Design and Analysis of Cryptographic Primitives based on Quasigroup

A thesis submitted during 2023 to the University of Hyderabad in partial fulfillment of the award of a Ph.D. degree in School of Computer and Information Sciences

by

Umesh Kumar

Reg. No.: 17MCPC03

School of Computer and Information Sciences

University of Hyderabad
P.O. Central University, Gachibowli
Hyderabad – 500046
Telangana, India

CERTIFICATE

This is to certify that the thesis entitled **"Design and Analysis of Cryptographic Primitives based on Quasigroup"** submitted by **Umesh Kumar** bearing **Reg. No. 17MCPC03** in partial fulfillment of the requirements for the award of **Doctor of Philosophy** in **Computer Science** is a bonafide work carried out by him under my supervision and guidance.

The thesis is free from plagiarism and has not been submitted previously in part or in full to this or any other University or Institution for the award of any degree or diploma. The student has the following publications before submission of the thesis for adjudication and has produced the evidence for the same.

- **Umesh Kumar**, Aayush Agarwal and V. Ch. Venkaiah: New Symmetric Key Cipher Based on Quasigroup. *International Conference on Cyber Security, Privacy and Networking (ICSPN 2021)*, LNNS, Vol. 370, Springer: pp. 83–94, 2022. https://doi.org/10.1007/978-981-16-8664-1_8. [Indexing: SCOPUS]

- **Umesh Kumar** and V. Ch. Venkaiah: A New Modified MD5-224 Bits Hash Function and an Efficient Message Authentication Code Based on Quasigroups. *International Conference on Cyber Security, Privacy and Networking (ICSPN 2021)*, LNNS, Vol. 370, Springer: pp. 1–12, 2022. https://doi.org/10.1007/978-981-16-8664-1_1. [Indexing: SCOPUS].

- **Umesh Kumar** and V. Ch. Venkaiah: An Efficient Message Authentication Code Based on Modified MD5-384 Bits Hash Function and Quasigroup. *International Journal of Cloud Applications and Computing*, **12**(1), pp. 1–27. https://doi.org/10.4018/IJCAC.308275. [Indexing: SCOPUS, UGC-CARE List (India)]

- **Umesh Kumar** and V. Ch. Venkaiah: A Novel Stream Cipher Based on Quasigroups and QG-PRNG. *International Journal of Information and Computer Security*, https://doi.org/10.1504/IJICS.2023.10050991, **(in press)**. [Indexing: DBLP, SCOPUS, UGC-CARE List (India)]

Further, the student has passed the following courses towards fulfillment of coursework requirement for Ph.D.

Course Code	**Title of the Course**	**Credits**	**Result**
CS851	Research Methods in Computer Science	4	Pass
CS801	Data Structure and Programming Lab	2	Pass
CS802	Algorithms	4	Pass
CS805	Cryptography	4	Pass

Supervisor

Prof. V. Ch. Venkaiah
School of Computer and
Information Sciences
University of Hyderabad
Hyderabad – 500046, India

Dean

Prof. Atul Negi
School of Computer and
Information Sciences
University of Hyderabad
Hyderabad – 500046, India

DECLARATION

I, **Umesh Kumar**, hereby declare that this thesis entitled **"Design and Analysis of Cryptographic Primitives based on Quasigroup"** submitted by me under the guidance and supervision of **Prof. V. Ch. Venkaiah** is a bonafide research work. I also declare that it has not been submitted previously in part or in full to this University or any other University or Institution for the award of any degree or diploma.

Name: Umesh Kumar **Date :**

Reg. No.: 17MCPC03 **Signature of the Student:**

Acknowledgements

I would like to thank University of Hyderabad for providing me with this opportunity to carry out my Ph.D. work successfully. I am also grateful to the University Grants Commission (UGC) for providing financial support under the scheme of the NET-JRF Fellowship. During my Ph.D., many people provided me with guidance, support, sacrifices, and encouragement. So, I take the privilege to express my sincere thanks and gratitude to all of them. Firstly, I would like to express my deepest gratitude to my supervisor **Prof. V. Ch. Venkaiah** for his valuable guidance and encouragement throughout my research work. He is not only a caring supervisor but also gave me enough freedom to interact with him without any inhibition. This has given me a lot of confidence to do my work smoothly. During our continued discussions, he introduced me to this exciting area of research. In fact, the majority of my ideas have been shaped by our thought-provoking discussions. Spending time and working with him has been a memorable, enjoyable, and enriching experience in my life. There are no words to adequately express my gratitude to him for patiently helping me to think clearly and consistently throughout this dissertation.

I would like to thank my doctoral review committee (DRC) members, **Dr. Y. V. Subba Rao** and **Dr. Wilson Naik**, for their valuable suggestions during my Ph.D. work. Apart from these members, I would also be grateful to **Prof. S. Bapi Raju** (my Ex-DRC member and Ex-faculty member of SCIS) for his valuable guidance. My interactions with them have been beneficial in identifying the research objectives and determining how to get there. I thank **Prof. Atul Negi**, Dean of the School of Computer and Information Sciences, University of Hyderabad, and Ex-Dean **Prof. Chakravarthy Bhagvati**, for providing me the necessary infrastructure

and academic support to carry out this research work. Also, I would like to thank **Prof. Alok Singh**, who reminded me many times about my research status, and all other faculty members and non-teaching staffs of the School of Computer and Information Sciences for their contributions.

I am very much indebted to my class-fellows/friends/seniors, Shuboo, Shadab, Madhuri, N. D. Patel, Narsimhulu, Dr. Abdul Basit, Dr. N. Chaitanya and Dr. Raghunadh Pasunuri, for their support during my Ph.D. tenure.

I would also like to give special thanks to my whole family members, including my father (**Mr. Sudeshwer Singh**) and my mother (**Mrs. Rameshwari Devi**) for their continuous support and understanding when undertaking my research work. Your prayer for me was what sustained me this far.

Umesh Kumar

Abstract

Quasigroups of order n can be represented as an $n \times n$ matrix in which each row as well as in each column has a different permutation of elements from a non-empty set $\mathbb{Z}_n$ so that no element appears more than once in any row or column. Quasigroups are important to cryptography because the number of quasigroups grows exponentially with its order, and every quasigroup has a unique inverse quasigroup. So, it makes an important case for the design of cryptosystems. This thesis mainly proposes three types of new crypto-primitives based on quasigroups. These are stream ciphers, block ciphers, and hash functions.

As the first contribution, we have proposed three variants of stream ciphers. The second stream cipher improves the speed of the first cipher, while the third stream cipher improves the memory consumption and security of both the first and second ciphers. The novelty of the proposed stream ciphers is that once a keystream is generated, it can be reused multiple times because the proposed ciphers are resistant to the reused key attack. Design of these ciphers is motivated by the fact that all conventional XOR-based stream ciphers are vulnerable to reused key attack.

As the second contribution, we have proposed two variants of block ciphers. The second block cipher is the revised version of the first block cipher that improves the security of the first block cipher. Both the ciphers are designed based on multiple quasigroups and use the same set of 16 optimal S-boxes of 4×4-bit in the form of an optimal quasigroup of order 16. Each of these sixteen S-boxes has the highest algebraic degree and the lowest linearity and differential characteristics. Therefore, these S-boxes provide great security against linear and differential attacks. The security and performance of the proposed ciphers are analyzed by comparing them with some of the existing

quasigroup-based proposals and we found that the proposed ciphers are more secure and efficient than that of the existing ciphers.

Finally, in the third contribution of this thesis, we have proposed two variants of hash functions and their corresponding message authentication codes (HMACs) based on quasigroup. The second hash function/HMAC is the extended version of the first hash function/HMAC that produces 160 bits more hash/MAC value. By the way, the first hash function and HMAC produce 224 bits of hash and MAC values, respectively. Each of these two hash functions can be seen as an extension of the MD5 since they are designed based on the underlying structure of MD5 along with a quasigroup. These designs are motivated by the fact that MD5 is vulnerable to a collision attack. The underlying structure of both the proposed hash function and HMAC are similar. The only difference between the two is that the quasigroup used in the hash function is publicly known, while the quasigroup used in HMAC acts as a secret key, thereby computing both the hash and MAC values both (hash function and HMAC) take the same amount of time. The security and the performance of the proposed schemes are analyzed by comparing them with their counterparts, such as SHA-224, SHA-384, HMAC-SHA-224, and HMAC-SHA-384. We found that the proposed schemes are more secure and efficient than the corresponding existing schemes.

Contents

CONTENTS

List of Figures

LIST OF FIGURES

List of Tables

LIST OF TABLES

Chapter 1

Introduction

With the increasing need to secure data, new cryptographic algorithms have become increasingly imperative. This is due to the fact that cryptography has become an integral part of data security today. In other words, as more hackers attempt to break into private conversation and communication channels, it has become vital to protect privacy and limit third-party visibility using various security tools or cryptographic algorithms. Cryptographic algorithms typically consist of an encryption algorithm, a decryption algorithm, a key generation algorithm, a hash function, a message authentication code (MAC), etc. These days, two types of encryption/decryption algorithms are commonly used for achieving message confidentiality: (i) symmetric-key algorithm (which is usually known as symmetric-key cryptosystem) and (ii) asymmetric-key algorithm (which is usually known as asymmetric-key or public-key cryptosystem).

1.1 Symmetric key cryptosystem

In the symmetric-key cryptosystem, two trusted parties, say Alice and Bob want to communicate confidentially on an insecure channel. As part of this agreement, Alice sends a confidential message to Bob or vice versa. Using a secret key, the original message (which is also known as plaintext) is transformed into an unintelligible form (also known as ciphertext); this process is called encryption. On the other hand, Bob recovers the original message by using the same key that was used by Alice in the encryption; this process is called decryption. Graphical representation of a symmetric-key cryptosystem is shown in Figure 1.1. AES, DES, IDEA, RC4, SEAL, and Blowfish are

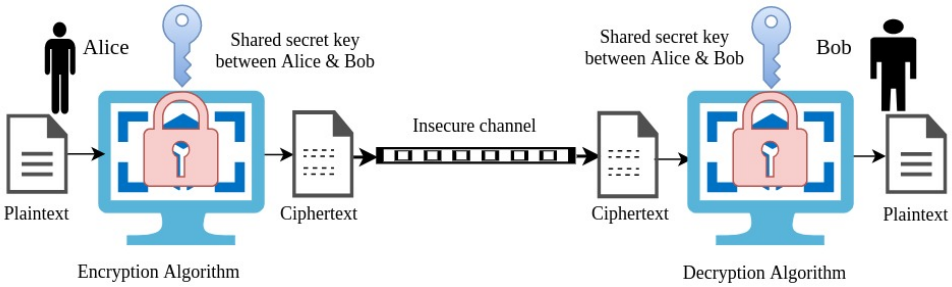


Figure 1.1: Workflow of the symmetric key cryptosystem.

some of the examples of symmetric-key cryptosystems. Symmetric-key cryptosystems are used in various applications such as credit cards, military, electronic commerce, digital media, encryption of passwords, email, documents, etc. However, the symmetric key cryptosystems have various drawbacks [72, 74]. One of the drawbacks is that it requires to share the secret key in advance to communicate between Alice and Bob. That is, one of the challenges is the management of key distribution. One possible solution for this key distribution is to use an asymmetric-key cryptosystem.

1.2 Asymmetric key cryptosystem

The concept of an asymmetric-key cryptosystem was introduced by Diffie and Hellman in 1976. In an asymmetric-key cryptosystem, two keys are used, one is called a public key and another is called a private key. Both public and private keys of the recipient (or Bob) are used for encryption and decryption, respectively. That is, Alice uses Bob's public key for encrypting the plaintext and Bob uses his own private key for decrypting the ciphertext. The graphical representation of an asymmetric-key cryptosystem is shown in Figure 1.2. Note that a public key can be known to "everyone", whereas a

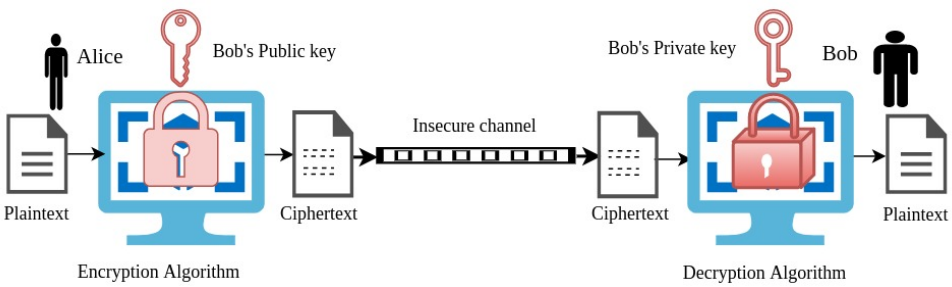


Figure 1.2: Workflow of the asymmetric key cryptosystem.

private key is only known by the recipient of the encrypted message. So a public-key cryptosystem would enable anyone to encrypt a message to be transmitted to Bob, and only Bob could decrypt the message. The RSA cryptosystem is one of the best examples of a public-key cryptosystem that was introduced by Rivest, Shamir and Adleman in 1977.

The primary strength of asymmetric-key cryptography over symmetric-key cryptography is that it is more secure than symmetric-key cryptography. This is because private keys never need to be revealed to anyone. But on the other hand, symmetric-key cryptography is more efficient than asymmetric-key cryptography. This is because symmetric-key cryptography requires fewer calculations than asymmetric-key cryptography. Depending on how the data is encrypted, symmetric-key cryptography or symmetric ciphers are of two types: (i) stream ciphers and (ii) block ciphers.

1.3 Stream cipher and block cipher

Stream ciphers encrypt a unit of data using a keystream which is as long as the plaintext and is generated based on a secret key, where the unit of data can be either a bit or a nibble or a byte, etc. A5/1, A5/2, RC4, SNOW, Edon80 are some of the examples of stream ciphers. Stream ciphers are widely used in cellular phones and wireless communications. For example, the A5/1 is used in GSM telephone communications, and the RC4 is used in wireless local area networks (WLANs). One of the security issues of the stream ciphers that use the XOR function to mix the plaintext and keystream is that they are vulnerable to reused key (two-time pad) attack. That is, once a keystream is generated, it can not be used more than once [72, 74]. These ciphers are also vulnerable to attacks such as a known-plaintext attack and insertion attack [7]. Also, a stream cipher may be analyzed to determine either the message or the employed secret key using several attacks such as ciphertext only attack, chosen-plaintext attack, chosen-ciphertext attack, etc. [67].

Block ciphers, on the other hand, encrypt a fixed amount of data at a time called a block. The size of the block depends on the encryption algorithm. DES and AES are examples of block ciphers. The DES was previously used as the standard for encryption. It was vulnerable to attacks such as brute force attack [18] because of its small key size of 56 bits, chosen-plaintext attack [9] and known-plaintext attack [52]. Hence a new

standard was required and therefore DES was replaced by AES [64]. Typically, stream ciphers are more efficient than block ciphers because of the following reasons:

- Stream ciphers work on individual bits and do not require buffering for large blocks.
- Padding a block is not needed in stream ciphers. Also, error propagation is less likely in stream ciphers.
- Stream ciphers require less sophisticated circuitry.

In addition, the stream ciphers are more appropriate, and in some cases mandatory (for example, in telecommunication applications) such as when buffering is limited or when characters must be individually processed as they are received, or when the block size cannot be determined before transmission. Since stream ciphers have minimal or no error propagation, they can be beneficial even in situations where transmission faults are highly probable.

1.4 Cryptographic hash function

A hash function takes an arbitrary length input message and produces a fixed length hash value, called the message digest or checksum. Generally, the digest value created by a hash function is known as a modification detection code (MDC). It detects the integrity of a message which is sent by a sender. A cryptographic hash function $\mathbb{H}$ has the following properties [69].

1. Pre-image resistant:- Given a hash value y, it is computationally infeasible to find a message x such that $\mathbb{H}(x) = y$.
2. Second pre-image resistant:- Given a message x_1 , it is computationally infeasible to find a second message x_2 such that $x_1 \neq x_2$ and $\mathbb{H}(x_1) = \mathbb{H}(x_2)$.
3. Collision resistance:- It must be computationally infeasible to find two messages x_1 , x_2 such that $x_1 \neq x_2$ but $\mathbb{H}(x_1) = \mathbb{H}(x_2)$.

The MD4, MD5, SHA-256, SHA-384, and SHA-512 are examples of hash functions given in the literature [33, 72, 74]. The security of these hash functions depends on

the size of the hash value and the underlying structure of the hash functions. These hash function are commonly used in message authentication codes [78], pseudo-random number generators [19], message signing, SSL, time stamping, and in many other cryptographic protocols.

1.5 Hash function with key or HMAC

Message authentication code (MAC) that uses a hash function is called HMAC. It was introduced by Bellare et al. in 1996 [8]. Later it was generalized and standardized by FIPS PUB 198-1 [78]. The output of HMAC is used to simultaneously verify both the authenticity and the data integrity of a message when two authorized parties communicate in an insecure channel. That is, the MAC-value is used to verify a sender's identity if two parties, say Alice and Bob are communicating in the presence of adversaries. Bob can use a MAC-value to ensure that the message he gets was truly transmitted by Alice and that it has not been altered or corrupted in transit. For this, Alice and Bob need to choose a MAC algorithm and exchange the secret key. Before transmitting the message, Alice calculates the message's MAC-value and appends it to the end. When Bob receives the message, he checks that the appended authentication tag is indeed the correct MAC-value, ensuring that the message was sent by Alice (or someone else with access to the secret key) and that it was not altered in transit. Because of the MAC's computation resistance property, an adversary will be unable to substitute his/her message or modify the message if he does not have access to the secret key. Even if an adversary has seen a certain number of previous messages with the authentication tags sent from Alice to Bob, he will still be unable to modify the current message or substitute a new one due to MAC's computation resistance property. The MAC is used in internet security protocols including SSL/TLS, SSH, IPsec. HMAC uses a hash function $\mathbb{H}$ and a secret key k shared between Alice and Bob. A HMAC, denoted by $\mathbb{H}_k$, has the following properties.

1. Easy to compute:- Given a message m and secret key k, $\mathbb{H}_k(m)$ is easy to compute.

2. Compression:- For an arbitrary length message m, $\mathbb{H}_k(m)$ produces a fixed length authentication tag also called the MAC-value of m.

3. Computation resistance:- Given a fixed number of pairs of messages and their corresponding authentication tags as $(m_1, x_1), (m_2, x_2), \ldots, (m_p, x_p)$, where $x_i = \mathbb{H}_k(m_i)$, and any other message $m \notin \{m_1, m_2 \ldots, m_p\}$ it is computationally infeasible to compute $\mathbb{H}_k(m)$ without the knowledge of k.

Note that a MAC-value is easy to compute with the knowledge of the key whereas very difficult to compute without the knowledge of the key. Because of this, it is possible that with the knowledge of the key one can find a collision for HMAC (such as in the case of MD5-based HMAC).

1.6 Motivation and Research goals

Research, invention, and augmentation in cryptography are not only a curiosity but also a necessity. This is because, for every cryptographic measure, a countermeasure has been found to make it ineffective. The weaknesses of the cryptosystems can be caused by social engineering or human error, or they can be discovered through cryptanalysis.

Cryptography is a collection of deterministic algorithms that one uses to protect information and communications against adversarial behavior. In other words, cryptography is primarily concerned with designing and analyzing protocols that prevent third parties from gaining access to private communications. A deterministic algorithm can be used to generate secret keys, create digital signatures, authenticate messages, verify messages, and protect the privacy of confidential information and communications such as credit card transactions and email correspondence. History has shown that cryptographic algorithms are designed based on several areas of mathematics, including Number theory, Group theory, Finite field, Linear algebra, Boolean algebra, and Boolean functions. Each of the approaches to design cryptographic algorithms using these mathematical areas employs an associative algebraic structure. In this thesis, we expand the set of approaches to design cryptographic algorithms by including a non-associative mathematical object called a quasigroup [68, 77]. Quasigroups play an important role in cryptography because of the following reasons:

- The number of quasigroups grows exponentially with its order, so they make an important case for the design of crypto-primitives.

- Quasigroups provide convenient tools for constructing crypto-primitives due to their closure and inversion properties.
- Due to the lack of associativity, one-way functions are easier to create.
- The crypto-primitives based on a quasigroup have a meager computational cost since they are table look-up operations.
- Quasigroup-based crypto-primitives are suitable for low resource devices such as smartphones, sensors, tablets, etc [6]. This is in contrast to the most popular cryptosystems such as AES, DES, and RSA, which drain the battery of such devices. With the increase of cloud services, the amount of data being transmitted and received by these devices is growing at an exponential rate [63].

This thesis focuses on designing new cryptographic algorithms based on quasigroups that are more friendly with hardware and software implementations. That is, using the quasigroup, we can improve the security and efficiencies of the conventional and some of the widely used standard cryptographic algorithms, such as MD5, RC4, etc. In 1992, Ronald Rivest proposed the MD5 hash function. It is a widely used hash function. This is because it is one of the hash functions requiring the least number of computations. Of late, many articles are published showing that the MD5 is not secure because the length of the hash value is too short. So, it is vulnerable to brute force birthday attacks [58], and a collision can be found within seconds with a complexity of around 2^{24} [73]. In another case, RC4 is one of the fastest stream ciphers widely used in various applications such as Wired Equivalent Privacy (WEP), SSL, Wi-Fi Protected Access (WPA), etc. RC4 is a byte-oriented cipher that uses the XOR function to mix the plaintext/ciphertext with keystream. So, the cipher is found to be vulnerable to reuse-key attack (two-time pad). This is one of the major hurdles in all XOR-based stream ciphers. Further, linear cryptanalysis has shown weaknesses in the DES cryptosystem [52]. Hence a new standard was required and therefore DES was replaced by AES. Both DES and AES are suitable for desktop-based applications [6]. We have also analyzed some existing quasigroup-based crypto-primitives such as stream ciphers, hash functions, and block ciphers. We found that some of these crypto-primitives are not as secure as is required; while some others have enhanced security at the expense of the efficiency of the crypto-primitives. So, quasigroup-based crypto-primitives that

are stronger and more efficient would be a good alternative to the existing quasigroup-based crypto-primitives. This is one of the primary motivations behind devising new crypto-promotives. So, we have set out the following goals:

1. **To devise new stream ciphers based on quasigroups**. To this end, we have designed the following:

   1.1. New symmetric key cipher based on a quasigroup and AES-256.

   1.2. a novel stream cipher based on a quasigroup of order 256 that uses a pseudo-random number generator based on a quasigroup of order 256, named as QG-PRNG. It is more efficient than that of the one mentioned in 1.1

   1.3. a stream cipher based on multiple quasigroups of order 16 that uses a pseudo-random number generator based on multiple quasigroups of order 16, named as MQG-PRNG. It is a revised version of both the algorithms mentioned above.

2. **To devise new block ciphers based on quasigroups**. To this end, we have arrived at the following.

   2.1. An efficient block cipher based on multiple optimal quasigroups and $\{e, d\}$-`transformation`.

   2.2. A block cipher based on multiple optimal quasigroups and $\{ne^{\ell}, nd^{\ell}\}$-`transformation`. It is a revised version of the one mentioned in 2.1.

3. **To devise new hash functions and HMACs based on quasigroups**. To this end, we have designed the following.

   3.1 An extended version of MD5, called here as the modified MD5-224 bits hash function and the corresponding message authentication code (HMAC) based on quasigroup.

   3.2 An efficient hash function, called here as the modified MD5-384 bits hash function and the corresponding message authentication code (HMAC) based on quasigroup. It is a revised version of the one specified in 3.1.

1.7 Thesis Contributions

Contributions of this thesis are broadly categorized into three parts with respect to the research goals discussed earlier. The first part discusses three variants of stream ciphers based on quasigroup; the second part discusses two block ciphers based on quasigroup, and the third part discusses two variants of hash functions and HMACs based on quasigroup.

1.7.1 New symmetric key cipher based on a quasigroup and AES-256

In this contribution, we have proposed a new stream cipher for encrypting/decrypting messages. It masks the weaknesses of the XOR-based stream ciphers and adds extra security. This is because it uses the quasigroup operation and its inverse instead of the XOR operation. For generating the keystream, we use AES-256. In fact, any secure pseudo-random number generator such as CRT-DPR 4 [4] can be employed for the generation of the keystream. However, we choose to describe the proposed stream cipher using the AES-256. The security of the proposed stream cipher is analyzed and the randomness of the obtained ciphertext is tested using the NIST-STS test suite. We found that the proposed cipher satisfied all the required properties. Previous works [46, 60] that use quasigroups in the design of secure systems are vulnerable to the chosen-plaintext and chosen-ciphertext attacks [45, 80]. The proposed cipher resists these attacks.

