now but still steady of hand and bright of eye, continued to weave. Her newest creation, The Tapestry of Echoes, depicted moments of silence shared between strangers, the strength of unspoken bonds, and the beauty of stories that require no words. Visitors came from afar to witness her art, not only for its beauty, but for the feeling it stirred, of something ancient and sacred remembered. The village endured, untouched by war or famine, guided by the quiet wisdom of those who knew that power lies not in dominance, but in connection. And in this way, the golden hills continued to roll, the chimes continued to sing, and the wind whispered the names of those who had learned to listen. As we reflect upon stories such as Eliras, it becomes clear that the greatest journeys are not always measured in miles. Some take place entirely within the soul. And perhaps, in this ever turning world, it is these quiet revolutions of spirit that shape our future more than any conquest or discovery. In cities far beyond the village, under skies dimmed by towers and glass, there are still those who feel the echo of Eliras song. They may not know its origin, but they pause at unexpected moments, when a breeze carries the scent of distant flowers, when a stranger offers kindness with no expectation of return, or when silence feels as full as speech. These are the threads of the old world, still woven into the new.

	For every soul that listens, a loom begins to turn. Stories begin to form not in ink, but in experience, not on parchment, but in the living tapestry of our days. This is how we remember, not through monuments or legends alone, but through our actions, our choices, and the way we touch one anothers lives. And so, dear reader, you too are a thread. May you weave wisely, love fiercely, and listen well. The tapestry is vast, and your pattern is very exquisite.