1.7.2 A novel stream cipher based on a quasigroup and QG-PRNG

It is an extension of the work stated in the previous sub-section 1.7.1. In this contribution, we have proposed a novel stream cipher that uses a keystream generated by a pseudo-random number generator, named QG-PRNG. The QG-PRNG is a quasigroup based pseudo-random number generator also designed and described in this thesis. Both the schemes (encryption/decryption and QG-PRNG) use a quasigroup of order 256. It is more efficient than the previous cipher mentioned in sub-section 1.7.1. This is because, it uses QG-PRNG in place of AES-256 for generating the keystream, and the QG-PRNG generates the keystream around 5 times faster than AES-256. Because of this, the new cipher is faster than the previous one. The cipher is analyzed against various attacks, including known-plaintext attack, chosen-plaintext attack, chosen-ciphertext

attack, reused key attack, and statistical attack. We found that the proposed cipher is resistant to these attacks. The novelty of this stream cipher and the previous version is that a keystream once generated can be reused multiple times. This is because the proposed ciphers are resistant to reused key attack as against the XOR-based stream ciphers. The security of the QG-PRNG is analyzed against the attacks such as exhaustive search attack and quasigroup attack. We found that QG-PRNG is resistant to these attacks. The randomness of the obtained ciphertext and pseudo-random number sequence is tested using the NIST-STS test suite. We found that both the ciphertext and the pseudo-random number sequences are highly random.

1.7.3 MQG-PRNG and non-associative quasigroup based stream cipher

It is a revised version of the works stated in the previous sub-sections 1.7.1 & 1.7.2. In this contribution, we have proposed a new stream cipher that uses a keystream generated by multiple quasigroups based pseudo-random number generator, named MQG-PRNG. The MQG-PRNG is also designed and described in this thesis. Both the encryption/decryption and MQG-PRNG algorithms are designed using multiple quasigroups of order 16, and they use 16 quasigroups of order 16. These 16 quasigroups are generated based on an original non-associative quasigroup of order 16. Mathematically, we have shown that the space of a single quasigroup can be leveraged to accommodate all these 16 quasigroups. This stream cipher is not only as secure as the previous ciphers stated in sub-sections 1.7.1 & 1.7.2, but also uses around 99% less space than the previous ciphers. This cipher is analyzed against various attacks, including known-plaintext attack, chosen-plaintext attack, chosen-ciphertext attack, and Time-memory-data tradeoff (TMDTO) attack. We found that the proposed cipher is resistant to these attacks. This cipher also overcomes the major hurdle that exists in the XOR-based stream ciphers against reused key attack. The security of the MQG-PRNG is analyzed against brute force attacks, and a study on the robustness of the MQG-PRNG against the slides and related-key attacks is carried out by analyzing the avalanche effect of the keystream, and we found that the MQG-PRNG satisfied all the required properties. The randomness of the obtained ciphertext and pseudo-random number sequence is tested using the NIST-STS test suite. We observed that both

the obtained ciphertext and the generated pseudo-random number sequence are highly random.

1.7.4 An efficient block cipher based on multiple optimal quasigroups and $\{e, d\}$-`transformation`

An efficient block cipher based on multiple quasigroups of order 16 is proposed. It uses 16 optimal S-boxes of 4×4 bits as an optimal quasigroup of order 16. It is an iterative cipher, and its design is based on the Substitution Permutation Network (SPN). It uses 16 optimal quasigroups of order 16 and a 128 bits secret key for encrypting/decrypting the messages. These 16 optimal quasigroups are constructed dynamically based on an original optimal quasigroup of order 16. Because of this, our cipher leverages the space of a single quasigroup and uses multiple quasigroups by generating them from an original quasigroup. That is, the space required by multiple optimal quasigroups is reduced to that of a single quasigroup. It performs a total of 16 rounds to encrypt or decrypt a block of 128 bits. Each round, except the last round of the encryption system, consists of a sequence of two transformations: (i) substitution and (ii) permutation. The last round only performs a substitution. The substitution layer is a key-dependent S-box layer and it is carried out using the $\{e, d\}$-`transformation`. The $\{e, d\}$-`transformation` is also defined in this thesis. It randomly selects an S-box out of 16, depending on the round sub-key.

We have analyzed the cipher for various attacks, including linear and differential attacks. We found that the proposed block cipher is resistant to these attacks. Also, the performance analysis (speed and time complexities) and diffusion power of the proposed cipher are analyzed by comparing with that of the AES-128 and other existing quasigroup based block ciphers [5, 6, 83]. Due to more computations, the proposed block cipher is slightly slower than AES-128, while the proposed cipher uses only 50% of the space of the AES-128. In addition, the proposed block cipher is more efficient than DES and other existing quasigroup-based block ciphers [5, 6, 83], and gives a better diffusion power than the existing quasigroup based block ciphers. The randomness of the obtained ciphertext is tested using the NIST-STS test suite. We observed that the proposed cipher produces highly random ciphertexts.

1.7.5 A block ciphers based on multiple optimal quasigroups and $\{ne^{\ell}, nd^{\ell}\}$-transformation

This work is an extension of the work discussed in the previous sub-section 1.7.4. It also uses the same 16 optimal S-boxes as an optimal quasigroup of order 16 as used in the previous block cipher. It uses $\{ne^{\ell}, nd^{\ell}\}$-transformation with 128 bits round key, and performs a total of 17 rounds to encrypt or decrypt a block of 128 bits. The $\{ne^{\ell}, nd^{\ell}\}$-transformation is also defined in this thesis. Each round, except the first and last rounds of the encryption system, consists of three transformations: (i) substitution, (ii) permutation, and (iii) add-round key. The first round performs only the add-round key and the last round performs the substitution and add-round key. The substitution layer (also called a non-linear transformation) is a key-dependent S-box layer and it is carried out using the $\{ne^{\ell}, nd^{\ell}\}$-transformation. That is, the $\{ne^{\ell}, nd^{\ell}\}$-transformation randomly selects an S-box out of 16, depending on the round key. The security and the randomness of this cipher are analyzed as in the case of the previous cipher mentioned in the previous sub-section 1.7.4, and we concluded that this cipher is also as secure or more than the previous one.

1.7.6 A QGMD5-224 bits hash function and a QGMAC-224 bits message authentication code based on a quasigroup

We proposed two schemes based on a quasigroup: (i) a cryptographic hash function, named here as QGMD5-224, and (ii) a message authentication code based on QGMD5-224, named here as QGMAC-224. The proposed schemes can be seen as extensions of the MD5 and HMAC-MD5. The QGMD5-224 hash function expands the hash size of the MD5 by converting 128 bits into 224 bits. The QGMAC-224 expands the MD5-based message authentication code (HMAC-MD5) by converting 128 bits into 224 bits. Both the expansions are carried out using the quasigroup expansion (QGExp128To224) and the quasigroup compression (QGComp224To128) layers. Note that the underlying structure of both the schemes QGMD5-224 and QGMAC-224 is the same. The only difference between the two is that the quasigroup used in the QGMD5-224 is publicly known, while the quasigroup used in the QGMAC-224 acts as a secret key. The security and efficiency of the proposed schemes (QGMD5-224 and QGMAC-224) are analyzed by comparing them with their counterparts, such as SHA-224 and HMAC-SHA-224. It

is observed that the proposed schemes are more secure and efficient than the existing proposals.

1.7.7 A QGMD5-384 bits hash function and a QGMAC-384 bits message authentication code based on a quasigroup

It is an extension of the work previously discussed in the sub-section 1.7.6. In this contribution also we propose two schemes based on a quasigroup: (i) a cryptographic hash function, named here as QGMD5-384, and (ii) a message authentication code based on QGMD5-384, named here as QGMAC-384. The primary goal of proposing these new schemes is to obtain a 160-bit longer hash value and MAC value than the previous one by spending a little bit of extra time. Because of this, the new schemes are found to be more secure than the previous ones. Also, the algorithm of QGMD5-384 uses an optimal quasigroup of order 16, while the algorithm of QGMAC-384 uses a quasigroup of order either 16 or 256. The proposed schemes can be seen as extensions of the MD5 and HMAC-MD5. The QGMD5-384 expands the hash size of the MD5 by converting 128 bits into 384 bits. The QGMAC-384 expands the HMAC-MD5 by converting 128 bits into 384 bits. Both the expansions are carried out through a series of QGExp128T384 expansion and QGComp384To128 compression layers. The QGExp128To384 expansion layer is implemented using two sub-expansion layers. The first sub-expansion layer of QGExp128To384 transforms 128 bits into 224 bits and it is referred to as QGExp128To224. The second sub-expansion layer of QGExp128To384 transforms 224 bits into 384 bits and it is referred to as QGExp224To384. And the QGComp384To128 compression layer compresses 384 bits into 128 bits. The security and efficiency of the proposed schemes (QGMD5-384 and QGMAC-384) are analyzed by comparing them with their counterparts, such as SHA-384 and HMAC-SHA-384. It is observed that the proposed schemes are more secure and efficient than the existing proposals.

1.8 Publications

- **Umesh Kumar**, Aayush Agarwal and V. Ch. Venkaiah: New Symmetric Key Cipher Based on Quasigroup. *International Conference on Cyber Security, Pri-*

vacy and Networking (ICSPN 2021), LNNS, Vol. 370, Springer: pp. 83–94, 2022. https://doi.org/10.1007/978-981-16-8664-1_8. [Indexing: SCOPUS]

- **Umesh Kumar** and V. Ch. Venkaiah: A New Modified MD5-224 Bits Hash Function and an Efficient Message Authentication Code Based on Quasigroups. *International Conference on Cyber Security, Privacy and Networking (ICSPN 2021)*, LNNS, Vol. 370, Springer: pp. 1–12, 2022. https://doi.org/10.1007/978-981-16-8664-1_1. [Indexing: SCOPUS].

- **Umesh Kumar** and V. Ch. Venkaiah: An Efficient Message Authentication Code Based on Modified MD5-384 Bits Hash Function and Quasigroup. *International Journal of Cloud Applications and Computing*, **12**(1), pp. 1–27. https://doi.org/10.4018/IJCAC.308275. [Indexing: SCOPUS, UGC-CARE List (India)]

- **Umesh Kumar** and V. Ch. Venkaiah: A Novel Stream Cipher Based on Quasigroups and QG-PRNG. *International Journal of Information and Computer Security*, https://doi.org/10.1504/IJICS.2023.10050991, **(in press)**. [Indexing: DBLP, SCOPUS, UGC-CARE List (India)]

Articles under review

- **Umesh Kumar** and V. Ch. Venkaiah: MQBC: A New Block Cipher Based on Multiple Quasigroups. *Wireless Personal Communications*, Springer. [Indexing: DBLP, SCIE, SCOPUS, UGC-CARE List (India)]

- **Umesh Kumar**, Y. Narasimhulu and V. Ch. Venkaiah: MQG-PRNG and Non-Associative Quasigroup based Stream Cipher. *ISC International Journal of Information Security (ISeCure)*. [Indexing: DBLP, ESCI, SCOPUS, UGC-CARE List (India)]

- **Umesh Kumar** and V. Ch. Venkaiah: An Efficient Block Cipher Based on Multiple Optimal Quasigroups. *International Journal of Information and Computer Security*. [Indexing: DBLP, SCOPUS, UGC-CARE List (India)]

1.9 Organization of the Thesis

The rest of the thesis is organized as follows:

Chapter 2: introduces the required mathematical objects such as quasigroup, left inverse quasigroup, right inverse quasigroup, optimal quasigroup, quasigroups as vector-valued Boolean functions, and the generation of quasigroups.

Chapter 3: presents a comprehensive survey of the existing quasigroup-based cryptographic primitives such as stream ciphers, block ciphers, hash functions, HMACs, etc.

Chapter 4: discusses the proposed three variants of stream ciphers based on quasigroup, including the basic structure, building elements, and security and performance analyses.

Chapter 5: discusses the proposed two variants of block ciphers based on multiple optimal quasigroups, including the basic structure, building elements, and security and performance analyses.

Chapter 6: describes the proposed two variants of hash functions and HMACs based on quasigroup, including the basic structure, building elements, and analyses the security and performance of the proposed schemes.

Chapter 7: presents the concluding remarks of the research work done, including future directions of research.

Chapter 2

Mathematical Backgrounds

2.1 Latin square

Definition 2.1.1. *A Latin square of order n is a $n \times n$ matrix in which the entries are taken from a finite set $\mathbb{S}$ and the symbols are arranged in such a way that each symbol occurs only once in each row and only once in each column.*

Example 2.1.2. *Table 2.1 is an example of a Latin square of order 6, where the elements are from the set $\mathbb{S} = \{a, b, c, d, e, f\}$.*

Table 2.1: Latin square of order 6.

a	b	c	d	e	f
d	e	f	a	b	c
c	d	e	f	a	b
b	c	d	e	f	a
e	f	a	b	c	d
f	a	b	c	d	e

The concept of Latin square was introduced by the mathematician Leonhard Euler $(1707 - 1783)$, and used Latin characters as symbols, but any set of symbols can be used; in the above example, the alphabetic sequence a, b, c, d, e, f can be replaced by the integer sequence $1, 2, 3, 4, 5, 6$.

2.1.3 Orthogonal representation of a Latin square

The orthogonal representation of the Latin square is a set of n^2 triples obtained by writing each entry of a $n \times n$ Latin square as a triple (r, c, s), where r is the row number, c is the column number, and s is the symbol. Example 2.1.4 shows a Latin square and its corresponding orthogonal representation.

Example 2.1.4. *Let $\mathbb{S} = \{1, 2, 3, 4\}$ be a set of order 4. A Latin square over $\mathbb{S}$ and its corresponding orthogonal representation are given in Table 2.2 (a) and in Table 2.2 (b), respectively.*

Table 2.2: Latin square of order 4 and corresponding orthogonal representation.

1	2	3	4	(1,1,1)	(1,2,2)	(1,3,3)	(1,4,4)
2	1	4	3	(2,1,2)	(2,2,1)	(2,3,4)	(2,4,3)
3	4	1	2	(3,1,3)	(3,2,4)	(3,3,1)	(3,4,2)
4	3	2	1	(4,1,4)	(4,2,3)	(4,3,2)	(4,4,1)
(a)				(b)			

Various operations on a Latin square can be performed to form another Latin square, in which one of the operations can be explained using an orthogonal representation of the Latin square. That is, by permuting the rows, columns, and symbols of an orthogonal representation of the Latin square, we can obtain a new Latin square, also called isotopic to the original Latin square [75]. That is, by permuting (r, c, s) of a Latin square, we can form 6 different Latin squares. For example, if we replace each triple (r, c, s) by (c, r, s), then we get the transpose of the original Latin square.

2.1.5 Number of Latin squares

For any n, Latin squares of order n can be easily constructed, but counting the distinct Latin squares of a large order n is very challenging. This is because the number of Latin squares of order n increases greatly as n increases. The number of distinct Latin squares of order n can be calculated by the following equation [1, 79]

$$\mathbb{L}_n = n! \times (n-1)! \times \mathbb{R}_n \tag{2.1}$$

where $\mathbb{L}_n$ denotes the number of distinct Latin squares, $\mathbb{R}_n$ denotes the number of distinct reduced Latin squares, where a reduced Latin square is one in which the first

row and the first column are in the natural order. An example of a reduced Latin square of order 4 is given in Table 2.2 (a). For $n \leq 11$, the results of $\mathbb{L}_n$ and $\mathbb{R}_n$ are given in Table 2.3 [14, 61]. The most recent case for $n = 11$ is determined in [53], and for $n \geq 12$, the problem remains open for a Latin square researcher [14, 61]. Also, the

Table 2.3: The number of Latin squares and reduced Latin squares of order n

n	number of Latin squares ($\mathbb{L}_n$)	number of reduced Latin squares ($\mathbb{R}_n$)
1	1	1
2	2	1
3	12	1
4	576	4
5	161, 280	56
6	812, 851, 200	9, 408
7	61, 479, 419, 904, 000	16, 942, 080
8	108, 776, 032, 459, 082, 956, 800	535, 281, 401, 856
9	5, 524, 751, 496, 156, 892, 842, 531, 225, 600	377, 597, 570, 964, 258, 816
10	9, 982, 437, 658, 213, 039, 871, 725, 064, 756, 920, 320, 000	7, 580, 721, 483, 160, 132, 811, 489, 280
11	776, 966, 836, 171, 770, 144, 107, 444, 346, 734, 230, 682, 311, 065, 600, 000	5, 363, 937, 773, 277, 371, 298, 119, 673, 540, 771, 840
≥ 12	?	?

Equation (2.1) is not practical for larger n. Therefore the following inequality gives an estimate of the most accurate upper and lower bounds of the Latin squares of order n [34]

$$\frac{(n!)^{2n}}{n^{n^2}} \leq |\mathbb{L}_n| \leq \prod_{k=1}^{n} (k!)^{\frac{n}{k}}, \tag{2.2}$$

where $|\mathbb{L}_n|$ denotes the number of Latin squares of order n. For $n = 2^k, k = 4, 5, 6, 7, 8$, these numbers are as follows:

$$0.101 \times 10^{119} \leq |\mathbb{L}_{16}| \leq 0.689 \times 10^{138}, \tag{2.3}$$

$$0.414 \times 10^{726} \leq |\mathbb{L}_{32}| \leq 0.985 \times 10^{785}, \tag{2.4}$$

$$0.133 \times 10^{4008} \leq |\mathbb{L}_{64}| \leq 0.176 \times 10^{4169}, \tag{2.5}$$

$$0.337 \times 10^{20666} \leq |\mathbb{L}_{128}| \leq 0.164 \times 10^{21091}, \tag{2.6}$$

$$0.304 \times 10^{101724} \leq |\mathbb{L}_{256}| \leq 0.753 \times 10^{102805}. \tag{2.7}$$

2.2 Quasigroup

Definition 2.2.1. *Let $\mathbb{Z}_n$ be the set of non-negative integers less than n. A quasigroup $Q = (\mathbb{Z}_n, *)$ defined over the set $\mathbb{Z}_n$ with a binary operation $*$ satisfies the following properties:*

(i) For all t_1, $t_2 \in \mathbb{Z}_n$, $t_1 * t_2 \in \mathbb{Z}_n$, (Closure property).

(ii) For each pair $(t_1, t_2) \in \mathbb{Z}_n \times \mathbb{Z}_n$, there exists unique pair $(t_3, t_4) \in \mathbb{Z}_n \times \mathbb{Z}_n$, such that $t_1 * t_3 = t_2$ and $t_4 * t_1 = t_2$.

- *Quasigroups also satisfy the following cancellation properties:*

(iii) $t_1 * t_2 = t_1 * t_3 \Rightarrow t_2 = t_3$ (Left cancelation).

(iv) $t_2 * t_1 = t_3 * t_1 \Rightarrow t_2 = t_3$ (Right cancelation).

Note that the binary operation $*$ is also called a quasigroup operation corresponding to the quasigroup Q. For a quasigroup Q, properties (i) and (ii) must be satisfied. The following example illustrates an example of a quasigroup of order 5.

Example 2.2.2. *Table 2.4 is an example of a quasigroup $Q = (\mathbb{Z}_5, *)$ of order 5 over the set $\mathbb{Z}_5$=$\{0,1,2,3,4\}$. Note that for $t_1 = 2$ and $t_2 = 4$, $t_3 = 3$ and $t_4 = 1$ are the unique elements of $\mathbb{Z}_5$. This is because $t_1 * t_3 = 2 * 3 = 4 = t_2$ and $t_4 * t_1 = 1 * 2 = 4 = t_2$. This is true for all $t_1, t_2 \in \mathbb{Z}_5$.*

A quasigroup is an algebraic structure resembling a group in the sense that division is always possible. For any positive integer n there exists a quasigroup $Q = (\mathbb{Z}_n, *)$ of order n. Quasigroups differ from groups mainly in that they do not necessarily require the properties of associativity and commutativity to be satisfied. They also do not

Table 2.4: Quasigroup of order 5.

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

need the existence of identity. That is, every group is a quasigroup but the converse is not true. Quasigroups of order n are usually represented using an $n \times n$ multiplication table. This multiplication table is formed by the permutations of the elements of the set $\mathbb{Z}_n = \{0, 1, 2, \ldots, n-1\}$ in such a way that each element occurs exactly once in each row and exactly once in each column. Such a table is also called a Latin square [16]. Also, the number of quasigroups is the same as that of the Latin squares. Since the number of Latin squares increases rapidly with its order it follows that the number of quasigroups increases rapidly with its order. That is, the generation of all the possible quasigroups of an arbitrary order n (where $n \geq 12$) is a hard problem. Properties (i) and (ii) of the quasigroups enforce the operation table of a quasigroup to be a Latin square. Therefore, Equation (2.2) is also an estimate of the number of quasigroups. Hence, the number of quasigroups of orders $16, 32, 64, 128$, and 256 also satisfy the equations 2.3, 2.4, 2.5, 2.6, and 2.7, respectively.

2.2.3 Non-associative quasigroup

Definition 2.2.4. *A quasigroup $Q = (\mathbb{Z}_n, *)$ is said to be a non-associative quasigroup if the following properties are satisfied:*

(i) Q must be a quasigroup (see, Definition 2.2.1)

(ii) If $\exists\ t_1, t_2, t_3 \in \mathbb{Z}_n$, $(t_1 * t_2) * t_3 \neq t_1 * (t_2 * t_3)$ (Non-associative property).

Like quasigroup generation, the generation of all possible non-associative quasigroups of the larger order is also a hard problem. This is because the number of non-associative quasigroups grows exponentially with its order. For instance, the number of possible quasigroups, associative quasigroups, and non-associative quasigroups up to order 6 is shown in Table 2.5. Note that associative/non-associative quasigroups

are a subset of all possible quasigroups of order n, $n \geq 2$. That is, the total number of all possible quasigroups is equal to the sum of the number of both the associative and non-associative quasigroups. As a result, given in Table 2.5, it can be observed that

Table 2.5: Number of associative quasigroups, non-associative quasigroups, and quasigroups

Order	Associative quasigroups	Non-associative quasigroups	Quasigroups
2	2	0	2
3	3	9	12
4	16	560	576
5	30	161,250	161,280
6	480	812,850,720	812,851,200

the number of associative quasigroups of order n (denoted by $NAQ(n)$) lies between $(n-1)!$ and $n!$, $2 < n \leq 6$, i.e. $(n-1)! < NAQ(n) < n!$. For a large value of n, it is a longstanding open problem to find a suitable tight bound to approximate the number of associative quasigroups of order n or to prove that such bounds do not hold. So, from Equation 2.2, the approximated number of non-associative quasigroups is bound above by

$$NNAQ(n) \leq \prod_{k=1}^{n} (k!)^{\frac{n}{k}} - NAQ(n) \approx \prod_{k=1}^{n} (k!)^{\frac{n}{k}}, \tag{2.8}$$

where $NNAQ(n)$ denotes the number of non-associative quasigroups of order n. That is, for $n = 16$ and using Equation 2.3, the approximated maximum number of non-associative quasigroups of order 16 is bonded above by

$$NNAQ(16) \approx 0.689 \times 10^{138} \approx 2^{456}. \tag{2.9}$$

2.2.5 Left inverse, right inverse and n-quasigroup

Definition 2.2.6. *Let $LIQ = (\mathbb{Z}_n, \backslash)$ denotes the left inverse quasigroup of the quasigroup $Q = (\mathbb{Z}_n, *)$. Then the LIQ satisfies the following conditions:*

(i) LIQ must be a quasigroup.

(ii) $t_2 * t_1 = t_3 \Leftrightarrow t_1 = t_2 \backslash t_3$, where $t_1, t_2, t_3 \in \mathbb{Z}_n$.

The binary operation '\' is a left inverse operator (or a left inverse quasigroup operation) corresponding to the quasigroup $LIQ = (\mathbb{Z}_n, \backslash)$. The example given below illustrates the concept of left inverse quasigroup.

Example 2.2.7. *Consider the quasigroup* $Q =< \mathbb{Z}_5, * >$ *with* $\mathbb{Z}_5 = \{0, 1, 2, 3, 4\}$*, let its operation table be as in Table 2.4. Then, the corresponding left inverse quasigroup is* $LIQ =< \mathbb{Z}_5, \setminus >$ *whose operation table is given in Table 2.6.*

Table 2.6: Operation table of LIQ.

$\setminus$	0	1	2	3	4
0	2	4	3	1	0
1	3	0	1	4	2
2	1	2	4	0	3
3	4	3	0	2	1
4	0	1	2	3	4

Definition 2.2.8. *Let* $RIQ = (\mathbb{Z}_n, /)$ *denotes the right inverse quasigroup of the quasigroup* $Q = (\mathbb{Z}_n, *)$*. Then the RIQ satisfies the following conditions:*

(i) RIQ must be a quasigroup.

(ii) $t_2 * t_1 = t_3 \Leftrightarrow t_2 = t_3/t_1$*, where* $t_1, t_2, t_3 \in \mathbb{Z}_n$*.*

The binary operation '/' is a right inverse operator (or a right inverse quasigroup operation) corresponding to the quasigroup $RIQ = (\mathbb{Z}_n, /)$. The example given below illustrates the concept of right inverse quasigroup.

Example 2.2.9. *Consider the quasigroup* $Q =< \mathbb{Z}_5, * >$ *with* $\mathbb{Z}_5 = \{0, 1, 2, 3, 4\}$*, let its operation table be as in Table 2.4. Then, the corresponding right inverse quasigroup is* $RIQ =< \mathbb{Z}_5, / >$ *whose operation table is given in Table 2.7.*

Table 2.7: Operation table of RIQ.

/	0	1	2	3	4
0	4	2	0	1	3
1	1	4	2	3	0
2	3	1	4	0	2
3	2	0	3	4	1
4	0	3	1	2	4

Definition 2.2.10. *An n-quasigroup is a finite algebra $\langle \mathbb{Z}_n, f, f_1, \ldots, f_n \rangle$ consisting of the elements of $\mathbb{Z}_n$ with $n+1$ n-ary operations and satisfies the following identities [60]:*

$$\begin{aligned} f(f_1(t_1, t_2, \ldots, t_n), t_2, \ldots, t_n) &= t_1 = f_1(f(t_1, t_2, \ldots, t_n), t_2, \ldots, t_n) \\ f(t_1, f_2(t_1, t_2, \ldots, t_n), \ldots, t_n) &= t_2 = f_2(t_1, f(t_1, t_2, \ldots, t_n), \ldots, t_n) \\ &\ldots \\ f(t_1, t_2, \ldots, f_n(t_1, t_2, \ldots, t_n)) &= t_n = f_n(t_1, t_2, \ldots, f(t_1, t_2, \ldots, t_n)) \end{aligned}$$

where $f, f_1, f_2, \ldots, f_n$ are n-ary operations (or n-ary quasigroup operations) such that $f(t_1, t_2, \ldots, t_n) = t_{n+1} = f_i(t_1, t_2, \ldots, t_n)$, $1 \leq i \leq n$, i.e. the knowledge of any n elements of $t_1, t_2, \ldots, t_n, t_{n+1}$ allows an n-ary operation to uniquely determine the remaining one element.

This thesis deals with a 2-ary (binary) quasigroup $\langle \mathbb{Z}_n, f, f_1, f_2 \rangle$, defined over a non-empty set Z_n together with three 2-ary (binary) operations f, f_1, and f_2. This is defined in Lemma 2.2.11, where f, f_1, and f_2 are denoted by $*, \backslash$, and $/$, respectively. Also, the symbols $*, \backslash$, and $/$ are called a quasigroup, a left inverse quasigroup, and a right inverse quasigroup operations, respectively (see definitions 2.2.1, 2.2.6 and 2.2.8). So, $f(t_1, t_2)$ is represented as $t_2 * t_1$, $f_1(t_1, t_2)$ is represented as $t_2 \backslash t_1$, and $f_2(t_1, t_2)$ is represented as t_2/t_1.

Lemma 2.2.11. *A quasigroup $Q = (\mathbb{Z}_n, *, \backslash, /)$ is an algebra with three binary operations $(*, \backslash, /)$, and satisfies the following identities:*

$$t_2 \backslash (t_2 * t_1) = t_1 \tag{2.10}$$

$$t_2 * (t_2 \backslash t_1) = t_1 \tag{2.11}$$

$$(t_2 * t_1)/t_1 = t_2 \tag{2.12}$$

$$(t_2/t_1) * t_1 = t_2 \tag{2.13}$$

Proof. Since $LIQ = (\mathbb{Z}_n, \backslash)$ is the left inverse quasigroup of the quasigroup $Q = (\mathbb{Z}_n, *)$. Then, from property (ii) of Definition 2.2.6, for each triplet $(t_1, t_2, t_3) \in \mathbb{Z}_n \times \mathbb{Z}_n \times \mathbb{Z}_n$, we have

$$t_2 * t_1 = t_3 \Leftrightarrow t_1 = t_2 \backslash t_3,$$

Hence, $t_2 \backslash (t_2 * t_1) = t_2 \backslash t_3 = t_1$.
In property (ii) of Definition 2.2.6, if we interchange the variables t_1 and t_3, we have

$$t_2 * t_3 = t_1 \Leftrightarrow t_3 = t_2 \backslash t_1.$$

Hence, $t_2 * (t_2 \setminus t_1) = t_2 * t_3 = t_1$.
Using the operation tables Table 2.4 and Table 2.6 corresponding to $(\mathbb{Z}_5, *)$ and $(\mathbb{Z}_5, \setminus)$, respectively, we can also prove that both identities (2.10) and (2.11) are true, $\forall\ t_1, t_2 \in \mathbb{Z}_5$.

Similarly, using the property of the right inverse quasigroup, defined in Definition 2.2.8, both the identities (2.12) and (2.13) can also be proved. Also, using the operation tables Table 2.4 and Table 2.7 corresponding to $(\mathbb{Z}_5, *)$ and $(\mathbb{Z}_5, /)$, respectively, both the identities (2.12) and (2.13) can be proved to be true, $\forall\ t_1, t_2 \in \mathbb{Z}_5$. □

2.2.12 {e, d}-transformation based on quasigroup

Let $\mathbb{Z}_n = \{p_0, p_1, \ldots, p_{n-1}\}$ be an alphabet, and $Q = (\mathbb{Z}_n, *, \setminus)$ be the quasigroup discussed earlier. Let ℓ be a leader (or a seed value) which is used in $\{e, d\}$-transformation. Then, the e-transformation is denoted by a mapping $f_{(*,\ell)} : \mathbb{Z}_n^+ \longrightarrow \mathbb{Z}_n^+$, where $\mathbb{Z}_n^+$ denotes the set of nonempty strings of the alphabet $\mathbb{Z}_n$, and it is defined as in the following equation.

$$\begin{aligned} f_{(*,\ell)}(p_0, p_1, \ldots, p_{k-1}) &= c_0, c_1, \ldots, c_{k-1}, \text{ for } k \geq 1, \\ \text{where } c_0 = \ell * p_0,\ c_1 &= c_0 * p_1,\ \ldots,\ c_{k-1} = c_{k-2} * p_{k-1}. \end{aligned} \tag{2.14}$$

Similarly, the d-transformation is denoted by a mapping $f_{(\setminus,\ell)} : \mathbb{Z}_n^+ \longrightarrow \mathbb{Z}_n^+$, where $\mathbb{Z}_n^+$ denotes the set of nonempty strings of the alphabet $\mathbb{Z}_n$, and it is defined by the following equation.

$$\begin{aligned} f_{(\setminus,\ell)}(c_0, c_1, \ldots, c_{k-1}) &= p_0, p_1, \ldots, p_{k-1}, \text{ for } k \geq 1, \\ \text{where } p_0 = \ell \setminus c_0,\ p_1 &= c_0 \setminus c_1,\ \ldots,\ p_{k-1} = c_{k-2} \setminus c_{k-1}. \end{aligned} \tag{2.15}$$

So, we can say that the sixtuple $(\mathbb{Z}_n, *, \setminus, \ell, f_{(*,\ell)}, f_{(\setminus,\ell)})$ is a quasigroup cipher over the alphabet $\mathbb{Z}_n$, and its correctness is shown by the following lemma.

Lemma 2.2.13. *If $(\mathbb{Z}_n, *, \setminus, \ell, f_{(*,\ell)}, f_{(\setminus,\ell)})$ is a quasigroup cipher over the alphabet $\mathbb{Z}_n = \{p_0, p_1, \ldots, p_{n-1}\}$, then $f_{(\setminus,\ell)}(f_{(*,\ell)}(p)) = p$, where p is the plaintext derived from the alphabet $\mathbb{Z}_n$.*

Proof. Let

$$f_{(*,\ell)}(p_{i_1}, p_{i_2}, \ldots, p_{i_j}) = c_{i_1}, c_{i_2}, \ldots, c_{i_j}, \text{ for some } j \geq 1, \text{ and } 0 \leq i \leq n-1.$$

Also, let

$$f_{(\setminus,\ell)} \circ f_{(*,\ell)}(p_{i_1}, p_{i_2}, \ldots, p_{i_j}) = f_{(\setminus,\ell)}(c_{i_1}, c_{i_2}, \ldots, c_{i_j}) = x_{i_1}, x_{i_2}, \ldots, x_{i_j}.$$

So, from Equation (2.14), we have

$$c_{i_1} = \ell * p_{i_1},\ c_{i_2} = c_{i_1} * p_{i_2},\ \ldots,\ c_{i_j} = c_{i_{j-1}} * p_{i_j}.$$

Also, from Equation (2.15), we have

$$x_{i_1} = \ell \setminus c_{i_1},\ x_{i_2} = c_{i_1} \setminus c_{i_2},\ \ldots,\ x_{i_j} = c_{i_{j-1}} \setminus c_{i_j}.$$

So, by Lemma 2.2.11,

$$\begin{aligned} x_{i_1} &= \ell \setminus (\ell * p_{i_1}) = p_{i_1}, \\ x_{i_2} &= c_{i_1} \setminus (c_{i_1} * p_{i_2}) = p_{i_2}, \\ &\ldots, \\ x_{i_j} &= c_{i_{j-1}} \setminus (c_{i_{j-1}} * p_{i_j}) = p_{i_j}. \end{aligned}$$

Hence, $f_{(\setminus,\ell)}(f_{(*,\ell)}(p)) = p$. □

So, it is quite clear from Lemma 2.2.13 that $f_{(*,\ell)}$ is an encoding function and $f_{(\setminus,\ell)}$ is a decoding function, for encryption and decryption over the alphabet $\mathbb{Z}_n$, respectively. The example given below illustrates the correctness of this lemma.

Example 2.2.14. *Consider the quasigroup $Q = (\mathbb{Z}_5, *, \setminus)$ with $\mathbb{Z}_5 = \{0, 1, 2, 3, 4\}$. Let its operation tables be as given in Table 2.4 and Table 2.6. Let $\ell = 4$ and plaintext $P = 2042301431$. Then, for encrypting the plaintext P, we have applied the encoding function $f_{(*,4)}$ and the quasigroup as shown in Table 2.4. So, the ciphertext*

$$C = f_{(*,4)}(P) = 2300234434.$$

For decrypting the ciphertext C, we have used left inverse quasigroup given in Table 2.6 and decoding function $f_{(\setminus,4)}$. The recovered plaintext is

$$P = f_{(\setminus,4)}(C) = 2042301431.$$

2.2.15 New $\{e, d\}$-transformation based on quasigroup

In this thesis, we also use a new $\{e, d\}$-transformation for designing the new cryptographic primitives. The use of this transformation is that it allows us to split the transformation into two parts: (i) $\{ne^l, nd^l\}$-transformation, and (ii) $\{ne^r, nd^r\}$-transformation, where ne^l-transformation and nd^l-transformation are the mutually inverse transformations of each other, similarly ne^r-transformation and nd^r-transformation

are the mutually inverse transformations of each other. That is, if ne^l-transformation (ne^r-transformation) is used in the encryption algorithm, then nd^l-transformation (nd^r-transformation) is used in the decryption algorithm, and vice versa.

Consider a non-empty alphabet $\mathbb{Z}_n = \{p_0, p_1, \ldots, p_{n-1}\}$ and a quasigroup $Q = (\mathbb{Z}_n, *, \backslash, /)$, and let a secret key to be used in the new $\{e, d\}$-transformation be $K = \{k_0, k_1, \ldots, k_{n-1}\}$. Let ne^l, nd^l, ne^r and nd^r transformations be mappings $f(*_\backslash, K)$, $f(\backslash_*, K)$, $f(*_/, K)$, $f(/_*, K)$ from $\mathbb{Z}_n^+$ to $\mathbb{Z}_n^+$, where $\mathbb{Z}_n^+$ denotes the set of non-empty strings over the alphabet $\mathbb{Z}_n$, and defined as in the following equations:

$$ne^l : \begin{cases} f(*_\backslash, K)(p_0, p_1, \ldots, p_{r-1}) = c_0, c_1, \ldots, c_{r-1}, r \geq 1 \\ \text{where } c_0 = k_0 * p_0, c_1 = k_1 * p_1, \ldots, c_{r-1} = k_{r-1} * p_{r-1} \end{cases} \tag{2.16}$$

$$nd^l : \begin{cases} f(\backslash_*, K)(c_0, c_1, \ldots, c_{r-1}) = p_0, p_1, \ldots, p_{r-1}, r \geq 1 \\ \text{where } p_0 = k_0 \backslash c_0, p_1 = k_1 \backslash c_1, \ldots, p_{r-1} = k_{r-1} \backslash c_{r-1} \end{cases} \tag{2.17}$$

$$ne^r : \begin{cases} f(*_/, K)(p_0, p_1, \ldots, p_{r-1}) = c_0, c_1, \ldots, c_{r-1}, r \geq 1 \\ \text{where } c_0 = p_0 * k_0, c_1 = p_1 * k_1, \ldots, c_{r-1} = p_{r-1} * k_{r-1} \end{cases} \tag{2.18}$$

$$nd^r : \begin{cases} f(/_*, K)(c_0, c_1, \ldots, c_{r-1}) = p_0, p_1, \ldots, p_{r-1}, r \geq 1 \\ \text{where } p_0 = c_0/k_0, p_1 = p_1/k_1, \ldots, p_{r-1} = c_{r-1}/k_{r-1} \end{cases} \tag{2.19}$$

Note that ne^l and nd^l transformations use the quasigroup $Q = (\mathbb{Z}_n, *)$ and its left inverse quasigroup $LIQ = (\mathbb{Z}_n, \backslash)$, respectively. Similarly, ne^r and nd^r transformations use the quasigroup $Q = (\mathbb{Z}_n, *)$ and its right inverse quasigroup $RIQ = (\mathbb{Z}_n, /)$, respectively.

Also, note that both ne^l and ne^r transformations use the same quasigroup $Q = (\mathbb{Z}_n, *)$, but for obtaining the value of c_i, ne^l-transformation evaluates the expression $k_i * p_i$ while ne^r-transformation evaluates the expression $p_i * k_i$, $i \geq 0$. Both of them are not the same. This is because a quasigroup is a non-commutative algebraic structure. So both ne^l and ne^r transformations would produce different results because $p_i * k_i \neq k_i * p_i$.

2.2.16 Optimal quasigroup

Definition 2.2.17. *An optimal quasigroup $Q = (\mathbb{Z}_{2^k}, *)$ is a groupoid which has the following properties:*

(i) Q must be a quasigroup (see Definition 2.2.1).

(ii) Each row or each column of Q must be an optimal S-box of $k \times k$ bits.

As of now, no algorithm has been developed for generating the optimal quasigroups of order $2^k, k \geq 4$. So, finding an optimal quasigroup of order 2^k is still a longstanding open problem. Note that each row of an optimal quasigroup is an optimal S-box. That is, an optimal quasigroup of order 2^k consists of 2^k optimal S-boxes of $k \times k$ bits. A $k \times k$ bits S-box is a Boolean map such as $S : \mathbb{F}_2^k \rightarrow \mathbb{F}_2^k$, where $\mathbb{F}_2$ is a Galois field over $\{0, 1\}$. In other words, an S-box is a permutation of the elements of $\mathbb{Z}_{2^k}$=$\{0, 1, 2, \ldots, 2^k - 1\}$. This thesis deals with forming an optimal quasigroup of order 16 and requires 16 optimal S-boxes of 4×4 bits. The description of a 4×4 bits optimal S-box is given in definition 2.2.18.

Definition 2.2.18. *A 4×4 bits S-box is said to be optimal if the following conditions are satisfied [44]:*

(i) S is a bijection,

(ii) Lin(S) = 8, and

(iii) Diff(S) = 4,

where Lin(S) and Diff(S) are the linearity and the differential characteristics of an S-box, and are defined as follows: Let $u = (u_0, u_1, \ldots, u_{k-1})$ and $v = (v_0, v_1, \ldots, v_{k-1})$ be two vectors, where both $u_i, v_i \in \mathbb{F}_2, 0 \leq i \leq k-1$. The dot product of u and v can be written as

$$u.v = \sum_{i=0}^{k-1} u_i.v_i,$$

then

$$Lin(S) = max\{|\mathbb{W}_S(u, v)| : u \in \mathbb{F}_2^k, v \in \mathbb{F}_2^k \, and \, v \neq 0\} \tag{2.20}$$

and

$$Diff(S) = max\{|\triangle_S\,(u, v)| : u \in \mathbb{F}_2^k, v \in \mathbb{F}_2^k \, and \, u \neq 0\}, \tag{2.21}$$

where

$$\mathbb{W}_S(u, v) = \sum_{x \in \mathbb{F}_2^k} (-1)^{u.x + v.S(x)}$$

and

$$\triangle_S\ (u, v) = \{x \in \mathbb{F}_2^k : S(x \oplus u) \oplus S(x) = v\},$$

$|\ldots|$ denotes the cardinality of a set, and $\oplus$ denotes bit-wise `XOR` (modulo 2) operation. An S-box's linearity and differential characteristic measure the resistance against the linear and differential cryptanalysis attacks, respectively. The smaller the linearity and the differential characteristic of an S-box, the more secure against these attacks.

Generation of the optimal quasigroups of order 16 is a hard problem since it consists of a total of 16 optimal S-boxes of 4×4 bits. We generated it based on 16 optimal S-boxes. Various algorithms that generate 4×4 bits optimal S-boxes exist in the literature [55, 84]. Note that all such S-boxes are not suitable for forming a quasigroup. This is because a quasigroup is a mathematical object and has specific properties that must be satisfied (see Definition 2.2.1). We have chosen 16 S-boxes, namely, $S_0, S_1, S_2, \ldots, S_{15}$ given in [83]. We verified that each of these 16 S-boxes is a bijection; that is, all the 16 S-boxes satisfy property (i) of Definition 2.2.18. For the linearity and the differential characteristic of S_0, for $k = 4$, we evaluated the values of $\mathbb{W}_{S_0}(u, v)$ and the values of $\triangle_{S_0}\ (u, v)$ using the equations Equation (2.20) and Equation 92.21), respectively, $\forall u, v \in \mathbb{F}_2^4$. The corresponding results are shown in Table 2.9 and Table 2.10, where u denotes the row number and v denotes the column number. We see that the maximum values in tables Table 2.9 and Table 2.10 are 8 and 4, respectively. So, $Lin(S_0) = 8$ and the $Diff(S_0) = 4$. That is, the S_0 S-box also satisfies both properties (ii) and (iii) of Definition 2.2.18. Hence S_0 is an optimal S-box. Similarly, we have also verified the remaining S-boxes, and we found that all of them satisfy all the required properties of an optimal S-box. Also, these 16 S-boxes are suitable to form an optimal quasigroup $Q = (\mathbb{Z}_{16}, *)$, shown in Table 2.8, where $*$ is a quasigroup operation corresponding to the quasigroup Q of order 16.

2.2.19 Quasigroups as vector valued Boolean functions

Quasigroups are suitable in cryptosystems because of their structure and their large number. A quasigroup with order 2^k can be represented as a vector valued Boolean function $f : \mathbb{F}_2^{2k} \to \mathbb{F}_2^k$, where $\mathbb{F}_2$ is a Galois field over $\{0, 1\}$. This representation can classify a quasigroup as linear or non-linear. A Boolean map $f : \mathbb{F}_2^{2k} \to \mathbb{F}_2^k$ is represented by k-tuple of polynomials in which each polynomial is a Boolean function

Table 2.8: Optimal quasigroup of order 16.

$*$		**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**
0	S_0 :	8	0	11	2	9	14	7	6	13	3	15	4	5	10	12	1
1	S_1 :	11	2	8	0	7	6	9	14	15	4	13	3	12	1	5	10
2	S_2 :	2	11	0	8	6	7	14	9	4	15	3	13	1	12	10	5
3	S_3 :	10	5	1	12	3	13	4	15	14	9	6	7	0	8	2	11
4	S_4 :	9	14	7	6	8	0	11	2	5	10	12	1	13	3	15	4
5	S_5 :	0	8	2	11	14	9	6	7	3	13	4	15	10	5	1	12
6	S_6 :	12	1	5	10	15	4	13	3	7	6	9	14	11	2	8	0
7	S_7 :	1	12	10	5	4	15	3	13	6	7	14	9	2	11	0	8
8	S_8 :	14	9	6	7	0	8	2	11	10	5	1	12	3	13	4	17
9	S_9 :	7	6	9	14	11	2	8	0	12	1	5	10	15	4	13	3
10	S_{10} :	3	13	4	15	10	5	1	12	0	8	2	11	14	9	6	7
11	S_{11} :	6	7	14	9	2	11	0	8	1	12	10	5	4	15	3	13
12	S_{12} :	5	10	12	1	13	3	15	4	9	14	7	6	8	0	11	2
13	S_{13} :	4	15	3	13	1	12	10	5	2	11	0	8	6	7	14	9
14	S_{14} :	15	4	13	3	12	1	5	10	11	2	8	0	7	6	9	14
15	S_{15} :	13	3	15	4	5	10	12	1	8	0	11	2	9	14	7	6

$f_i : \mathbb{F}_2^{2k} \to \mathbb{F}_2$, $1 \leq i \leq k$. A Boolean function $f_i : \mathbb{F}_2^{2k} \to \mathbb{F}_2$ can be uniquely written in its Algebraic Normal Form (ANF), as a polynomial in $2k$ variables as

$$f(x_1, x_2, \ldots, x_{2k}) = \sum_{I \subseteq \{1,2,\ldots,2k\}} \mathbb{C}_I x^I, \tag{2.22}$$

$$\text{where } x^I = \prod_{i \in I} x_i, x^\phi = 1, \text{ and } \mathbb{C}_I \in \{0, 1\}.$$

The algebraic degree of the Boolean map f is the maximum algebraic degree of its component functions $(f_1, f_2, \ldots, f_k)$. So,

$$deg(f) = max\{deg(f_i(x)) : x \in \mathbb{F}_2^{2k} \text{ and } x \neq 0,\ 1 \leq i \leq k\}. \tag{2.23}$$

If $deg(f) = 1$, then f is linear otherwise non-linear.

Let $Q = (\mathbb{Z}_{2^k}, *)$ be a quasigroup of order 2^k. Also, let $x, y \in \mathbb{Z}_{2^k}$, where $x = (x_1, x_2, \ldots, x_k) \in \mathbb{F}_2^k$ and $y = (y_{k+1}, y_{k+2}, \ldots, y_{2k}) \in \mathbb{F}_2^k$. Then, the representation of Q

Table 2.9: Results of $W_{S_0}(u, v)$ for the S-box S_0.

$u \setminus v$	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	-8	4	4	-4	4	0	0	-8	0	-4	-4	-4	4	0	0
2	0	4	-4	4	4	0	8	-4	-4	0	8	0	0	-4	4
3	0	8	-8	0	0	0	0	4	4	-4	-4	-4	-4	-4	-4
4	0	0	8	4	4	-4	4	0	8	0	0	4	-4	-4	-4
5	0	-4	-4	0	0	4	4	-8	0	4	-4	0	-8	4	-4
6	0	4	4	0	0	4	4	4	-4	-8	0	4	-4	8	0
7	8	0	0	-4	4	-4	-4	-4	-4	-4	4	0	0	0	-8
8	4	-4	0	4	0	8	4	0	4	-4	0	-4	8	0	-4
9	-4	0	4	8	-4	0	-4	0	-4	0	4	-8	-4	0	-4
10	4	8	4	0	-4	8	-4	-4	0	4	0	4	0	-4	0
11	4	-4	0	4	0	0	-4	-4	0	-8	-4	0	-4	-4	8
12	4	4	0	8	4	-4	0	0	-4	4	-8	0	4	4	0
13	4	0	4	-4	-8	-4	8	0	-4	0	-4	-4	0	-4	0
14	4	0	4	-4	8	4	0	4	0	4	0	-8	-4	0	4
15	-4	-4	0	0	4	4	0	4	-8	0	-4	4	0	-8	-4

as vector valued Boolean function is as follows:

$$x * y \equiv f(x_1, x_2, \ldots, x_{2k}) = f(f_1(x_1, x_2, \ldots, x_{2k}), f_2(x_1, x_2, \ldots, x_{2k}), \ldots, f_k(x_1, x_2, \ldots, x_{2k})),$$
$$\forall x, y \in \mathbb{Z}_{2^k}, \text{ and } f_i : \mathbb{F}_2^{2k} \to \mathbb{F}_2,\ 1 \leq i \leq k.$$

Now, a quasigroup Q is said to be linear if $deg(f_i) \leq 1, \forall\, 1 \leq i \leq k$; otherwise, Q is said to be a non-linear quasigroup. Note that the number of quasigroups grows exponentially as the value of k increases [79]. So, it is very difficult to identify all the linear and non-linear quasigroups of order 2^k. However, it is shown in [21] that for $k = 2$, there are 576 quasigroups of order 4, of which 144 are linear quasigroups, and 432 are non-linear quasigroups. The following example illustrates one such non-linear quasigroup of order 4 with degree 2.

Example 2.2.20. *Let $Q = (\mathbb{Z}_4, *)$ be a quasigroup of order 4 with mapping f:$\mathbb{F}_2^4 \to \mathbb{F}_2^2$. The quasigroup Q can be represented as a vector valued Boolean function $f(x_1, x_2, x_3, x_4) = (x_1 + x_2 + x_3, x_1x_3 + x_2x_3 + x_1 + x_3 + x_4 + 1)$. So, Q is a non-linear quasigroup with degree 2, its operation table is given in Table 2.11.*

Table 2.10: Results of $\triangle_{S_0}(u, v)$ for the S-box S_0.

$u \setminus v$	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**
1	0	2	0	0	0	0	0	2	2	2	0	2	0	2	2	2
2	0	0	4	2	0	0	0	2	2	2	0	2	0	0	2	0
3	0	0	0	0	2	0	2	0	0	4	2	2	2	0	0	2
4	0	2	0	2	2	2	0	0	2	2	0	0	2	0	2	0
5	0	0	0	0	0	2	4	2	2	2	0	0	0	2	2	0
6	0	2	4	0	0	0	2	0	0	0	2	0	2	0	2	2
7	0	2	0	0	0	4	0	2	0	0	0	2	2	0	2	2
8	0	0	0	2	4	2	2	2	0	0	0	2	2	0	0	0
9	0	0	0	2	0	0	2	0	0	0	2	4	0	4	0	2
10	0	2	2	0	2	2	2	2	0	0	0	0	2	0	0	2
11	0	0	2	2	0	0	0	0	4	0	0	0	2	2	0	4
12	0	0	2	2	2	0	0	2	2	0	2	0	0	4	0	0
13	0	0	2	4	0	2	0	0	0	2	4	0	0	0	2	0
14	0	2	0	0	2	2	2	0	2	0	4	0	0	0	2	0
15	0	4	0	0	2	0	0	2	0	2	0	2	2	2	0	0

.

2.2.21 Quasigroup generation

Generating all possible quasigroups of a given order is a hard problem. This is because as the order increases, the number of quasigroups grows exponentially [79]. Various algorithms exist in the literature [42, 54, 62], to generate the quasigroups of arbitrary order. Given a quasigroup, a new quasigroup can be generated by permuting rows, columns, and symbols of the given quasigroup. This is referred to as an isotopy or isotopism [42, 75], see the following definition.

Definition 2.2.22. *Let $Q_1 = (\mathbb{Z}_n, *_1)$ and $Q_2 = (\mathbb{Z}_n, *_2)$ be two quasigroups. An order triple of bijective mappings $\alpha, \beta, \gamma : \mathbb{Z}_n \to \mathbb{Z}_n$ is called an isotopism from Q_1 to Q_2, if* $\forall\, x, y \in \mathbb{Z}_n$

$$\alpha(x *_1 y) = \beta(x) *_2 \gamma(y).$$

It is noteworthy that an isotopism can allow a quasigroup Q_2 to be created from another quasigroup Q_1. This is illustrated in the following example.

Table 2.11: Non-linear quasigroup of order 4.

$*$	**0**	**1**	**2**	**3**
0	1	0	2	3
1	3	2	1	0
2	2	3	0	1
3	0	1	3	2

Example 2.2.23. *Let $Q_2 = (\mathbb{Z}_4, *_2)$ be a quasigroup of order 4 over the set $\mathbb{Z}_4 = \{0, 1, 2, 3\}$, whose operation table is shown in Table 2.13. Let the bijective mappings $\alpha, \beta, \gamma : \mathbb{Z}_4 \to \mathbb{Z}_4$ be defined as*

$$\alpha = \begin{pmatrix} 0 & 1 & 2 & 3 \\ 2 & 1 & 0 & 3 \end{pmatrix}, \qquad \beta = \begin{pmatrix} 0 & 1 & 2 & 3 \\ 1 & 2 & 3 & 0 \end{pmatrix}, \qquad \gamma = \begin{pmatrix} 0 & 1 & 2 & 3 \\ 3 & 0 & 2 & 1 \end{pmatrix}$$

Then the new quasigroup $Q_1 = (\mathbb{Z}_4, *_1)$ is defined by $x *_1 y = \alpha^{-1}(\beta(x) *_2 \gamma(y))$, which is isotopism to $Q_2 = (\mathbb{Z}_4, *_2)$, shown in Table 2.12.

Table 2.12: Quasigroup Q_1

$*_1$	**0**	**1**	**2**	**3**
0	0	1	3	2
1	1	3	2	0
2	3	2	0	1
3	2	0	1	3

Table 2.13: Quasigroup Q_2

$*_2$	**0**	**1**	**2**	**3**
0	2	1	0	3
1	1	3	2	0
2	3	0	1	2
3	0	2	3	1

Claim 2.2.24. *If Q is a quasigroup of order n. Then, a maximum of $n!^2$ quasigroups can be created by permuting both the rows and columns of Q, where $n \geq 4$.*

Proof. Let $Q = (\mathbb{Z}_n, *)$ be a quasigroup over the set $\mathbb{Z}_n = \{0, 1, \ldots, n-1\}$. It can be represented as an $n \times n$ matrix (for example, a 4 order quasigroup is shown in Table 2.11), and consists of n number of rows and n number of columns, where each row and each column is a permutation of $\mathbb{Z}_n = \{0, 1, \ldots, n-1\}$.

Now, by permuting only rows of quasigroup Q, at most $n!$ quasigroups can be created. Similarly, by permuting only columns of quasigroup Q, at most $n!$ quasigroups can also be created. So, by permuting the rows and the columns of the quasigroup Q, a total of $n!^2$ quasigroups can be created. This is because, for each row permutation of Q, a maximum of $n!$ quasigroups can be formed by permuting the n columns of Q. □

The above claim is helpful in designing new cryptosystems that use multiple quasigroups. Also, the new cryptosystems can employ multiple quasigroups by leveraging the space of a single quasigroup. In other words, for the design of new cryptosystems based on multiple quasigroups, it is not necessary to store all quasigroups since they can be generated by permuting rows, columns, or both of an original quasigroup. Next section gives an algorithm to generate n quasigroups of order $n = 2^k$ by permuting the rows of an original quasigroup $Q = (\mathbb{Z}_{2^k}, *)$.

2.2.25 Quasigroup generation based on row permutations

Let $Q = (\mathbb{Z}_{2^k}, *)$ be an original quasigroup of order $n = 2^k$ over the set $\mathbb{Z}_{2^k} = \{0, 1, \ldots, n-1\}$. Since the quasigroup Q is represented by an $n \times n$ matrix, where each row and each column is a permutation of the elements of $\mathbb{Z}_{2^k}$. Let $S_0, S_1, \ldots, S_{n-1}$ be the rows of Q, where each S_i is a permutation of the elements of $\mathbb{Z}_{2^k}, 0 \leq i \leq n-1$. Then, the quasigroup Q can also be represented as $Q = (S_0, S_1, \ldots, S_{n-1})$. For instance, a quasigroup $Q = (S_0, S_1, \ldots, S_{15})$ of order 16 is shown in Table 2.8.

By permuting the rows of an order n quasigroup Q, $n!$ quasigroups can be created. These $n!$ quasigroups are nothing but permutations of $S_0, S_1, S_2, \ldots,$ and S_{n-1} rows. Note that here we have specified only n quasigroups to be generated. In other words, we have to generate or select any n quasigroups out of the total $n!$ quasigroups. Let the generated or selected quasigroups be denoted by $Q_0 = (\mathbb{Z}_n, *_0), Q_1 = (\mathbb{Z}_n, *_1), ..., Q_{n-1} = (\mathbb{Z}_n, *_{n-1})$, where $*_0, *_1, ..., *_{n-1}$ are the quasigroup operations corresponding to $Q_0, Q_1, ..., Q_{n-1}$, respectively. The selection of quasigroups can be carried out using an $n \times 1$ multiplexer, where $n = 2^k$. A multiplexer is a combinational circuit that has a maximum of 2^k input values for k selection lines, and it produces a single output. An $n \times 1$ multiplexer, along with its truth table, are shown in Figure 2.1. This multiplexer selects a quasigroup based on the current state of the selection lines $s_0, s_1, \ldots, s_{k-1}$, the value of s_α is either 0 or 1, $0 \leq \alpha \leq k-1$. Let `QGSELECT` be a k-bit value associated with the selection lines s_0, s_1, ..., and s_{k-1}, where s_0 and s_{k-1} are the least significant bit and the most significant bit, respectively. That is, if $s_\alpha = 0$, $\forall\ 0 \leq \alpha \leq k-1$, then `QGSELECT` $= 0$, and the multiplexer selects a quasigroup Q_0. Similarly, if `QGSELECT` $= 1$, then the multiplexer selects the quasigroup Q_1, and so on. So, based on the original quasigroup $Q = (S_0, S_1, \ldots, S_{n-1})$, the following equation can

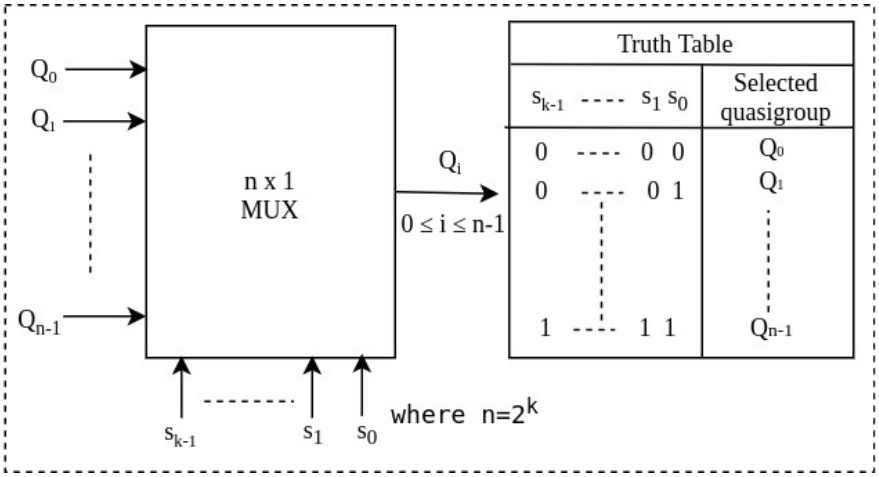


Figure 2.1: $n \times 1$ multiplexer and its truth table.

generate a total of n quasigroups

$$Q = (S_{(0+\texttt{QGSELECT}) \bmod n}, S_{(1+\texttt{QGSELECT}) \bmod n}, \ldots, S_{(n-1+\texttt{QGSELECT}) \bmod n}) \tag{2.24}$$

where QGSELECT, $0 \leq$ QGSELECT $\leq n-1$ is a k-bit value to be given as an input. Note that the result is a quasigroup $Q \in \{Q_0, Q_1, \ldots, Q_{n-1}\}$. Each of these $\{Q_0, Q_1, \ldots, Q_{n-1}\}$ quasigroups consists of the same rows $S_0, S_1, \ldots, S_{n-1}$ but in different order (permutation).

Let $LIQ = (\mathbb{Z}_n, \backslash)$ be a left inverse quasigroup of the quasigroup $Q = (\mathbb{Z}_n, *)$. Also, let $LIQ_0, LIQ_1, \ldots, LIQ_{n-1}$ be the quasigroups generated from the left inverse quasigroup LIQ using Equation (2.24). Then, each LIQ_i is also the left inverse quasigroup of the quasigroup Q_i, $0 \leq i \leq n-1$. The correctness of this assertion follows from the following theorem.

Theorem 1. *Let Q and LIQ be a quasigroup and its left inverse quasigroup, respectively. Let Q_i, and LIQ_i be the result of applying a permutation $\mathcal{P}$ on the rows of Q and LIQ, respectively. Then, LIQ_i is the left inverse quasigroup of the quasigroup Q_i.*

Proof. Let $Q = (\mathbb{Z}_n, *)$ and $LIQ = (\mathbb{Z}_n, \backslash)$ be quasigroups of order n, whose operation tables look like the ones given in Table 2.14 (a) and Table 2.14 (b), respectively. Since LIQ is the left inverse quasigroup of Q, we have

$$t_1 * t_2 = a_{t_1,t_2} \Leftrightarrow t_1 \backslash a_{t_1,t_2} = b_{t_1,t_2}$$
$$\forall\, t_1, t_2 \in \{0, 1, \ldots, n-1\}.$$

Now let

$$\mathcal{P} = \begin{pmatrix} \mathcal{P}(0) & \mathcal{P}(1) & \mathcal{P}(2) & \mathcal{P}(2) & \cdots & \mathcal{P}(n-1) \\ 0 & 1 & 2 & 3 & \cdots & n-1 \end{pmatrix}$$

be a permutation applied on the rows of the quasigroups Q and LIQ, and let $Q_i = (\mathbb{Z}_n, *_i)$ and $LIQ_i = (\mathbb{Z}_n, \backslash_i)$ denote the resulting quasigroups, where $\mathcal{P}(j)^{th}$ rows of both the Q and the LIQ becomes the j^{th} rows of both the Q_i and the LIQ_i, respectively, $0 \leq j, \mathcal{P}(j) \leq n-1$. That is, the operation tables of Q_i and LIQ_i will be as shown in Table 2.15 (a) and Table 2.15 (b), respectively. Therefore,

$$t_1 *_i t_2 = a_{\mathcal{P}(t_1),t_2} \Leftrightarrow t_1 \backslash_i a_{\mathcal{P}(t_1),t_2} = b_{\mathcal{P}(t_1),t_2}$$
$$\forall\, t_1, t_2 \in \{0, 1, \ldots, n-1\}.$$

This is true because

$$t_1 * t_2 = a_{t_1,t_2} \Leftrightarrow t_1 \setminus a_{t_1,t_2} = b_{t_1,t_2}$$
$$\forall\, t_1, t_2 \in \{0, 1, \ldots, n-1\}.$$

□

Table 2.14: Representation of quasigroups Q and LIQ of order n.

$*$	**0**	**1**	...	$n-1$	$\backslash$	**0**	**1**	...	$n-1$
0	$a_{0,0}$	$a_{0,1}$	...	$a_{0,n-1}$	**0**	$b_{0,0}$	$b_{0,1}$	...	$b_{0,n-1}$
1	$a_{1,0}$	$a_{1,1}$	...	$a_{1,n-1}$	**1**	$b_{1,0}$	$b_{1,1}$	...	$b_{1,n-1}$
2	$a_{2,0}$	$a_{2,1}$	...	$a_{2,n-1}$	**2**	$b_{2,0}$	$b_{2,1}$	...	$b_{2,n-1}$
⋮	⋮	⋮	⋮	⋮	**⋮**	⋮	⋮	⋮	⋮
$n-1$	$a_{n-1,0}$	$a_{n-1,1}$	...	$a_{n-1,n-1}$	$n-1$	$b_{n-1,0}$	$b_{n-1,1}$	...	$b_{n-1,n-1}$
(a)					(b)				

The application of Theorem 1 is illustrated in Example 2.2.26.

Example 2.2.26. *Let quasigroup $Q = (\mathbb{Z}_8, *)$ over $\mathbb{Z}_8 = \{0, 1, 2, 3, 4, 5, 6, 7\}$ is defined as $t_1 * t_2 = (t_1 + t_2)$ mod 8, where $t_1, t_2 \in \mathbb{Z}_8$. Then its operation table is as shown in Table 2.16 (a). Also, let $LIQ = (\mathbb{Z}_8, \backslash)$ be a left inverse quasigroup of the quasigroup Q, and its operation be as shown in Table 2.16 (b). Now, let*

$$\mathcal{P} = \begin{pmatrix} 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 \\ 1 & 0 & 3 & 2 & 5 & 4 & 7 & 6 \end{pmatrix}$$

Table 2.15: Representation of quasigroups Q_i and LIQ_i of order n.

$*_i$	**0**	**1**	...	$n-1$
0	$a_{\mathcal{P}(0),0}$	$a_{\mathcal{P}(0),1}$	...	$a_{\mathcal{P}(0),n-1}$
1	$a_{\mathcal{P}(1),0}$	$a_{\mathcal{P}(1),1}$	...	$a_{\mathcal{P}(1),n-1}$
2	$a_{\mathcal{P}(2),0}$	$a_{\mathcal{P}(2),1}$	...	$a_{\mathcal{P}(2),n-1}$
⋮	⋮	⋮	⋮	⋮
$n-1$	$a_{\mathcal{P}(n-1),0}$	$a_{\mathcal{P}(n-1),1}$	...	$a_{\mathcal{P}(n-1),n-1}$
(a)				

$\backslash_i$	**0**	**1**	...	$n-1$
0	$b_{\mathcal{P}(0),0}$	$b_{\mathcal{P}(0),1}$	...	$b_{\mathcal{P}(0),n-1}$
1	$b_{\mathcal{P}(1),0}$	$b_{\mathcal{P}(1),1}$	...	$b_{\mathcal{P}(1),n-1}$
2	$b_{\mathcal{P}(2),0}$	$b_{\mathcal{P}(2),1}$	...	$b_{\mathcal{P}(2),n-1}$
⋮	⋮	⋮	⋮	⋮
$n-1$	$b_{\mathcal{P}(n-1),0}$	$b_{\mathcal{P}(n-1),1}$	...	$b_{\mathcal{P}(n-1),n-1}$
(b)				

be a permutation applied on the rows of the quasigroup Q, and let $Q_i = (\mathbb{Z}_8, *_i)$ denote the resulting quasigroup. Then Q_i is as shown in Table 2.17 (a), where $*_i$ is the quasigroup operation corresponding to the quasigroup Q_i. Now, we applied the same rows permutation $\mathcal{P}$ on the left inverse quasigroup LIQ, and let $LIQ_i = (\mathbb{Z}_8, \backslash_i)$ denote the resulting quasigroup. Then LIQ_i is as shown in Table 2.17 (b), where $\backslash_i$ is the left inverse quasigroup operation corresponding to the quasigroup LIQ_i. It can be verified that the quasigroup LIQ_i is the left inverse quasigroup of the quasigroup Q_i.

Table 2.16: Quasigroups Q and LIQ of order 8.

$*$	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**
0	0	1	2	3	4	5	6	7
1	1	2	3	4	5	6	7	0
2	2	3	4	5	6	7	0	1
3	3	4	5	6	7	0	1	2
4	4	5	6	7	0	1	2	3
5	5	6	7	0	1	2	3	4
6	6	7	0	1	2	3	4	5
7	7	0	1	2	3	4	5	6
(a)								

$\backslash$	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**
0	0	1	2	3	4	5	6	7
1	7	0	1	2	3	4	5	6
2	6	7	0	1	2	3	4	5
3	5	6	7	0	1	2	3	4
4	4	5	6	7	0	1	2	3
5	3	4	5	6	7	0	1	2
6	2	3	4	5	6	7	0	1
7	1	2	3	4	5	6	7	0
(b)								

Table 2.17: Quasigroups Q_i and LIQ_i of order 8.

$*_i$	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**
0	1	2	3	4	5	6	7	0
1	0	1	2	3	4	5	6	7
2	3	4	5	6	7	0	1	2
3	2	3	4	5	6	7	0	1
4	5	6	7	0	1	2	3	4
5	4	5	6	7	0	1	2	3
6	7	0	1	2	3	4	5	6
7	6	7	0	1	2	3	4	5

(a)

$\backslash_i$	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**
0	7	0	1	2	3	4	5	6
1	0	1	2	3	4	5	6	7
2	5	6	7	0	1	2	3	4
3	6	7	0	1	2	3	4	5
4	3	4	5	6	7	0	1	2
5	4	5	6	7	0	1	2	3
6	1	2	3	4	5	6	7	0
7	2	3	4	5	6	7	0	1

(b)

Chapter 3

Literature survey

Various applications of the quasigroup can be found in cryptography, coding theory, design theory, and other fields. Our research focuses on the use of quasigroups in cryptography. Because of their structure, characteristics, and an exponential number of quasigroups of its order, quasigroups are ideal for cryptographic applications. The effects of quasigroups in cryptography are primarily determined by the quasigroup chosen. So, one of the issues is determining which quasigroup is appropriate for use and what preconditions the quasigroup must meet. In this chapter, we present a comprehensive survey of the existing quasigroup based cryptographic algorithms such as stream ciphers, block ciphers, hash functions, MACs, etc.

3.1 Stream ciphers based on quasigroup

We have studied various quasigroup based stream ciphers present in literature [12, 28, 43, 46, 59, 60, 81]. The quasigroup based stream ciphers allow us to make polyalphabetic substitution ciphers, and this property has been used in the design of stream ciphers for more than 400 years. In 1586, Blaise de Vigenere proposed the first polyalphabetic stream cipher based on the letters of a keyword, now known as the Vigenere cipher. This cipher uses a Latin square of the same order as his target language (i.e., for the English language, the order of a Latin square would be 26). This cipher was thought to be impenetrable until 1863 when it was discovered that the ciphertext showed repetitions for a sufficiently large plaintext.

In 1996, Czeslaw Koscielny showed how a quasigroup can be used in the design of a stream cipher [41]. In 1997, Markovski et al. proposed a quasigroup based stream cipher [46]. This cipher is designed based on $\{e, d\}$-transformation and uses a leader ℓ (also called a seed value) along with a quasigroup of an arbitrary order n for encrypting/decrypting the plaintext/ciphertext. Note that the leader and the quasigroup are secrets in this cipher, because of the leader ℓ and $\{e, d\}$-transformation the cipher is found to be vulnerable to chosen-plaintext, chosen-ciphertext, and known-plaintext attacks [35, 45, 80]. This is because, it creates a relation between the plaintext and the corresponding ciphertext since it uses the output of each e-transformation as a key (or keystream) for encrypting the next character of the plaintext. For a known-plaintext attack, an attacker has the knowledge of the plaintext $P = \{p_1, p_2, \ldots, p_k\}$ and the corresponding ciphertext $C = \{c_1, c_2, \ldots, c_k\}$. In this attack, an attacker uses the e-transformation and retrieves both the leader ℓ and the quasigroup employed. This is illustrated in the following example.

Example 3.1.1. *Let Q=($\mathbb{Z}_4, *$) be a quasigroup over $\mathbb{Z}_4 = \{0, 1, 2, 3\}$. Also, let an attacker knows a plaintext $P = 23012312123000312$ and the corresponding ciphertext $C = 13322110030120231$. Then, an attacker carries out the operations except the first one involving the leader using the e-transformation defined in section 2.2.12 of Chapter 2 and reconstructs the quasigroup.* $1*3 = 3$, $3*0 = 3$, $3*1 = 2$, $2*2 = 2$, $2*3 = 1$, $1*1 = 1$, $1*2 = 0$, $0*1 = 0$, $0*2 = 3$, $3*3 = 0$, $0*0 = 1$, $1*0 = 2$, $2*0 = 0$, $0*3 = 2$, $2*1 = 3$, $3*2 = 1$. *Note that the first equation $1*3 = 3$ is obtained by looking at the first element of the ciphertext, the second element of the plaintext, and the second element of the ciphertext. Similarly, the second equation is obtained by taking the second element of the ciphertext, the third element of the plain text, and the third element of the ciphertext. In general, the r^{th} equation is formed from the r^{th} element of the ciphertext, $(r+1)^{th}$ element of the plaintext, and the $(r+1)^{th}$ element of the ciphertext. The resultant quasigroup is shown in Table 3.1. Once a quasigroup is reconstructed, the leader ℓ can be uniquely determined by solving the following equation*

$$\ell * p_1 = c_1. \tag{3.1}$$

Since $p_1 = 2$, $c_1 = 1$, then the leader $\ell = 3$. Hence, the $\{e, d\}$-transformation based stream ciphers that use a single quasigroup are vulnerable to the known-plaintext attack. Similar argument shows that the system is also vulnerable to the chosen-plaintext (or

chosen-ciphertext) attack. In this attack, an attacker can choose the plaintext (ciphertext), gets the corresponding ciphertext (plaintext), and solves for the quasigroup used and hence the leader of the scheme.

Table 3.1: Quasigroup of order 4

$*$	**0**	**1**	**2**	**3**
0	1	0	3	2
1	2	1	0	3
2	0	3	2	1
3	3	2	1	0

Attack complexity of known-plaintext attack: The complexity of the attack on a $\{e, d\}$-transformation based stream cipher with a single quasigroup of order n is equivalent to the computational complexity for determining the quasigroup used. This is evident from Example 3.1.1. The number of equations required to determine a quasigroup of order n can be seen to be n^2. Hence the computational complexity of determining the quasigroup used and hence that of the attack is $O(n^2)$. In general, the attack complexity for order n quasigroup is equal to $2(n-1)^2$ [45]. Hence, the existing cipher can be cryptanalyzed in polynomial time.

In [12, 59, 60], the authors discussed stream ciphers based on the n-quasigroup (also called n-ary quasigroup operations) of an arbitrary order m. The description of an n-quasigroup is given in Definition 2.2.10 of Chapter 2. They analyzed their ciphers only against the exhaustive key search attack and tried to determine the size of the key space of their proposals. For practical purposes, they considered the values of n and m to be at least 3 and 256, respectively. So, if $n = 3$ and $m = 256$, the size of the key space of the stream ciphers presented in [59, 60] is found to be around $2^{65} \approx 3.69 \times 10^{19}$, while that of an improved 3-quasigroup based stream cipher presented in [12] is found to be around $2^{195} \approx 5.02 \times 10^{58}$. We noted that the value of n directly affects the software performance of these ciphers. In other words, as the value of n increases, the software performance decreases. Also, the ciphers need at least $64K$ bytes of extra space, this may be a challenge for small computing devices.

In 2008, Gligoroski et al. introduced an Edon-80 stream cipher [28]. It is an XOR-based (a binary additive) synchronous stream cipher that was one of the candidates in the eSTREAM project. The block diagram of Edon-80 is given in Figure 3.1. The

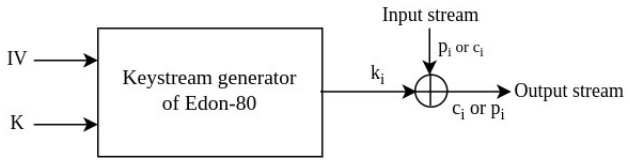


Figure 3.1: Block diagram of Edon-80.

keystream generator of Edon-80 takes two inputs (i) a secret key K of 80 bits, and (ii) an initial value IV of 64 bits. And it produces keystream $(k_1, k_2, \ldots)$ of required length. Then, the keystream is added to the input stream (plaintext/ciphertext) in the encryption/decryption process. The internal structure of Edon-80 is a pipelined architecture that consists of 80 stages [28]. This is carried out using 80 e-transformations based on 4 quasigroups of order 4. The primary strength of the Edon-80 is that it is hardware friendly as it uses only 4 quasigroups of order 4. In contrast, its software performance is significantly decreased on modern CPUs. The cipher is analyzed against various attacks, including exhaustive key search attack, related key attack, and guess-and-verify attack, and found it to be resistant to these attacks. However, it is vulnerable to reused key and known-plaintext attacks. This is because it uses the XOR function to mix the plaintext/ciphertext with the keystream. Hell and Johansson have done one of the best attacks on Edon-80, called key recovery attack [36]. And they managed to recover the key in 2^{72} operations by analyzing the periods of the keystream sequences of Edon-80.

In 2012, Zhang and Xu proposed a new version of Edon-80 stream cipher based on an arbitrary period length keystream sequence [81]. The purpose of proposing this new version of the Edon-80 was to enhance the security of the Edon-80 against key recovery attack. The keystream generation algorithm of the new cipher uses 80 e-transformations along with a quasigroup of order 256, and produces keystream of required length. Also, the cipher uses the quasigroup operation instead of the XOR function, such as used in Edon-80, to mix the plaintext/ciphertext with the keystream, making the cipher resistant to a reused key attack. The cipher is also analyzed against various attacks, including exhaustive key search attack, chosen-plaintext attack, known-plaintext attack. We noted that cipher is resistant to these attacks, but it is found to be inefficient, and the cipher needs at least $256K$ bytes of extra space.

Lakshmi et al. introduced a synchronous stream cipher based on a quasigroup of order 256 [43]. The cipher takes a secret key and an initialization vector as its input,

each of 128 bits in size. The algorithm of keystream generation is iterative and uses 3-quasigroup operations. For each iteration, the algorithm generates 8 bits (one byte) of the keystream. Then, each byte of the generated keystream is added to one byte of plaintext/ciphertext in the encryption/decryption process. The security of the cipher is analyzed only using an algebraic attack. The randomness of the obtained ciphertext is tested using the NIST-STS test suite and various structural tests and observed that the ciphertext passes both these tests. Since it is an XOR-based synchronous stream cipher that uses a quasigroup of order 256 for generating the keystream, it is vulnerable to known-plaintext and reused-key attacks.

3.2 Block ciphers based on quasigroup

A block cipher based on quasigroups was invented by Gligorovsky and Markovsky [29]. They tried to show the potential of $\{e, d\}$-transformation as a new paradigm in cryptography. As in stream ciphers, the design of a new block cipher relies on $\{e, d\}$-transformation. Typically, in a stream cipher, the $\{e, d\}$-transformation is applied only once to the input stream (plaintext/ciphertext) to produce the corresponding output stream (ciphertext/plaintext). But, in a block cipher, the $\{e, d\}$-transformation is applied more than once times on the input stream to obtain the output stream. Let $P = \{p_1, p_2, \ldots, p_n\}$ and $C = \{c_1, c_2, \ldots, c_n\}$ be the input stream and output stream blocks of length n, respectively. Also, let $L = \{\ell_1, \ell_2, \ldots, \ell_k\}$ be a given password (or a sequence of leaders) of length k. Based on given quasigroups and their left/right inverse quasigroups, the encryption and decryption algorithms can be performed, as shown in Algorithms 1 and 2, respectively. The description of $\{e, d\}$-transformation is specified in section 2.2.12 of Chapter 2.

Algorithm 1: Encryption

```
[1] C = P
[2] for i = 1 to k do
[3]     if L[i] mod 2 = 0 then
[4]         C = (e_ℓ-transformation on C with leader ℓ = L[i])
[5]     else
[6]         C = (d_ℓ-transformation on C with leader ℓ = L[i])
```

Algorithm 2: Decryption

```
[1] P = C
[2] for i = k down to 1 do
[3]     if L[i] mod 2 = 0 then
[4]         P = (d_ℓ-transformation on P with leader ℓ = L[i])
[5]     else
[6]         P = (e_ℓ-transformation on P with leader ℓ = L[i])
```

In 2012 and 2013, Battey and Parakh introduced two block ciphers based on a quasigroup of order 256 [5, 6]. These ciphers use a sequence of leaders (a seed value) of 256 bits, and perform a total of 32 rounds to encrypt or decrypt a block of 128 bits, where for each round, they perform two transformations: (i) $\{e, d\}$-transformation, and (ii) left shift operation. Note that they employed the same encryption/decryption algorithms in both their proposals, and a randomly chosen quasigroup of order 256. A primary strength of these ciphers is that they are resistant to quasigroup only attack (exhaustive quasigroup search attack) due to a large number of quasigroups of its order. But, a randomly chosen quasigroup may not be optimal from linear and differential cryptanalysis. Also, it may be a challenge to store it in small computing devices. These existing block ciphers are only analyzed against the randomness property of the ciphers using the NIST-STS test suite. Since the NIST-STS test suite only evaluates the system's randomness, it does not provide security strength against attacks.

Another $\{e, d\}$-transformation based block cipher is introduced by Zhao and Xu in 2017 [83]. It uses an optimal quasigroup of order 16 along with a secret key of 80 or 128 bits, and perform 32 rounds to encrypt or decrypt a block of 64 bits. This cipher is also analyzed against (i) the randomness property of the cipher using the NIST-STS test suite and (ii) the algebraic properties of the optimal quasigroup used. It is not analyzed on the basis of the overall structure of the cipher. Noted that it uses less space than [5, 6]. We analyzed the software performance of the cipher by varying the inputs, and we observed that the cipher is slower than those of the existing ciphers [5, 6]. Also, the cipher does not exhibit good confusion and diffusion effects.

In 2014, Markovski et al. introduced a block cipher based on matrix presentation of quasigroups, named BCMPQ [50]. The cipher uses 128 quasigroups of order 4 in

both encryption and decryption. In encryption, the quasigroup operation $*$ of order 4 quasigroup $(Q, *)$ is defined as

$$x * y = m^t + Ax^t + By^t + CAx^t \cdot CBy^t$$

where, $A = \begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix}$ and $B = \begin{bmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \end{bmatrix}$ are non-singular Boolean matrices and $C = \begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix}$, both $x = (x_1, x_2), y = (y_1, y_2) \in Q$, all $x_1, x_2, y_1, y_2 \in \{0, 1\}, m = [m_1, m_2]$ is a Boolean vector. The operation '$\cdot$' denotes the scalar product, and the addition and multiplication are over the field $GF(2)$.

In decryption the left inverse quasigroup $(Q^{-1}, \backslash)$ of the quasigroup $(Q, *)$ is used, which is defined as

$$x \backslash z = B^{-1}m^t + B^{-1}(I + C)Ax^t + B^{-1}(Cm^t \cdot CAx^t) + B^{-1}z^t + B^{-1}(CAx^t \cdot Cz^t)$$

where $\backslash$ denotes the left inverse quasigroup operation corresponding to the left inverse quasigroup Q^{-1}, and I denotes the identity matrix. The aim of this work is to show how small quasigroups (quasigroups of order 4) can be used in a block cipher. Also, this cipher does not require too much space or computational power, so it is suitable for lightweight cryptographic applications and can be implemented and used for small devices. The cipher is analyzed against brute-force attacks and it is found to have a complexity of 2^{82}. Also, the randomness properties of this cipher are analyzed using the avalanche effect and strict avalanche criterion and examined the input and output bits of the BCMPQ [20].

3.3 Hash functions based on quasigroup

At first, in 2001 and 2002, Dvorsky et al. introduced how quasigroup can be used to create a cryptographic hash function [23, 24]. These hash functions do not have actual implementations. Later, in 2009, they proposed a quasigroup based hash function presented in [70]. This hash function, namely $\mathbb{H}$, uses an e-transformation (defined in section 2.2.12 of Chapter 2) on the message $m_1, m_2, \ldots, m_k$ to be hashed, such that

$$m'_1 = IV * m_1, m'_2 = m'_1 * m_2, \ldots, m'_k = m'_{k-1} * m_k,$$

where $m'_1, m'_2, \ldots, m'_k$ are the results of the e-transformation corresponding to $m_1, m_2, \ldots, m_k$. Both $(m'_1, m'_2, \ldots, m'_k)$ and $(m_1, m_2, \ldots, m_k)$ belong to a chosen quasigroup Q, and IV is an initial (seed) value also belongs to Q. Then, the hash function $\mathbb{H}$ with IV is defined as

$$\mathbb{H}_{IV}(m_1, m_2, \ldots, m_k) = (\ldots((IV * m_1) * m_2) * \ldots) * m_k.$$

In 2010, Slaminkova et al. analyzed this hash function and found that it is vulnerable to prefix and suffix attacks [69]. That is, an attacker can find a collision by creating a false message by adding a prefix or suffix to the original message. This is illustrated through the following example.

Example 3.3.1. *Let $M_1 = 102201$ be a message to be hashed with the initial value $IV = 3$ Also, let the quasigroup Q be one given in Table 3.1. Then, the hash value of M_1 is calculated as follows:*

$$\mathbb{H}_3(102201) = (((((3 * 1) * 0) * 2) * 2) * 0) * 1 = 3.$$

Now, an attacker can create a false message M_2 by adding a prefix to the original message M_1, and it can be written as $M_2 = X_1X_2 \ldots X_k102201, X_i \in Q, 1 \leq i \leq k$. The prefix is chosen such that $(\ldots((IV * X_1) * X2)\ldots) * X_k = IV$. For $IV = 3$ and $k = 4$, one set of values for X_1, X_2, X_3, X_4 is $1, 2, 0, 2$. That is, the new message $M_2 = 1202102201$ will have the same hash value as that of the original message M_1. This is verified as in the following:

$$\mathbb{H}_3(1202102201) = (((((((((3 * 1) * 2) * 0) * 2) * 1) * 0) * 2) * 2) * 0) * 1 = 3.$$

That is, for two different messages M_1 and M_2, $\mathbb{H}_3(M_1) = \mathbb{H}_3(M_2)$. That is a collision can be found with the help of a prefix attack. Hence, the existing hash function based on quasigroups is vulnerable to a collision attack. A similar argument can be shown that the system is also vulnerable to the suffix attack. In this attack, an attacker can create a false message M_2 by appending a suffix to the original message M_1 and arrives at a collision.

This hashing technique is very useful for small-size hash values. But for larger hash values, storing the quasigroup in the existing quasigroup based hash function can be a challenge. This is because, for n bits hash value, the order of the quasigroup needed would be 2^n and hence the storage requirement of such a quasigroup is $n \times 2^n \times 2^n$

bits. In Example 3.3.1, the hash value is of 2 bits and the storage requirements of the quasigroup (Table 3.1) used in this example is $2 \times 2^2 \times 2^2$ bits or 4 bytes. If the size of a hash value is 18 bits, then the storage requirements of a corresponding quasigroup will be more than 1 TB. Note that, the size of the hash being used nowadays is more than 128 bits. One of the special quasigroups that defined in [24], called the modular subtraction quasigroup, is used to solve this problem. In this thesis, we also overcome these shortcomings of the existing quasigroup based hash function by proposing a novel hash function of 224 (384) bits.

In 2006, Gligoroski et al. described a generic hash function based on quasigroup reverse string transformation [26]. Edon-F is another generic hash function based on quasigroup, introduced in [49] without implementation. In 2008, Gligoroski et al. introduced the first implementation of the generic hash function named Edon-R(256, 384, 512) [27]. Edon-R is a family of hash functions based on Merkle-Damgard straightening, and they are the wide-pipe iterative hash functions. In 2008, Edon-R was the first round fastest candidate of the NIST SHA-3 competition. The compression function of Edon-R works based on two strings, in which one consists of length $2n$ and another of length n, where n is the size of the hash value in w-bit words, and the size of w varies according to the hash size. In 2009, Khovratovich et al. investigated Edon-R against various attacks and found that all primary three attacks, such as pre-image, second pre-image, and collision attacks, can be applied with minimal effort [39]. This is a free-start attack scheme with minimal changeable initial chaining values. In these attacks, the asymmetrical diffusion of the chaining values in the compression function is exploited. A meet-in-the-middle attack on Edon-R to find real pre-images was launched by partially reversing the compression function and fixing one component of the chaining value.

In 2008, Markovski and Mileva proposed another family of hash functions based on quasigroup, named NaSHA-(m, k, r) [47], where m and r are positive integers, denoting the length of the message digest and in the form of order 2^{2^r} quasigroup respectively, and k is a positive even integer that denotes the number of elementary quasigroup string transformations. It is also the wide-pipe iterative hash function and uses Merkle-Damgard straightening. For $k = 2$, $r = 6$, and $m \in \{224, 256, 384, 512\}$, the implementation of NaSHA-(m, k, r) is discussed in literature [47, 48]. Gligoroski

and Knapskog analyzed NaSHA-(m, k, r) hash family against various attacks, including length extension attack, Joux's multicollision attack [37], 2^{nd} collision attack, and Kelsey and Schneier's long message 2^{nd} preimage attack [40]. The free-start preimage attack on NaSHA-m with the complexity of $2^{m/2}$ and free-start collision attacks on NaSHA-(m, k, r) with the complexity of 2^{32} are given by Nikolic and Knovratovich [57].

Chapter 4

Stream Ciphers based on Quasigroup

Three types of stream ciphers are discussed in this chapter. All these proposed stream ciphers provide a high level of security. This is because they use a non-associative algebraic structure called a quasigroup. Each of these ciphers differs slightly from the other in terms of time and space complexities. Each cipher is a symmetric key cipher and uses the same keystream for encryption/decryption. They use a quasigroup and its inverse quasigroup in encryption and decryption, respectively. The size of the keystream is the same as that of the plaintext/ciphertext and it is generated using a secret key. Both the secret key and the quasigroup are kept secret, so ciphers provide excellent security. In this chapter, we give a brief overview of the proposed stream ciphers, define the basic structure and the building elements of the proposed stream ciphers, and analyze the performance and perform security analyses of the proposed stream ciphers.

One of the goals of the proposed new stream ciphers is to overcome the issue of the two-time pad in the existing XOR-based stream ciphers. Two-time pad is nothing but a reused key attack. We know that an XOR-based stream cipher provides perfect security against a one-time pad. But, the keystream should be as long as the plaintext/ciphertext, which increases the difficulty of key management and key distribution.

4.1 Introduction

A stream cipher is a symmetric key cipher that uses the same key for encryption and decryption. It uses a pseudo-random keystream sequence to generate the ciphertext, which encrypts individual characters (typically binary digits) of a plaintext one at a time. A block cipher, on the other hand, encrypts groups of characters of the plaintext at the same time. Stream ciphers are typically faster and less complex in hardware than block ciphers. They are also more suitable and, in some cases, mandatory (e.g., in wireless communications) when block size cannot be determined before transmission. Because they operate on individual bits, padding of bits is not required, and error propagation is less likely.

Stream ciphers are of two types: (i) synchronous, and (ii) self-synchronous (or asynchronous). The only difference is that in the synchronous stream cipher, the keystream generation is independent of the plaintext and the ciphertext. In contrast, the keystream generation depends on the fixed number of previous ciphertext digits in the asynchronous stream cipher. The most common self-synchronizing stream ciphers are based on block ciphers and currently use 1-bit Cipher Feedback (CFB) mode. Most of the synchronous stream ciphers proposed in the literature to date are XOR-based (additive) stream ciphers. The graphical representation of the additive stream cipher is given in Figure 5.1. In the figure, both encryption and decryption use the same

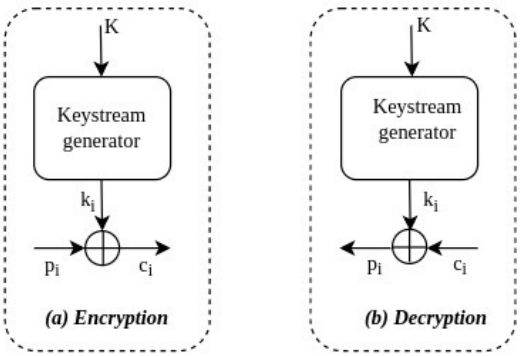


Figure 4.1: Representation of additive cipher.

keystream generation algorithm using the secret key K. In encryption algorithm, each plaintext digit p_i is encrypted with a keystream digit k_i and produces a ciphertext digit c_i. In decryption algorithm, on the other hand, each ciphertext digit c_i is decrypted

with the same keystream digit k_i as used in encryption and recovers the original plaintext digit p_i. The following is an example of additive stream cipher.

Example 4.1.1. *Encryption:*

$$\begin{aligned} PlainText &:= 1001010111001010 \\ &\bigoplus \\ KeyStream &:= 1010000110101011 \\ &\text{– – – – – – – – – – – – – – – –} \\ CipherText &:= 0011010001100001 \end{aligned}$$

Decryption:

$$\begin{aligned} CipherText &:= 0011010001100001 \\ &\bigoplus \\ KeyStream &:= 1010000110101011 \\ &\text{– – – – – – – – – – – – – – – –} \\ PlainText &:= 1001010111001010 \end{aligned}$$

where $\bigoplus$ *denotes a bit-wise addition modulo* 2 *operation.*

4.2 Brief overview of the proposed stream ciphers

This chapter proposes three stream ciphers based on quasigroups. Each cipher is a synchronous stream cipher that uses a quasigroup operation instead of the XOR operation of the conventional stream ciphers for encryption/decryption of the message. So, the new ciphers resolve the major issue of reused key attack that exists in the XOR-based stream ciphers. Also, the proposed ciphers use a new $\{e, d\}$`-transformation`, named $\{ne^{l/r}, nd^{l/r}\}$`-transformation` instead of $\{e, d\}$`-transformation` of the existing cipher based on quasigroup [46]. This new transformation allows the proposed ciphers to be resistant to known-plaintext, chosen-plaintext, and chosen-ciphertext attacks [45, 80]. The details of the new $\{e, d\}$`-transformation` are given in section 2.2.15 of Chapter 2.

The proposed ciphers use a keystream of size as long as the plaintext/ciphertext, is generated by a pseudo-random number generator. The ciphers consist of (i) an encryption algorithm, (ii) a decryption algorithm, and (ii) a keystream generation algorithm.

For encryption, the ciphers use a quasigroup $Q = (\mathbb{Z}_n, *)$ of order 16 or 256 (i.e. $n = 16$ or 256) along with either an ne^l`-transformation` or an ne^r`-transformation`, depending on the encryption algorithm. And for decryption, the ciphers use either an nd^l`-transformation` along with a left inverse quasigroup $LIQ = (\mathbb{Z}_n, \backslash)$ or an nd^r`-transformation` along with a right inverse quasigroup $RIQ = (\mathbb{Z}_n, /)$ of order 16 or 256 (i.e. $n = 16$ or 256), depending on the decryption algorithm. The symbols $*, \backslash$ and $/$ are the binary operations, here called a quasigroup operation, a left inverse quasigroup operation, and a right inverse quasigroup operation, respectively. The details of $Q = (\mathbb{Z}_n, *)$, $LIQ = (\mathbb{Z}_n, \backslash)$ and $RIQ = (\mathbb{Z}_n, /)$ are given in section 2.2.5 of Chapter 2.

All the proposed stream ciphers are iterative in nature and can be seen as polyalphabetic substitution ciphers since the relationship between the plaintext and the corresponding ciphertext is one-to-many. Note that the polyalphabetic substitution cipher is more secure than the monoalphabetic substitution cipher [56]. This is because, in the monoalphabetic substitution cipher, each plaintext character is substituted with the same ciphertext character; while in the polyalphabetic substitution cipher, each plaintext character is substituted with different ciphertext characters. Each of these substitution operations is carried out using a substitution table or a quasigroup. The first stream cipher[1] and the second stream cipher [2] use AES-256 and QG-PRNG for generating their keystreams, respectively. These ciphers can encrypt/decrypt a maximum of 16 bytes (128 bits) of plaintext/ciphertext in each iteration. The third stream cipher [3] uses MQG-PRNG for generating its keystream; and it can encrypt/decrypt a maximum of 8 bytes (64 bits) of plaintext/ciphertext in each iteration. Note that QG-PRNG and MQG-PRNG are quasigroup based pseudo-random number generators, where for generating the pseudo-random sequences, the QG-PRNG uses a quasigroup of order 256 while MQG-PRNG uses 16 quasigroups of order 16. Each of these stream ciphers is discussed in the following sections.

[1] New symmetric key cipher based on a quasigroup and AES-256
[2] A novel stream cipher based on a quasigroup and QG-PRNG
[3] MQG-PRNG and non-associative quasigroup based stream cipher

4.3 New symmetric key cipher based on a quasigroup and AES-256

This section discusses the structure and the building elements of a stream cipher. It is a symmetric key cipher and uses a secret key (denoted by K) for generating the keystream. For encrypting the plaintext, the cipher uses a quasigroup Q of order 256. In contrast, for decrypting the ciphertext, it uses a right inverse quasigroup RIQ of the quasigroup Q. It keeps both the secret key K and the quasigroup Q/RIQ secret, due to which the proposed cipher is resistant to various attacks, including known-plaintext, chosen-plaintext, and chosen-ciphertext attacks. It uses AES-256 for generating the keystream, denoted by K'.

4.3.1 Selection of a quasigroup of order 256

A quasigroup operation that uses 1-byte plaintext characters and 1-byte random keystream characters to produce 1-byte ciphertext characters at a time is the primary idea of employing a quasigroup of order 256. In the proposed algorithm, any order quasigroup in place of quasigroup of order 256 can be employed. Higher-order quasigroups are preferable because the number of quasigroups grows exponentially as the order increases. Since all ASCII values can be represented in 8 bits, and each character has an integer value ranging from 0 to 255, we chose a 256-order quasigroup in our proposed algorithm. It can be verified that the number of quasigroups of order 256 is very large (see section 2.2 of Chapter 2. So, it is practically impossible to guess correctly the employed quasigroup.

4.3.2 Keystream generation

The encryption/decryption algorithm of the stream cipher uses a keystream K' with a length equal to the plaintext/ciphertext. To generate such a long keystream the cipher uses an AES-256 encryption system. Using a secret key K, an initialization vector IV, and a counter '`Counter`', the encryption algorithm of AES-256 generates keystream K' of the required length. Also, the AES-256 encryption system assures that the generated keystream K' is random. The algorithm of the keystream generation based on the Cipher Block Chaining (CBC) mode of operation is given in Algorithm 3. In this algorithm, N denotes the number of iterations that the keystream generation algorithm

is to be performed, and the symbol $\oplus$ denotes an addition modulo 2 operation. The

Algorithm 3: Generation of keystream $K^{'}$

Input: 1. A `Counter` with 128 bits random value.
2. An IV with 128 bits initial value.
3. A secret key K of 256 bits.

Output: An 128-bit of keystream K'.

[1] **for** $i = 0$ *to* $N - 1$ **do**
[2] $\quad IV = IV \oplus Counter$;
[3] $\quad K'_{[i\times16+1,i\times16+2,\ldots,i\times16+16]}$ =AES-256_$ENC(IV, K)$;
[4] $\quad IV = K'_{[i\times16+1,i\times16+2,\ldots,i\times16+16]}$;
[5] $\quad Counter = Counter + 1$;

algorithm generates 16 bytes of keystream in each iteration and repeated until the keystream size is the same as that of the plaintext. The generated keystream can be represented as

$$K'_{[1,2,\ldots,n]} = k_1, k_2, \ldots, k_n$$

where each k_i is a 1-byte (character) value and will be used to encrypt the 1-byte (character) value of the plaintext. Note that the generated keystream K' is different from the secret key K used in AES-256. The keystream generation algorithm can use either the encryption or decryption algorithm of AES with a secret key of 128/192/256 bits. The proposed stream cipher uses an AES encryption system with a 256-bit secret key. The block diagram of the keystream generation is given in Figure 4.2.

4.3.3 Encryption algorithm

The encryption algorithm uses an ne^r`-transformation` along with a quasigroup $Q = (\mathbb{Z}_{256}, *)$ of order 256 for encrypting the plaintext $P = p_1\, p_2\, p_3 \ldots p_n$ using the generated keystream $K^{'} = k_1\, k_2\, k_3 \ldots k_n$, and produces the ciphertext $C = c_1\, c_2\, c_3 \ldots c_n$, whose size is equal to the plaintext. The ne^r`-transformation` is defined (see section 2.2.15 of Chapter 2) as

$$c_i = p_i * k_i$$

where p_i, k_i, and c_i are characters of 1-byte each, $1 \leq i \leq n$, and '$*$' is the quasigroup operation corresponding the chosen quasigroup $Q = (\mathbb{Z}_{256}, *)$. Note that the encryption

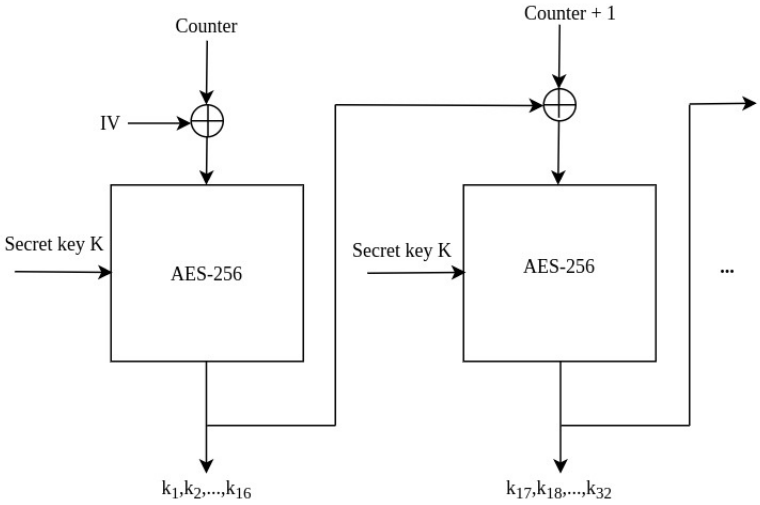


Figure 4.2: Generation of keystream using AES-256.

algorithm works on characters of 1-byte each. It encrypts each plaintext character p_i using a keystream character k_i and produces a ciphertext character c_i. If the keystream is not precomputed and is generated sequentially, the encryption algorithm can encrypt 16 bytes of plaintext in one iteration. The algorithm can encrypt the entire plaintext in a single iteration if the keystream is precomputed. Graphical representation of the encryption algorithm is shown in Figure 4.3. In this figure, an ne^r is nothing but the ne^r`-transformation`. Example 4.3.4 illustrates the functioning of this encryption algorithm.

Example 4.3.4. *Consider the quasigroup* $Q = (\mathbb{Z}_6, *)$ *with* $\mathbb{Z}_6 = \{0, 1, 2, 3, 4, 5\}$, *and its operation table is given in Table 4.1. Let* 16 *bytes of plaintext* P *be*

$$p_1, p_2, \ldots, p_{16} = 4145230103452012,$$

and 16 *bytes of the keystream* K' *be*

$$k_1, k_2, \ldots, k_{16} = 3410135243235301.$$

Then applying the foregoing encryption algorithm using the quasigroup Q *given in Table 4.1, we have* 16 *bytes of the ciphertext* C *as*

$$c_1, c_2, \ldots, c_{16} = 2552251225304302.$$

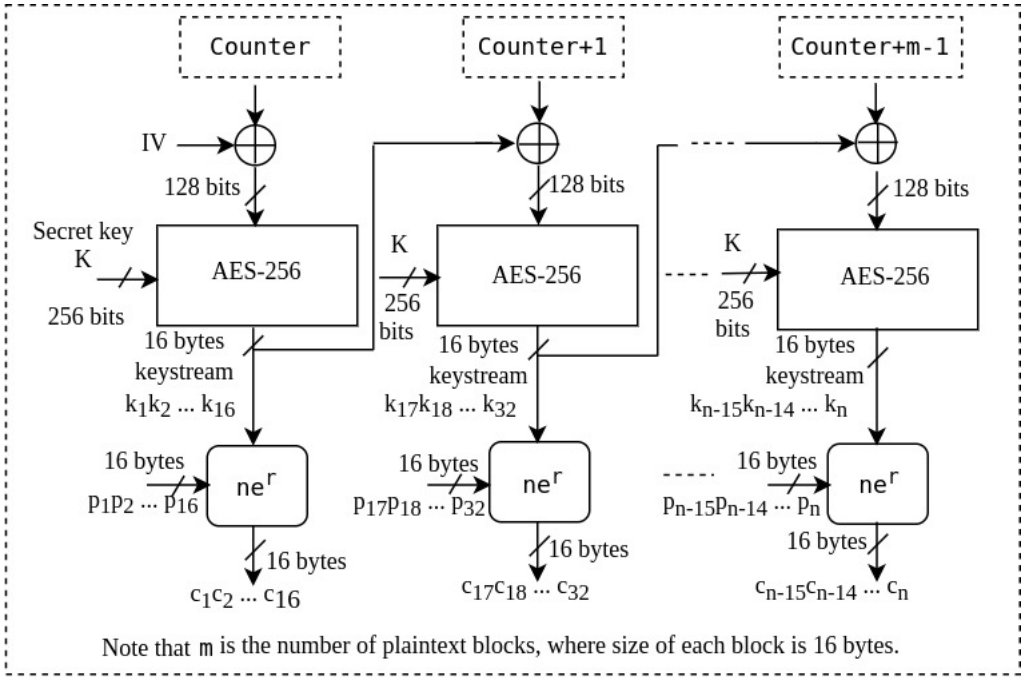


Figure 4.3: Encryption algorithm.

4.3.5 Decryption algorithm

The decryption algorithm is the reverse of the encryption algorithm. It uses an nd^r-`transformation` along with the right inverse quasigroup $RIQ = (\mathbb{Z}_{256}, /)$ of the quasigroup Q used in the encryption algorithm. It recovers the plaintext $P = p_1\, p_2\, p_3 \ldots p_n$ from the ciphertext $C = c_1\, c_2\, c_3 \ldots c_n$ using the same keystream $K^{'} = k_1\, k_2\, k_3 \ldots k_n$ as used in the encryption. The nd^r-`transformation` is defined (see section 2.2.15 of Chapter 2) as

$$p_i = c_i / k_i$$

where '/' is the right inverse quasigroup operation corresponding to the quasigroup RIQ, p_i, k_i, and c_i are characters of 1-byte each, $1 \leq i \leq n$. Note that the algorithm works on characters of 1-byte each, and it decrypts each ciphertext character c_i using a keystream character k_i and produces a plaintext character p_i. If the keystream is not precomputed and is generated sequentially, the decryption algorithm can decrypt 16 bytes of ciphertext in one iteration. The algorithm can decrypt the entire ciphertext in a single iteration if the keystream is precomputed. Graphical representation of the

Table 4.1: Quasigroup of order 6.

$*$	**0**	**1**	**2**	**3**	**4**	**5**
0	5	4	0	3	2	1
1	0	1	2	4	5	3
2	3	2	5	1	0	4
3	1	0	4	5	3	2
4	4	5	3	2	1	0
5	2	3	1	0	4	5

decryption algorithm is shown in Figure 4.4. In this figure, an nd^r is nothing but the nd^r-`transformation`. Example 4.3.6 illustrates the functioning of this decryption algorithm.

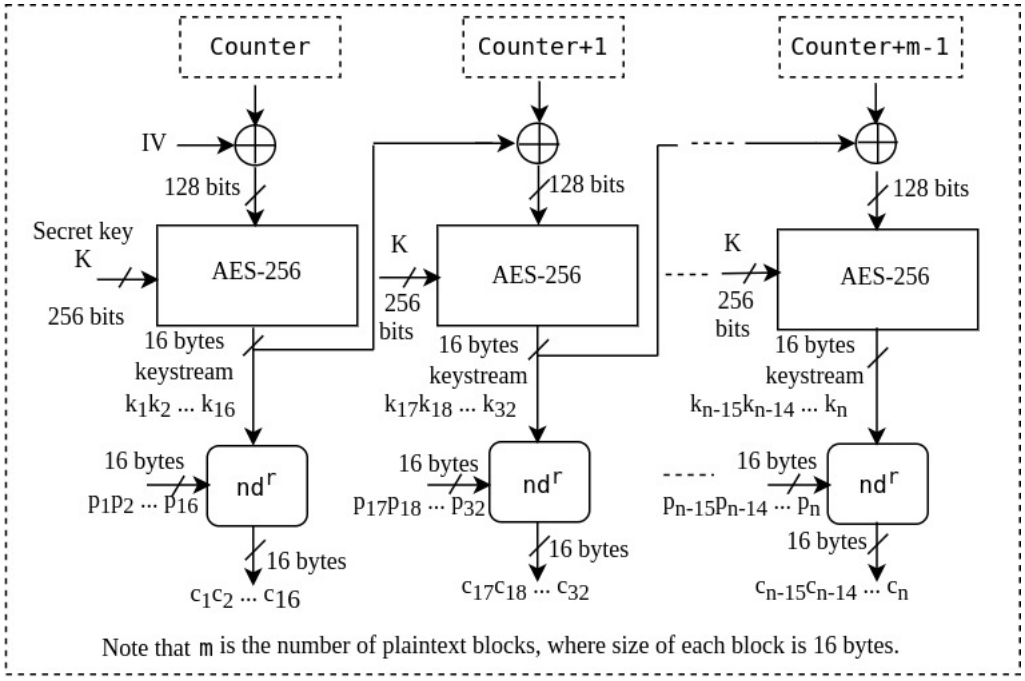


Figure 4.4: Decryption algorithm.

Example 4.3.6. *Consider the quasigroup* $RIQ = (\mathbb{Z}_6, /)$ *with* $\mathbb{Z}_6 = \{0, 1, 2, 3, 4, 5\}$, *and its operation table is given in Table 4.2. Note that RIQ is the right inverse quasigroup of the quasigroup Q, whose operation table is given in Table 4.1 (see section 2.2.5 of Chapter 2, for more details about quasigroup and its right inverse quasigroup). We*

consider the ciphertext C and the keystream K' form the Example 4.3.4. That is,

$$C = c_1, c_2, \ldots, c_{16} = 2552251225304302$$

and

$$K' = k_1, k_2, \ldots, k_{16} = 3410135243235301.$$

Then by applying the foregoing decryption algorithm using the quasigroup RIQ given in Table 4.2, we recovered the original plaintext P as

$$p_1, p_2, p_3, \ldots, p_{16} = 4145230103452012.$$

Table 4.2: Right inverse quasigroup of order 6.

/	**0**	**1**	**2**	**3**	**4**	**5**
0	1	3	0	5	2	4
1	3	1	5	2	4	0
2	5	2	1	4	0	3
3	2	5	4	0	3	1
4	4	0	3	1	5	2
5	0	4	2	3	1	5

4.3.7 Performance analysis

In this section, we analyzed the performance of the proposed stream cipher in terms of time (speed) and space complexities. Also, its performance is compared with some existing quasigroup based stream ciphers presented in the literature [12, 28, 43, 59, 60, 81].

- *Space complexity:* The space complexity of the proposed stream cipher is same as that of the existing XOR-based stream ciphers, except for the space required for the quasigroup used by the proposed cipher. The new cipher needs one quasigroup of order 256 along with an S-box of 256 bytes since it employed AES-256 for generating the keystream; that is, the cipher needs 65792 bytes of extra space for encryption/decryption. The space required to store S-boxes or quasigroups of the proposed cipher is also compared with that of the other existing quasigroup based stream ciphers [12, 28, 43, 59, 60, 81]. The results of this analysis are shown

in Table 4.3. It can be observed that the proposed cipher is as expensive as the existing quasigroup based stream ciphers, except for Edon-80 introduced in [28].

Table 4.3: Comparison of space complexity of the proposed cipher with existing ciphers.

Stream ciphers	**Space in KB**	
	Encryption	Decryption
Proposal cipher	64.25	64.25
Existing ciphers		
Chakrabarti et al. [12]	64	16384
Edon-80 [28]	0.02	0.02
Lakshmi et al. [43]	64	64.007
Petrescu [59, 60]	64	16384
Zhang and Xu [81]	256	320

- *Time complexity:* The proposed stream cipher is implemented in C++ language on a system with the following configuration: Intel(R) Core(TM) i5-2400 CPU @3.40 GHz processor with 8 GB RAM and 64-bit Linux operating system. The source code of the proposed cipher is run 1000 times for different inputs and we calculated the average execution time in seconds. The cipher used the C++ standard <`chrono`> library to measure the execution time [38]. The performance of the proposed cipher is compared with those of the existing quasigroup based stream ciphers presented in literature [12, 28, 43, 59, 60, 81]. Table 4.4 shows the time complexities of different ciphers, which are listed in the first column of the table. For each of these ciphers, the second, third, and fourth columns of the table list the execution times in seconds for different inputs. As a result, It can be observed that the proposed cipher is faster than all the existing quasigroup based stream ciphers, except for one introduced in [43].

4.3.8 Security analysis

The primary strength of the proposed stream cipher is that it uses AES-256 in Cipher Block Chaining (CBC) mode encryption for generating the keystream K'. Also, the key elements for the AES-256 consist of the secret key K, the initialization vector IV,

Table 4.4: Comparison of time complexity of the proposed cipher with existing ciphers.

Stream ciphers	Execution time in seconds		
	1.22 MB	2.11 MB	6.01 MB
Proposal cipher	0.21	0.39	1.12
Existing ciphers			
Chakrabarti et al. [12]	0.30	0.51	1.32
Edon-80 [28]	3.51	6.89	13.78
Lakshmi et al. [43]	0.04	0.09	0.28
Petrescu [59, 60]	0.23	0.40	1.15
Zhang and Xu [81]	0.62	1.24	3.73

and the *Counter*. Since the size of secret key K is 256 bits, it is resistant to brute force attack as there are 2^{256} possible keys. AES-256 has already been proven to be secure against variety of attacks, including brute force attack, related-key attack, and linear and differential attacks. Hence the keystream K' generation is secure from various attacks.

Note that the proposed stream cipher keeps both the quasigroup Q and the generated keystream K' secret so as to make the cipher resistant to known-plaintext, chosen-plaintext, and chosen-ciphertext attacks. In known-plaintext attack, the cryptanalyst knows a plaintext string $p_1\, p_2\, \ldots p_n$ and the corresponding ciphertext string $c_1\, c_2\, \ldots c_n$. In chosen-plaintext attack, the cryptanalyst chooses a plaintext string $p_1\, p_2\, \ldots p_n$, and obtains the corresponding ciphertext string $c_1\, c_2\, \ldots c_n$ by temporarily accessing the encryption system. And in chosen-ciphertext attack, the cryptanalyst chooses a ciphertext string $c_1\, c_2\, \ldots c_n$, and obtains the corresponding plaintext string $p_1\, p_2\, \ldots p_n$ by temporarily accessing the decryption system. For encrypting the plaintext, if the quasigroup Q used to encrypt is not secret, it is easy for the cryptanalyst to obtain the keystream K'. So this stream cipher can not resist the known-plaintext, chosen-plaintext, and chosen-ciphertext attacks. Consequently, we need to keep the quasigroup Q secret, details are given in the next section.

4.3.8.1 Known plaintext attack

The proposed cipher uses an ne^r/nd^r`-transformation` along with a quasigroup Q/RIQ of order 256 in the encryption/decryption system. Note that the maximum number of

quasigroups of order 256 is bounded above by

$$0.753 \times 10^{102,805}$$

. Now, let us assume that a cryptanalyst knows a plaintext $P = p_1\, p_2\, \ldots p_n$ and the corresponding ciphertext $C = c_1\, c_2\, \ldots c_n$. Also, assume that everyone knows the employed quasigroup Q. Then for determining the keystream $K' = k_1, k_2 \ldots, k_n$, which is a confidential element of the encryption-decryption system, the adversary must solve the following system of equations corresponding to the ne^r`-transformation` used in the encryption system:

$$\left.\begin{aligned} c_1 &= p_1 * k_1 \\ c_2 &= p_2 * k_2 \\ &\cdots \\ c_n &= p_n * k_n \end{aligned}\right\} \tag{4.1}$$

where $k_1, k_2, \ldots, k_n$ are unknown.

This system of equations has a unique solution due to the uniqueness property of the quasigroup (see section 2.2 of Chapter 2). So, the cipher can not resist the known-plaintext attack. Whereas, if we keep the quasigroup Q secret, then the system of equations given in Equation (4.1) has as many solutions as there are the number of quasigroups of order 256. So, determining the employed quasigroup Q and hence the keystream $k_1, k_2, \ldots, k_n$ make it practically impossible. Therefore, the proposed stream cipher is resistant to the known-plaintext attack.

Using a similar argument, it can be shown that the cipher is resistant to the chosen-ciphertext and chosen-plaintext attacks as well.

4.3.8.2 Statistical test for randomness using NIST-STS test suite

The randomness of the ciphertexts obtained from the proposed stream cipher is tested using the NIST-STS test suite [65]. Each NIST-STS test yields a P-value between 0 and 1 (both included) and indicates success or failure. A P-value represents the probability that a perfect random number generator would produce a less random sequence than the one being tested[65]. For these tests, we have chosen a threshold value (α) to be 0.01, also called the significance level, and other parameters as shown in Table 4.5. Typically, the value of α is chosen in the range $[0.001, 0.01]$. For the randomness of a sequence, we

compare the P-value of a sequence to a threshold value (α). If P-value $\geq \alpha$, then the sequence is considered to be random, otherwise non-random. We have used NIST Spec. Publ. 800-22 rev. 1a package that consists of 15 types of statistical tests [65]. Each of these 15 tests has different input parameters and a different number of P-values [13]. Each P-value corresponds to a single statistical test on a binary sequence. Some of these tests perform a series of t sub-tests, $t \in \{2, 8, 18, 148\}$. Following are these 15 tests and some of their sub-tests:

- Frequency Test: It evaluates the frequency of ones and zeros in the entire sequence.
- Block Frequency (BF) Test: It evaluates the frequency of ones and zeros in m-bit blocks.
- Cumulative Sum (CS) Test: It performs two types of cumulative sum tests (or performs 2 sub-tests). Both evaluate whether the maximal cumulative sum of partial sequences is outside the range for the expected behavior of a random sequence.
- Runs Test: This test evaluates the longest sequence of contiguous ones in the entire sequence and compares the oscillation between ones and zeros to a standard frequency.
- Longest Run (LR) Test: It compares the longest contiguous run of ones in m-bit blocks to the expected frequency of the same.
- Rank Test: It evaluates the rank of disjoint sub-matrices within the entire sequence.
- Discrete Fourier Transform (DFT) Test: It is implemented as a Fast Fourier Transform. It detects repeating or periodic features that are near to each other.
- Non-overlapping Template (NOT) Test: The purpose of this test is to detect generators that produce too many occurrences of a given non-periodic (aperiodic) pattern. It performs 148 sub-tests and uses an m-bit window to search for a specific m-bit pattern. If the pattern is not found, the window slides a one-bit position. If the pattern is found, the window is reset to the bit after the found pattern, and the search resumes.

- Overlapping Template (OT) Test: It evaluates the number of occurrences of pre-specified target strings.
- Universal Statistical (US) Test: It is also called a Maurer's test. It detects whether or not the sequence can be significantly compressed without loss of information. A significantly compressible sequence is considered to be non-random.
- Approximate Entropy (AE) Test: It evaluates the frequency of all possible overlapping m-bit patterns across the entire sequence.
- Random Excursion (RE) Test: The purpose of this test is to determine if the the number of visits to a state within a random walk exceeds what one would expect for a random sequence. This test performs actually a series of 8 sub-tests.
- Random Excursion Variants (REV) Test: The purpose of this test is to detect deviations from the expected number of visits to various states in the random walk. This test performs actually a series of 18 sub-tests.
- Serial Test: It performs two types of serial tests (or performs 2 sub-tests). Both compare the frequency of all the m-bit overlapping patterns in the full sequence separately.
- Linear Complexity (LC) Test: It uses linear complexity to test for randomness.

Table 4.5: Parameters for the NIST-STS test.

Tests	Block length(m)
Block frequency test	128
Non-overlapping template test	9
Overlapping template test	9
Approximate entropy test	10
Serial test	16
Linear complexity test	500

We run each of these tests for 1000 obtained ciphertext sequences. The size of each sequence is 10^6 bits. Table 4.6 shows the results of the average P-value of the NIST-STS test suite. As a result, it can be observed that the P-value of each of these tests crosses

the significance level ($\alpha = 0.01$), and conclude that the obtained ciphertext sequences are random.

Table 4.6: Results of the NIST-STS test.

Si. No.	Tests	P-value
1	Frequency	0.507678
2	Block frequency	0.513950
3	Cumulative sum	0.675720
4	Runs	0.542138
5	Longest run	0.552834
6	Rank	0.443481
7	Discrete fourier transform	0.500707
8	Non-overlapping template	0.479753
9	Overlapping template	0.491251
10	Universal statistical	0.511753
11	Approximate entropy	0.471052
12	Random excursion	0.501743
13	Random excursion variants	0.494101
14	Serial	0.553913
15	Linear complexity	0.576939

4.4 A novel stream cipher based on a quasigroup and QG-PRNG

This section discusses (i) a new cipher algorithm based on a quasigroup of order 256, and (ii) a pseudo-random number generator based on a quasigroup of order 256, named QG-PRNG. This stream cipher is an extension of work initially discussed in section 4.3. It uses QG-PRNG for generating the keystream in place of AES-256 used in the previous stream cipher. So, the new stream cipher is more efficient than the previous one. It can be noted that to generate a keystream K' of 160 KB, AES-256 takes around 27499 μs, while QG-PRNG takes only 579 μs. Because of this, the overall cipher is faster than the previous cipher, and it takes around 6367 μs to encrypt data of $4MB$, while the previous stream cipher takes around 800301 μs. Also, the decryption algorithm of this stream cipher uses the left inverse quasigroup in place of the right inverse quasigroup

such as used in the previous cipher. The randomness of the obtained keystream K' generated by QG-PRNG is analyzed using the NIST-STS test suite. It is observed that the outputs of QG-PRNG are highly random. The structure of the proposed stream cipher and its building elements are discussed in the subsequent sections.

4.4.1 Keystream generation

The encryption/decryption algorithm uses a keystream K' of size as long as the plaintext. To generate such a long keystream K', we used a pseudo-random number generator based on a quasigroup of order 256, named QG-PRNG. It uses an initialization vector (IV), also called a seed-value of 128 bits (16 bytes), and a quasigroup of order 256. The algorithm of QG-PRNG produces keystream K' of the required length. Since the encryption algorithm requires the keystream K' to be random, QG-PRNG ensures this. Workflow of the keystream generation is given in Figure 4.5.

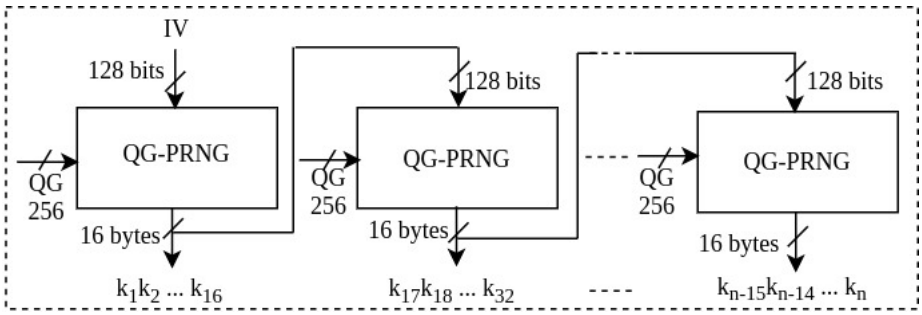


Figure 4.5: Workflow of keystream generation using QG-PRNG.

Note that the algorithm of QG-PRNG is described as well as implemented using the Output Feedback (OFB) mode of operation. Each iteration of QG-PRNG generates 128 bits (16 bytes) of keystream K' and it is repeated until the size of keystream $K^{'}$ is the same as that of the plaintext. This generates the keystream as

$$K^{'} = k_1\, k_2\, k_3 \ldots k_n\,,$$

where each k_i is a 1-byte character and will be used to encrypt a 1-byte character of the plaintext. It can also be implemented using other modes of operation such as Cipher Feedback (CFB) mode, Cipher Block Chaining (CBC), and Counter (CTR) mode. Each mode of operation has its advantages and disadvantages [67].

Let $IV = (s_0, s_1, \ldots, s_{15})$ be a seed of 128 bits (16 bytes), where each s_i is a byte value for $0 \leq i \leq 15$. QG-PRNG employs two levels of e-`transformation` on IV successively to arrive at the required keystream. Details of the e-`transformation` are given in section 2.2.12 of Chapter 2. Each of these two levels of e-`transformation` is carried out in the specified range and direction. We use `e-left`$(s_j \rightarrow s_{j+3}, \ell)$ transformation, commonly known as e-`transformation` that works in the increasing sequence of bytes (8-bit values) or from the left to right direction; and `e-right`$(s_{j+3} \rightarrow s_j, \ell)$ transformation, it works in the reverse direction of `e-left`$(s_j \rightarrow s_{j+3}, \ell)$. The description of the `e-left`$(s_j \rightarrow s_{j+3}, \ell)$ and `e-right`$(s_{j+3} \rightarrow s_j, \ell)$ are as follows:

$$\texttt{e-left}(s_j \rightarrow s_{j+3}, \ell) : \begin{cases} s_j = \ell * s_j \\ s_{j+1} = s_j * s_{j+1} \\ s_{j+2} = s_{j+1} * s_{j+2} \\ s_{j+3} = s_{j+2} * s_{j+3} \end{cases}$$

and

$$\texttt{e-right}(s_{j+3} \rightarrow s_j, \ell) : \begin{cases} s_{j+3} = \ell * s_{j+3} \\ s_{j+2} = s_{j+3} * s_{j+2} \\ s_{j+1} = s_{j+2} * s_{j+1} \\ s_j = s_{j+1} * s_j \end{cases}$$

where $*$ is one of the quasigroup operations corresponding to the selected quasigroup $Q = (\mathbb{Z}_{256}, *)$ and ℓ denotes a leader that belongs to $\mathbb{Z}_{256}$. The successive application of these two levels of e-`transformation` is as follows:

- First level of e-`transformation`:- The first level of e-`transformation` on the original IV for the first iteration (or on the intermediate IV for the remaining iterations) is carried out using the following four successive transformations with initial leader $a \in \mathbb{Z}_{256}$: (i) `e-left`$(s_0 \rightarrow s_3, \ell = a)$, (ii) `e-right`$(s_7 \rightarrow s_4, \ell = s_3)$, (iii) `e-left`$(s_8 \rightarrow s_{11}, \ell = s_4)$, and (iv) `e-right`$(s_{15} \rightarrow s_{12}, \ell = s_{11})$. This is depicted in Figure 4.6.

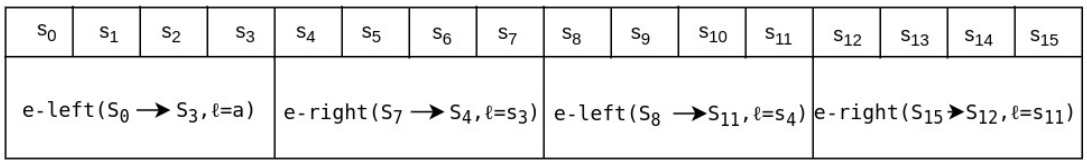


Figure 4.6: First level of e-`transformation`.

- Second level of e-`transformation`:- The second level of e-`transformation` works in reverse order of the first level of e-`transformation`, as shown in Figure 4.7. And the e-`transformation` is applied on the output of the first level of e-`transformation`. The successive expressions governing this transformation are: (i) `e-left`$(s_{12} \rightarrow s_{15}, \ell = b)$, (ii) `e-right`$(s_{11} \rightarrow s_8, \ell = s_{15})$, (iii) `e-left`$(s_4 \rightarrow s_7, \ell = s_8)$, and (iv) `e-right`$(s_3 \rightarrow s_0, \ell = s_7)$, where the initial leader $b \in \mathbb{Z}_{256}$.

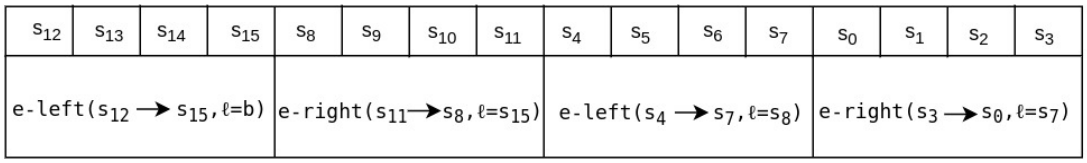


Figure 4.7: Second level of e-`transformation`.

These successive transformations ensure that if a single bit of the original seed (IV) is changed, then each bit of the intermediate IV (or in the corresponding keystream) would be changed with high probability. The pseudocode of the generation of the keystream based on the Output Feedback (OFB) mode of operation is given in Algorithm 4. In this algorithm, N denotes the number of iterations that the QG-PRNG is to be performed, and IV_I denotes the I^{th} iteration initialization vector (or intermediate IV). For the 0^{th} iteration, IV_0 is IV itself. In each iteration of QG-PRNG, line numbers from 3 to 6 perform the first level of e-`transformation` on the original IV_0 (or on the intermediate $IV_I, I \geq 1$). This is followed by the second level of e-`transformation` specified in the lines from 7 to 10. That is, both the levels of e-`transformation` update IV_I in each iteration of QG-PRNG, and the results are stored in the same IV_I. Also, the recent IV_I is fed into the next iteration of QG-PRNG.

4.4.1.1 Statistical test of QG-PRNG using NIST-STS test suite

The randomness of the obtained keystream K', which is generated by the QG-PRNG, is tested using the NIST-STS test suite [65]. The NIST-STS test suite consists of various statistical tests. The details of each of such tests are discussed already in section 4.3.8.2. We have run each test of the NIST-STS test suite with the significance level $\alpha = 0.01$. We ran each test of NIST-STS for 1000 obtained pseudo-random sequences (keystreams), where the size of each keystream is 10^6 bits. The obtained

Algorithm 4: Pseudocode of the QG-PRNG based on OFB mode

Input: 1. Original $IV = (s_0, s_1, \ldots, s_{15})$ of 128 bits (16 bytes).
2. A quasigroup $Q = (\mathbb{Z}_{256}, *)$ of order 256.
3. The initial leaders $a, b \in \mathbb{Z}_{256}$

Output: Each iteration produces 16 bytes (128 bits) of keystream.

[1] **for** $I = 0$ *to* $N - 1$ **do**
[2] $IV_I = IV$;
[3] $\ell = a$, `e-left`$(s_0 \rightarrow s_3, \ell)$; ▷ 1^{st} level of e-transformation start
[4] $\ell = s_3$, `e-right`$(s_7 \rightarrow s_4, \ell)$;
[5] $\ell = s_4$, `e-left`$(s_8 \rightarrow s_{11}, \ell)$;
[6] $\ell = s_{11}$, `e-right`$(s_{15} \rightarrow s_{12}, \ell)$; ▷ 1^{st} level of e-transformation end
[7] $\ell = b$, `e-left`$(s_{12} \rightarrow s_{15}, \ell)$; ▷ 2^{nd} level of e-transformation start
[8] $\ell = s_{15}$, `e-right`$(s_{11} \rightarrow s_8, \ell)$;
[9] $\ell = s_8$, `e-left`$(s_4 \rightarrow s_7, \ell)$;
[10] $\ell = s_7$, `e-right`$(s_3 \rightarrow s_0, \ell)$; ▷ 2^{nd} level of e-transformation end.
[11] $IV = IV_I$

results of the QG-PRNG are compared with that of the AES-256. Table 4.7 shows the results of both the QG-PRNG and AES-256. From the results, as shown in Table 4.7, It can be observed that the randomness of the QG-PRNG is approximately the same as that of the AES-256.

4.4.2 Encryption algorithm

The encryption algorithm of this cipher is almost the same as that of the encryption algorithm of the previous cipher, discussed in section 4.3.3. The only difference is that it uses an ne^l`-transformation` instead of ne^r`-transformation`. The description of ne^l`-transformation` is specified in section 2.2.15 of Chapter 2. It uses the same quasigroup as used by the keystream generation algorithm QG-PRNG . For a plaintext $P = p_1\, p_2\, p_3 \ldots p_n$ and the keystream $K^{'} = k_1\, k_2\, k_3 \ldots k_n$, it produces the ciphertext $C = c_1\, c_2\, c_3 \ldots c_n$, whose size is equal to the plaintext, as follows:

$$c_i = k_i * p_i$$

where p_i is a plaintext character, k_i is a keystream character, and c_i is a ciphertext character; the size of each character is 8 bits (1-byte), $1 \leq i \leq n$, and '$*$' is the

Table 4.7: For 1000 random sequences, results of the NIST-STS test suite for QG-PRNG as compared to AES-256.

Tests	QG-PRNG			AES-256		
	Number of success	Number of failures	Proportion of success out of 1000	Number of success	Number of failures	Proportion of success out of 1000
Frequency	995	5	0.995	991	9	0.991
BF	990	10	0.990	992	8	0.992
CS	992	8	0.992	991	9	0.991
Runs	990	10	0.990	989	11	0.989
LR	989	11	0.989	990	10	0.990
Rank	994	6	0.994	992	8	0.992
DFT	987	13	0.987	986	14	0.986
NOT	981	19	0.981	980	20	0.980
OT	994	6	0.994	995	5	0.995
US	988	12	0.988	988	12	0.988
AE	991	9	0.991	990	10	0.990
RE	988	12	0.988	985	15	0.985
REV	990	10	0.990	990	10	0.990
Serial	993	7	0.993	994	6	0.994
LC	996	4	0.996	983	17	0.983

quasigroup operation corresponding the chosen quasigroup $Q = (\mathbb{Z}_{256}, *)$ of order 256. The workflow of the encryption algorithm is shown in Figure 4.8. In this figure, ne^l is nothing but an ne^l-`transformation`. Functionality of this encryption algorithm is illustrated by Example 4.4.3. Compare this with the one given in Example 4.3.4 of section 4.3.3. Note that for the same inputs (P, K', and Q), these two examples produce different results. This is because the quasigroup $Q = (\mathbb{Z}_{256}, *)$ is a non-commutative algebraic structure, i.e., as $p_i * k_i \neq k_i * p_i$.

Example 4.4.3. *Consider the same quasigroup $QG = (\mathbb{Z}_6, *)$ with $\mathbb{Z}_6 = \{0, 1, 2, 3, 4, 5\}$ used in Example 4.3.4, and its operation table is given in Table 4.1. We also consider the same plaintext P to be encrypted and the same keystream K' used in Example 4.3.4. That is,*

$$P = p_1, p_2, \ldots, p_{16} = 4145230103452012$$

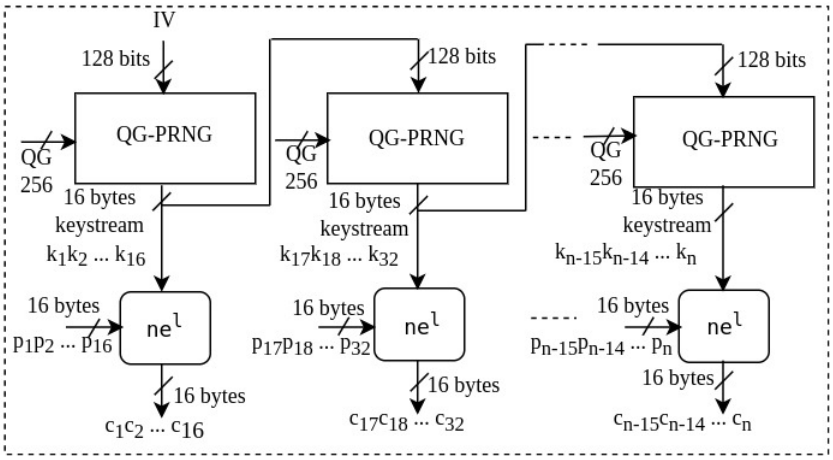


Figure 4.8: Workflow of encryption algorithm.

and

$$K' = k_1, k_2, \ldots, k_{16} = 3410135243235301.$$

Then applying the foregoing encryption algorithm using the quasigroup Q given in Table 4.1, we have the ciphertext as

$$C = c_1, c_2, \ldots, c_{16} = 3551252245021142.$$

4.4.4 Decryption algorithm

The decryption algorithm is the reverse of the encryption algorithm. It uses two quasigroups Q and LIQ of order 256, where Q is used for generating the keystream K' and LIQ is used for decryption since both encryption and decryption algorithms use the same keystream. The decryption algorithm of this cipher is almost the same as that of the decryption algorithm of the previous cipher, discussed in section 4.3.5. The only difference is that it uses an nd^l`-transformation` instead of nd^r`-transformation`. The description of nd^l`-transformation` is specified in section 2.2.15 of Chapter 2. It uses the left inverse quasigroup $LIQ = (\mathbb{Z}_{256}, \backslash)$ of the quasigroup Q used in the encryption algorithm. The main principle of the decryption algorithm is that for the given ciphertext $C = c_1\, c_2\, c_3 \ldots c_n$ and the same keystream $K^{'} = k_1\, k_2\, k_3 \ldots k_n$ used in the encryption algorithm, it recovers the original plaintext $P = p_1\, p_2\, p_3 \ldots p_n$ as follows:

$$p_i = k_i \backslash c_i$$

where '\' is the left inverse quasigroup operation corresponding to the quasigroup LIQ, and all p_i, k_i, and c_i are characters of 8 bits (1-byte), $1 \leq i \leq n$. Workflow of the decryption algorithm is given in Figure 4.9. In this figure, nd^l is nothing but an nd^l-`transformation`. Example 4.4.5 illustrates the functioning of this decryption

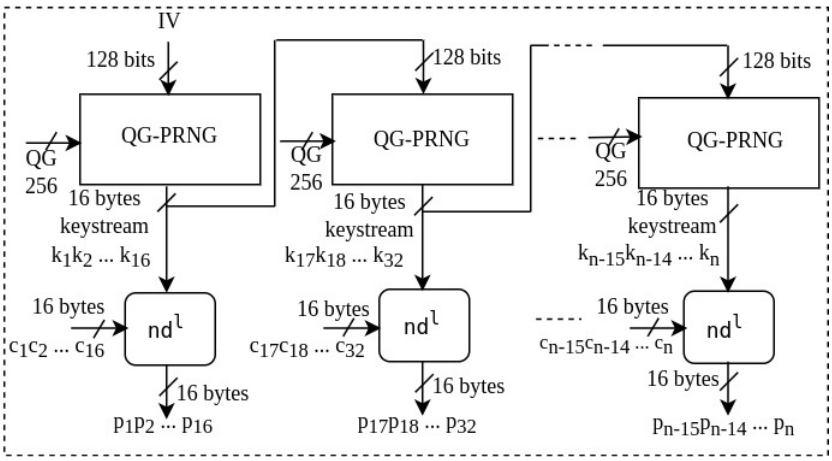


Figure 4.9: Workflow of decryption algorithm.

algorithm.

Example 4.4.5. *Consider the quasigroup* $LIQ = (\mathbb{Z}_6, \backslash)$ *with* $\mathbb{Z}_6 = \{0, 1, 2, 3, 4, 5\}$. *Its operation table is given in Table 4.8. Note that LIQ is the left inverse quasigroup of the quasigroup Q, whose operation table is given in Table 4.1 (see section 2.2.5 of Chapter 2, for more details about quasigroup and its left inverse quasigroup). We consider the ciphertext C and the keystream K′ from the Example 4.4.3. That is,*

$$C = c_1, c_2, \ldots, c_{16} = 3551252245021142$$

and

$$K' = k_1, k_2, \ldots, k_{16} = 3410135243235301.$$

Then by applying the foregoing decryption algorithm using the quasigroup LIQ given in Table 4.8, we recovered the original plaintext P as

$$p_1, p_2, p_3, \ldots, p_{16} = 4145230103452012.$$

4.4.6 Performance analysis

The proposed cipher is implemented in C++, its performance is analyzed in terms of space and time (speed) complexities by comparing it with the previous cipher discussed

Table 4.8: Right inverse quasigroup of order 6.

\	**0**	**1**	**2**	**3**	**4**	**5**
0	2	5	4	3	1	0
1	0	1	2	5	3	4
2	4	3	1	0	5	2
3	1	0	5	4	2	3
4	5	4	3	2	0	1
5	3	2	0	1	4	5

in section 4.3 and some existing quasigroup based stream ciphers presented in the literature [12, 28, 43, 59, 60, 81]. To analyze the performance of this cipher, we used the same procedures as we used to analyze the previous cipher, discussed in section 4.3.7. The cipher ran 1000 times for different inputs and we calculated the average execution time in seconds. Experimental results for both time and space complexities are given in Table 4.9. If we compare this result with the previous results shown in section 4.3.7, it can be observed that in terms of memory requirement, this cipher is a bit more expensive than the previous ciphers. But, it is faster than our first cipher discussed in section 4.3, including other existing ciphers introduced in literature [12, 28, 43, 59, 60, 81].

Table 4.9: Time and space complexities of the proposed cipher.

Stream cipher	**Execution time in seconds**			**Space in KB**	
	1.22 MB	2.11 MB	6.01 MB	Encryption	Decryption
Proposed cipher	0.002	0.003	0.007	64	128

4.4.7 Security analysis

For generating the keystream K', the proposed cipher uses QG-PRNG. The QG-PRNG is a pseudo-random number generator that uses an initialization value IV (also called a seed value) of 128 bits along with a quasigroup $Q = (\mathbb{Z}_{256}, *)$ of order 256. As discussed in section 2.2 of Chapter 2, the maximum number of quasigroups of order 256 is bounded above by

$$0.753 \times 10^{102,805}.$$

So, QG-PRNG can generate a maximum of $0.753 \times 10^{102,805}$ possible keystreams K' for each IV. Since, quasigroup Q is one of the core elements of both the schemes (QG-PRNG and encryption/decryption algorithms), so it is kept secret. If the quasigroup Q used is not secret, it is easy for the cryptanalyst to crack the cipher against a known-plaintext attack. This is already illustrated in the previous section 4.3.8.1. Typically, IV is known to everyone. However, our observation is that for any cryptographically secure pseudo-random generator, we need a secret random seed so that the corresponding output is unpredictable. If we keep IV secret along with the quasigroup Q, the keystream generator (QG-PRNG) can be seen to provide greater security. This is because the security of the QG-PRNG depends not only on the IV but also on the employed quasigroup Q, which can be changed rapidly. For determining the IV and the employed quasigroup Q, a cryptanalyst can apply the brute force methods as follows:

- *Exhaustive IV search only attack*: The QG-PRNG uses an IV of 128 bits. Therefore, the number of possible IVs is $2^{128} \approx 3.4 \times 10^{38}$. Let us assume cryptanalyst uses a supercomputer and tries 5.37×10^{17} IVs per second, then the cryptanalyst needs around 2.01×10^{13} years to determine the employed IV. This is because these days a supercomputer can execute 5.37×10^{17} FLOPS[1] [76].

- *Quasigroups only attack*: The QG-PRNG uses only one quasigroup of order 256, and a maximum number of quasigroup of order 256 is 0.753×10^{102805}. Here also, we employ the same argument as that given in the exhaustive IV search only attack. That is, if a cryptanalyst tries 5.37×10^{17} quasigroups per second, then the attacker needs around 0.044×10^{102781} years to guess the employed quasigroup.

- *Attack complexity*: The attack complexity of QG-PRNG can be defined in the following two ways: (i) If we keep the IV public, then the attack complexity of QG-PRNG against brute-force attack is equivalent to that of the quasigroups only attack, and (ii) if we keep the IV secret along with quasigroup Q, then the attack complexity of QG-PRNG against brute-force attack is equal to the number of computations required by a cryptanalyst to discover both the IV and the employed quasigroup. But this is equal to the product of the number of computations required by the exhaustive IV search only attack and the number

[1] Floating-point Operations Per Second

of computations required by the quasigroups only attack. So, the complexity of brute force attack (or key-space complexity) against QG-PRNG is equal to 2.560×10^{102843}. That is, if an attacker performs 5.37×10^{17} computations per second [76], then the cryptanalyst needs around 0.151×10^{102819} years to crack the QG-PRNG.

In the case of the encryption of the proposed stream cipher, the same quasigroup Q is used in both the encryption algorithm and the QG-PRNG. The quasigroup Q used in the QG-PRNG of the decryption is also the same as that of the encryption. However, in the case of decryption, the quasigroup used is the left inverse of that used in the encryption. This is because both encryption and decryption algorithms use the same keystream K'. As discussed above, in the quasigroups attack, it is practically impossible to determine the employed quasigroups. Hence, the proposed cipher is resistant to brute-force attacks.

4.4.7.1 Known plaintext attack

Since the encryption/decryption algorithm of this cipher is almost the same as that of the previous cipher, discussed in section 4.3. The only difference is that it uses ne^l/nd^l`-transformation` instead of ne^r/nd^r`-transformation` for encrypting/decrypting the messages. Therefore, the attack complexity of this cipher against known-plaintext attack would be the same as that of the previous cipher illustrated in section 4.3.8.1. The attack can be carried out by solving the following system of equations:

$$\left.\begin{aligned} c_1 &= k_1 * p_1 \\ c_2 &= k_2 * p_2 \\ &\cdots \\ c_n &= k_n * p_n \end{aligned}\right\} \tag{4.2}$$

where $k_1, k_2, \ldots, k_n$ are unknown. This system of equations has as many solutions as there are the number of quasigroups of order 256., which is practically infinite, and hence the cipher is resistant to known-plaintext attack.

Using a similar argument, it can be shown that the cipher is resistant to the chosen-ciphertext and chosen-plaintext attacks as well.

4.4.7.2 Reused key attack

Reused key attack is applied on a stream cipher that uses the keystream more than once for encrypting the messages. If we use the same keystream K' to encrypt two or more different messages, an attacker in the XOR-based stream ciphers, can eliminate the encryption by XORing the two messages; whereas in the proposed cipher such an elimination is not possible. To see this, consider

$$C_1 = K' * M_1$$
$$C_2 = K' * M_2$$

where C_1 and C_2 are the ciphertexts obtained using the proposed stream cipher, and $*$ is one of the quasigroup operations corresponding to the employed quasigroup of order 256. If we use $\oplus$ operation instead of $*$, then the cipher becomes XOR-based stream cipher and the cryptanalyst in the reused key attack computes

$$C_1 \oplus C_2 = (K' \oplus M_1) \oplus (K' \oplus M_2) = M_1 \oplus M_2.$$

where $\oplus$ is a bit-wise addition modulo 2 operation. See that the obtained byte sequence does not depend on the keystream K'. In other words, if anyone intercepts two messages encrypted with the same key, they can recover $M_1 \oplus M_2$ which is a form of running key cipher. Even if neither message is known, as long as both the messages are in a natural language, such a cipher can often be broken because of enough redundancy present in English and ASCII encoding. So, an attacker can easily recover the original messages from:

$$C_1 \oplus C_2 = M_1 \oplus M_2 \rightarrow M_1, M_2$$

But in our case:

$$C_1 * C_2 = (K' * M_1) * (K' * M_2) \neq M_1 * M_2$$

Hence, the proposed stream cipher is resistant to reused-key attack.

4.4.7.3 Statistical attack

The encryption system of the proposed cipher uses a quasigroup $Q = (\mathbb{Z}_{256}, *)$ with $\mathbb{Z}_{256} = \{0, 1, \ldots, 255\}$. All possible elements of $\mathbb{Z}_{256}$ occurs with equal probability in the quasigroup Q. Also, the algorithm of keystream generation (QG-PRNG) uses the

quasigroup Q for generating the keystream $K' = k_1, k_2, \ldots, k_n$, where each k_j is a byte value and belongs to $\mathbb{Z}_{256}$, $1 \leq j \leq n$. So, all possible elements of $\mathbb{Z}_{256}$ also occur with equal probability in the generated keystream sequence $k_1, k_2, \ldots, k_n$. That is, each element of $\mathbb{Z}_{256}$ occurs as often as any other in each position of $k_1, k_2, \ldots, k_n$. Since the probability $P(k_j) = \frac{1}{256}, 1 \leq j \leq n$, the distribution of the keystream sequence is uniform. We cannot say the plaintext $p_1, p_2, \ldots, p_n$ is a random sequence. It can be inferred that the corresponding ciphertext $c_1, c_2, \ldots, c_n$ is uniformly distributed. This is because, the ciphertext obtained using the proposed cipher is shown in the next section to be random. So, the distribution of the elements in the ciphertext $c_1, c_2, \ldots, c_n$ is also uniform. The resistance of the proposed stream cipher to a statistical attack seems very good.

4.4.7.4 Statistical test for randomness using NIST-STS test suite

The randomness of the obtained ciphertexts is tested using the NIST-STS test suite [65]. The NIST-STS test suite consists of various statistical tests. The details of each of such tests are discussed in section 4.3.8.2. We ran each test of the NIST-STS test suite using a significance level $\alpha = 0.01$ for three inputs (i) input contains all binary $0's$ `(0X00)`, (ii) input contains all binary $1's$ `(0XFF)`, and (iii) input contains `random` values. The size of each input file is 1048576 bits. We generated 1000 binary sequences (ciphertexts) for each of these three files using 1000 different keystreams. Now, these 1000 binary sequences of each file are tested using NIST-STS test suite. The results of each of these tests are shown in Table 4.10. The first column of the table lists the name of the tests carried out. The proportion of sequences that passed a statistical test at $\alpha = 0.01$ significance level for all `0X00`, all `0XFF`, and `random` inputs are listed in columns second, third, and fourth, respectively. According to the experimental results, as shown in Table 4.10, It can be observed that on average, 98.99% of sequences pass each of these tests for the significance level $\alpha = 0.01$, so, it can be concluded that the proposed cipher produces highly random binary sequences.

Table 4.10: NIST-STS test results for the 1000 ciphertexts produced by the new cipher for variant inputs.

Tests	Proportion of success out of 1000 for all `0X00` input	Proportion of success out of 1000 for all `0XFF` input	Proportion of success out of 1000 for `random` input
Frequency	0.988	0.996	0.983
Block frequency	0.978	0.994	0.989
Cumulative sum	0.980	0.991	0.991
Runs	0.972	0.993	0.990
Longest run	0.997	0.995	0.995
Rank	0.996	0.989	0.991
Discrete fourier transform	0.992	0.993	0.990
Non-overlapping template	0.979	0.989	0.988
Overlapping template	0.996	0.992	0.991
Universal statistical	0.988	0.988	0.988
Approximate entropy	0.987	0.990	0.992
Random excursion	0.989	0.985	0.987
Random excursion variants	0.992	0.987	0.989
Serial	0.991	0.992	0.991
Linear complexity	0.992	0.987	0.989

4.5 MQG-PRNG and non-associative quasigroup based stream cipher

This section discusses a modified version of the previous ciphers, discussed in sections 4.3 & 4.4. The main goal of proposing this cipher is to reduce the space complexities of the previous ciphers. This can be done using a smaller order quasigroup instead of a larger one. And hence, we carried it out using a quasigroup of order 16 instead of a quasigroup of order 256 that was used in the previous ciphers. It is a symmetric key cipher and uses 16 quasigroups of order 16 and their inverse quasigroups in encryption and decryption algorithms, respectively. These 16 quasigroups are generated during the encryption/decryption process by permuting the rows of the original non-associative quasigroup $Q = (\mathbb{Z}_{16}, *)$ of order 16, where the size of each quasigroup is 128 bytes.

That is, the proposed cipher leverages the space of a single quasigroup and uses 16 quasigroups. As a result, the space required by 16 quasigroups (2048 bytes) is reduced to that of a single quasigroup (128 bytes).

The proposed cipher consists of three parts: (i) a keystream generation algorithm, (ii) an encryption algorithm, and (iii) a decryption algorithm. The cipher uses a keystream K', generated by a keystream generation algorithm (or a pseudo-random number generator), which is based on multiple (16) quasigroups, named here as MQG-PRNG. The MQG-PRNG is also described in this section. Both the schemes (MQG-PRNG and encryption/decryption algorithms) are iterative and use a different set of 16 quasigroups of order 16. Each iteration of both the schemes produces 16 nibbles (64 bits) of output. That is, for each iteration, MQG-PRNG generates a 16 nibbles (64 bits) of the keystream, and the cipher encrypts/decrypts a 16 nibbles of plaintext/ciphertext. The randomness of the pseudo-random number (keystream K') generated by MQG-PRNG is analyzed using the NIST-STS test suite. We noted that the obtained keystream K' is highly random.

4.5.1 Generation of quasigroups

Our proposed schemes use 16 quasigroups of order 16 for both the keystream generation and the encryption/decryption algorithms. Let $Q = (\mathbb{Z}_{16}, *)$ be an original quasigroup of order 16 over set $\mathbb{Z}_{16} = \{0, 1, \ldots, 15\}$. By permuting the rows of the multiplication (or operation) table of the quasigroup Q, we arrive at another quasigroup. So, by permuting the rows of the quasigroup Q, 16! quasigroups can be created. That is, these 16! quasigroups are the result of permutations of rows of the original quasigroup Q. Note that both schemes (encryption/decryption and keystream generation algorithms) use only 16 quasigroups. So, any 16 out of the total 16! quasigroups can be selected. Let the selected quasigroups be denoted by $Q_0 = (\mathbb{Z}_{16}, *_0), Q_1 = (\mathbb{Z}_{16}, *_1), ..., Q_{15} = (\mathbb{Z}_{16}, *_{15})$, where $*_0, *_1, ..., *_{15}$ are the quasigroup operations corresponding to $Q_0, Q_1, ..., Q_{15}$, respectively. Note that all these 16 quasigroups need not be stored. This is because each quasigroup consists of the same rows but in a different order (permutation).

The proposed schemes are iterative in nature, and each iteration of the proposed schemes uses only one quasigroup out of 16 quasigroups. Also, a quasigroup may be used in more than one iteration. This is decided by a 16×1 multiplexer used in each iteration of the proposed schemes. Multiplexer is a combinational circuit that has a

maximum of n input values for k selection lines and it produces a single output for each input, where $n = 2^k$. This multiplexer can also be used to select a quasigroup of order n by permuting the rows of an original quasigroup $Q = (\mathbb{Z}_n, *)$, given in section 2.2.25 of Chapter 2. Here, in this case, $k = 4$. So, as in the case of Equation 2.24 of Chapter 2, the following equation can be used to select a total of 16 quasigroups of order 16

$$Q_i = (R_{(0+\mathbf{Comp})\, mod\, 16}, R_{(1+\mathbf{Comp})\, mod\, 16}, \ldots, R_{(15+\mathbf{Comp})\, mod\, 16}) \tag{4.3}$$

where $R_0, R_1, \ldots, R_{15}$ denote row numbers of the original quasigroup $Q = (\mathbb{Z}_{16}, *)$, $Q_i \in \{Q_0, Q_1, \ldots, Q_{15}\}$, $0 \leq \mathbf{Comp}, i \leq 15$, and the value of **Comp** is determined as follows: Let SD be a seed (or an intermediate seed) of 64 bits (16 nibbles), which is used to generate a 64-bit of keystream K' in each iteration of the keystream generation algorithm, described later in section 4.5.2. Let both the seed SD and the keystream K' of 16 nibbles be organized as a 4×4 matrix of s_j's as shown in Figure 4.10, $0 \leq j \leq 15$. Then, the value of **Comp** is calculated as

s_0	s_1	s_2	s_3
s_4	s_5	s_6	s_7
s_8	s_9	s_{10}	s_{11}
s_{12}	s_{13}	s_{14}	s_{15}

Figure 4.10: Representation of SD or keystream K' of length 16 nibbles.

$$\begin{aligned} \mathbf{Comp} &= \mathtt{Temp}_0 \oplus \mathtt{Temp}_1 \oplus \mathtt{Temp}_2 \oplus \mathtt{Temp}_3, \\ \text{where,} \mathtt{Temp}_0 &= s_0 \oplus s_4 \oplus s_8 \oplus s_{12}, \\ \mathtt{Temp}_1 &= s_1 \oplus s_5 \oplus s_9 \oplus s_{13}, \\ \mathtt{Temp}_2 &= s_2 \oplus s_6 \oplus s_{10} \oplus s_{14}, \\ \mathtt{Temp}_3 &= s_3 \oplus s_7 \oplus s_{11} \oplus s_{15}. \end{aligned} \tag{4.4}$$

The **Comp** is a compression function, which is used in both the keystream generation and the encryption/decryption algorithms that compresses a 64-bit value to a 4-bit value, denoted by x_0, x_1, x_2 and x_3. These four bits are being considered as the selection

lines of the 16×1 multiplexer employed in both schemes (encryption/decryption and keystream generation algorithms), which are updated by the current iteration key of the keystream K' in the case of encryption algorithm; whereas it is the current iteration seed `SD` (or intermediate `SD`) in the case of the keystream generation algorithm. The multiplexer selects or generates a quasigroup based on the current state of selection lines x_0, x_1, x_2, x_3. The value of x_α is either 0 or 1, where x_0 and x_3 are the least and the most significant bits, $0 \leq \alpha \leq 3$. If $x_3 = 0, x_2 = 0, x_1 = 0, x_0 = 0$, then the multiplexer selects or generates a quasigroup Q_0. If $x_3 = 0, x_2 = 0, x_1 = 0, x_0 = 1$, then the multiplexer selects or generates a quasigroup Q_1, and so on.

Note that using the same equation (Equation 4.3) to generate the row's permutations on the left inverse quasigroup of the quasigroup used in encryption, the decryption algorithm that uses this left inverse quasigroup gives the correct result, see Theorem 1 in Chapter 2.

4.5.2 Generation of keystream

The new stream cipher uses a keystream of size as long as the plaintext/ciphertext. To generate such a long keystream, we used a pseudo-random number generator based on multiple quasigroups of order 16, named here as MQG-PRNG. It uses a seed $\texttt{SD} = (IV, \texttt{Counter})$ of 64 bits (16 nibbles). The `SD` is a combination of an initialization value (IV) and a `Counter`, where each of IV and `Counter` is 64 bits in size. For a cryptographically secure pseudo-random generator, we need both IV and `Counter` to be random so that the corresponding output is unpredictable. For generating the keystream, the MQG-PRNG uses 16 quasigroups of order 16. These 16 quasigroups are generated during the keystream generation based on an original non-associative quasigroup $Q = (\mathbb{Z}_{16}, *)$ of order 16. The algorithm of MQG-PRNG is implemented using the Cipher Block Chaining (CBC) mode of operation. That is, the output of an iteration of the MQG-PNRG is fed into the next iteration of MQG-PRNG as intermediate IV. Also, the `Counter` is increased by one. Each iteration of MQG-PRNG uses either `SD` or intermediate `SD` of 64 bits along with a quasigroup $Q_i = (\mathbb{Z}_{16}, *_i)$, which is randomly selected out of 16 quasigroups, $1 \leq i \leq 16$. Selection of this quasigroup is based on a 16×1 multiplexer employed in each iteration of the MQG-PRNG. The workflow of the keystream generation is given in Figure 4.11. This generates the keystream $K' = k_1\, k_2\, k_3 \ldots k_n$, where each k_i is a nibble value and will be used to

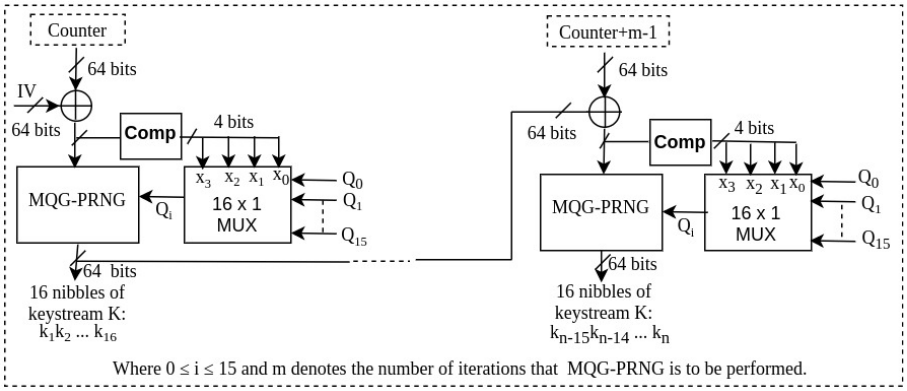


Figure 4.11: Workflow of keystream generation using MQG-PRNG.

encrypt a nibble value of the plaintext. The keystream generation algorithm can also be implemented using other modes of operation such as Cipher Feedback (CFB) mode, Output Feedback (OFB), and Counter (CTR) mode. Each mode of operation has their advantages and disadvantages [67].

Let $\texttt{SD} = (s_0, s_1, \ldots, s_{15})$ be a seed of 64 bits (16 nibbles), where each s_i is a nibble value for $0 \leq i \leq 15$. The algorithm of MQG-PRNG uses two levels of e-`transformation` for generating the keystream K'. Note that the algorithm of MQG-PRNG is almost the same as that of the algorithm of QG-PRNG, discussed in section 4.4.1. This is because both (MQG-PRNG and QG-PRNG) use two successive levels of e-`transformation` on $\texttt{SD}/IV$ (or intermediate $\texttt{SD}/IV$) to arrive at the required keystream K'. These two levels of the e-`transformation` are also described in section 4.4.1. However, some of the differences between QG-PRNG and MQG-PRNG are as follows:

1. QG-PRNG uses a single quasigroup of order 256, while MQG-PRNG uses 16 quasigroups of order 16, where these 16 quasigroups are generated based on an original non-associative quasigroup Q.

2. QG-PRNG uses an IV of 128 bits (16 bytes), while MQG-PRNG uses an SD of 64 bits (16 nibbles).

So, based on the algorithm of QG-PRNG, which is given in Algorithm 4 of section 4.4.1, the algorithm of MQG-PRNG is designed and it is given in Algorithm 5. In this algorithm, N denotes the total number of iterations that the MQG-PRNG is to be

Algorithm 5: Pseudocode of the MQG-PRNG based on CBC mode

Input: 1. Original $\texttt{SD} = (s_0, s_1, \ldots, s_{15})$ of 64 bits (16 nibbles).
2. A non-associative quasigroup $Q = (\mathbb{Z}_{16}, *)$ of order 16.
3. The initial leaders $a, b \in \mathbb{Z}_{16}$

Output: Each iteration produces 16 nibbles (64 bits) of keystream.

[1] **for** $I = 0$ *to* $N - 1$ **do**
[2] $\quad \texttt{SD}_I = IV \oplus \texttt{Counter}$;
[3] $\quad Q_i$ =selection of a quasigroup based on I^{th} iteration $\texttt{SD}_I$;
[4] $\quad$ `Followed by successive two levels of` e`-transformation specified in line numbers from` [3] `to` [10] `of Algorithm` 4;
[5] $\quad IV = \texttt{SD}_I$;
[6] $\quad \texttt{Counter} = \texttt{Counter+1}$;

performed, Q_i denotes a selected quasigroup of order 16 based on current $\texttt{SD}_I$, and its corresponding quasigroup operation is denoted by $*_i$. $\texttt{SD}_I$ denotes the I^{th} iteration seed value (or intermediate SD). For the 0^{th} iteration, $\texttt{SD}_0$ is SD itself. In each iteration of MQG-PRNG, line number [4] performs both levels of e-transformation on the $\texttt{SD}_I$, which is defined in Algorithm 4. That is, both the levels of e-transformation update $\texttt{SD}_I$ in each iteration of MQG-PRNG and the results are stored in the same $\texttt{SD}_I$. Also, the recent $\texttt{SD}_I$ is fed into the next iteration of MQG-PRNG. Each iteration of MQG-PRNG generates 64 bits (16 nibbles) of keystream and is repeated until the size of keystream K' is the same as that of the plaintext/ciphertext. Since the encryption/decryption algorithm requires the keystream to be random, MQG-PRNG ensures this.

4.5.3 Analysis of the MQG-PRNG

The key elements of MQG-PRNG are the initialization vector IV of 64 bits, Counter of 64 bits, and 16 quasigroups of order 16. These 16 quasigroups are generated based on an original non-associative quasigroup Q that acts as a secret key. It can be seen that the maximum number of non-associative quasigroups of order 16 is bounded above by 2^{456} (see Equation 2.9 of section 2.2.3 in Chapter 2). So, the algorithm of MQG-PRNG can generate a maximum of 2^{456} possible keystreams K' for each seed SD. Since quasigroup Q functions as the core of the proposed scheme, it is kept secret. If the quasigroup Q

used is not secret, it is easy for the attacker to crack the proposed scheme against a chosen-plaintext attack, illustrated later in section 4.5.7.3. In order to determine the employed quasigroup, an attacker can apply the brute-force methods as follows:

• *Quasigroup attack*: The MQG-PRNG uses a non-associative quasigroup of order 16, which acts as a secret key, and a maximum number of non-associative quasigroups of order 16 is 2^{456} (see Equation 2.9 of section 2.2.3 in Chapter 2). Assuming an attacker uses a supercomputer and tries $537.21 \times 10^{15} \approx 2^{58}$ quasigroups per second, then the attacker needs around 2^{373} years to determine the employed quasigroup. Note that nowadays supercomputers can execute 537.21 PFLOPS $= 537.21 \times 10^{15} \approx 2^{58}$ floating-point operations per second (FLOPS) [76].

4.5.3.1 Avalanche effect of the keystream K'

In this subsection, we looked at how the MQG-PRNG modifies the bits in a pseudo-random sequence (keystream) when the seed (SD) is modified slightly. Whenever a single bit of a seed is modified (from 1 to 0 or from 0 to 1), and if a pseudo-random number generator (PRNG) changes the pseudo-random sequence's bits with a probability of approximately 50%, then the PRNG would provide good diffusion power and protection from some slide and related-key attacks.

We conducted various experiments to evaluate the performance of the MQG-PRNG against the avalanche effect on the keystream K'. In each experiment, we chose special cases of the seeds, changed a particular bit in each of the seeds, generated the corresponding keystreams, and we compared them with the original keystream to see how they differ from each other.

Here, we have given some results from one of the experiments conducted during this research. The seeds that were used in this experiment are given in Table 4.11. Table 4.12 shows the experimental results for the keystreams of sizes 1024 bits and 4096 bits. Table 4.12 shows the minimum, maximum, and average (mean) change percentage in each of the generated keystreams whenever a single bit is changed in the corresponding seed. From the results given in Table 4.12, we see that change of one bit in the seed changed more than 49% of the generated keystream bits. This is quite close to the ideal value of 50%. Also, the amount of dispersion (or variation) of all possible changes in each of the generated keystream bits is measured using the

standard deviation, denoted by *sd*. The results show that the change percentage of each of the generated keystream bits is not only close to the mean but also the value of *sd* is decreasing as the size of the generated keystream increases. Since a low *sd* indicates that the values tend to be close to the mean (also called the expected value) of the set, while a high *sd* indicates that the values are spread out over a wider range. Hence, we can conclude that the MQG-PRNG produces good random sequences. That is, it has a higher diffusion and hence provides protection from some slide and key-related attacks.

Table 4.11: Seeds used for MQG-PRNG in binary format.

Seeds	
$seed_1$	00000000000000000000000000000000000000000000000000000000000000000000000000000000
$seed_2$	11111111111111111111111111111111111111111111111111111111111111111111111111111111
$seed_3$	10010110100101101001011010010110100101101001011010010110100101101001011010010110
$seed_4$	11111111111111111111111111111111000000000000000000000000000000000000000000000000
$seed_5$	00000000000000000000000000000000111111111111111111111111111111111111111111111111
$seed_6$	00000000000000001111111111111111000000000000000011111111111111111111111111111111
$seed_7$	01010101010101010101010101010101010101010101010101010101010101010101010101010101
$seed_8$	00000000111111110000000011111111000000001111111100000000111111110000000011111111

4.5.3.2 Statistical test of MQG-PRNG using NIST-STS test suite

We have tested the quality of the obtained pseudo-random sequences using the NIST-STS test suite. We have used the NIST Spec. Publ. 800-22 rev. 1a package with significance level $\alpha = 0.01$ that consists of 15 different statistical tests [65]. More details of the NIST-STS test suite are given in section 4.3.8.2. We ran each test of NIST-STS for 1000 obtained pseudo-random sequences produced by the MQG-PRNG, where the size of each sequence is 10^6 bits. The results of each of these tests are shown in Table 4.13. First column of the table lists the name of the tests carried out. The number of accepted (success) sequences, the number of rejected (failures) sequences, and the proportion of sequences that passed a statistical test at $\alpha = 0.01$ significance level are listed in columns second, third, and fourth, respectively. According to the experimental results as shown in Table 4.13, it can be observed that 98.3% of sequences

Table 4.12: Avalanche effect of keystream for the different seeds.

Seeds of 128 bits	Keystream of 1024 bits	Keystream of 4096 bits	Seeds of 128 bits	Keystream of 1024 bits	Keystream 4096 bits
$seed_1$	$min = 44.73\%$, $max = 54.39\%$, $avg = 49.89\%$, $sd = 1.67$	$min = 48.05\%$, $max = 51.29\%$, $avg = 49.77\%$, $sd = 0.73$	$seed_5$	$min = 46.09\%$, $max = 54.00\%$, $avg = 49.96\%$, $sd = 1.63$	$min = 48.71\%$, $max = 51.66\%$, $avg = 50.11\%$, $sd = 0.76$
$seed_2$	$min = 47.27\%$, $max = 52.25\%$, $avg = 49.84\%$, $sd = 1.25$	$min = 48.46\%$, $max = 51.56\%$, $avg = 50.03\%$, $sd = 0.69$	$seed_6$	$min = 46.97\%$, $max = 53.52\%$, $avg = 50.25\%$, $sd = 1.44$	$min = 47.83\%$, $max = 51.68\%$, $avg = 49.86\%$, $sd = 0.91$
$seed_3$	$min = 46.68\%$, $max = 53.32\%$, $avg = 50.21\%$, $sd = 1.45$	$min = 47.73\%$, $max = 52.37\%$, $avg = 49.84\%$, $sd = 0.75$	$seed_7$	$min = 46.48\%$, $max = 53.81\%$, $avg = 50.25\%$, $sd = 1.57$	$min = 48.58\%$, $max = 53.03\%$, $avg = 50.26\%$, $sd = 0.85$
$seed_4$	$min = 46.68\%$, $max = 54.98\%$, $avg = 50.27\%$, $sd = 1.63$	$min = 48.10\%$, $max = 51.32\%$, $avg = 49.91\%$, $sd = 0.77$	$seed_8$	$min = 46.39\%$, $max = 52.64\%$, $avg = 49.75\%$, $sd = 1.49$	$min = 47.71\%$, $max = 51.29\%$, $avg = 49.91\%$, $sd = 0.75$

pass each of these tests for the significance level $\alpha = 0.01$, implying that the pseudo-random sequences we obtained are random. Also, if we compare the performance of the MQG-PRNG against NIST-STS test with that of the AES-256, whose results are given in Table 4.7 of section 4.4.1.1, we see that the randomness of the outputs of the MQG-PRNG and AES-256 are comparable to each other.

4.5.4 Encryption Algorithm

The encryption algorithm of this cipher is almost the same as that of the encryption algorithm of the previous cipher, discussed in section 4.4.2. This is because, it also uses the ne^l-`transformation`. The only difference is that for the ne^l-`transformation`, it uses 16 quasigroups $Q_i = (\mathbb{Z}_{16}, *_i)$ of order 16 instead of a single quasigroup $Q = (\mathbb{Z}_{256}, *)$ of order 256. These 16 quasigroups are generated based on an original non-associative quasigroup $Q = (\mathbb{Z}_{16}, *)$ of order 16. The encryption algorithm works on one nibble (4 bits) of data. It encrypts a plaintext $P = P_{[1,\ldots,n]} = p_1\, p_2\, \ldots p_n$ with the

Table 4.13: Results of the NIST-STS test for 1000 sequences generated by MQG-PRNG.

Tests	Number of success	Number of failures	Proportion of success out of 1000
Frequency	991	9	0.991
Block frequency	983	17	0.983
Cumulative sum	992	8	0.992
Runs	993	7	0.993
Longest run	988	12	0.988
Rank	993	7	0.993
Discrete fourier transform	988	12	0.988
Non-overlapping template	988	12	0.988
Overlapping template	991	9	0.991
Universal statistical	988	12	0.988
Approximate entropy	988	12	0.988
Random excursion	990	10	0.990
Random excursion variants	991	9	0.991
Serial	988	12	0.988
Linear complexity	993	7	0.993

keystream $K' = K'_{[1,\ldots,n]} = k_1\, k_2 \ldots k_n$, and produces the ciphertext $C = C_{[1,\ldots,n]} = c_1\, c_2\, c_3 \ldots c_n$ as follows:

$$c_j = k_j *_i p_j$$

where p_j, k_j, and c_j are characters of 4 bits (1-nibble), $1 \leq j \leq n$, and $*_i$ is one the quasigroup operations corresponding to the employed quasigroup Q_i, $0 \leq i \leq 15$. The encryption algorithm encrypts 16 nibbles (64 bits) of plaintext P in one iteration. This is because the MQG-PRNG generates 16 nibbles (64 bits) of keystream K' in one iteration. For each iteration, it uses a quasigroup Q_i randomly selected out of 16 quasigroups, which is decided by a 16×1 multiplexer used, $0 \leq i \leq 15$. The workflow of the encryption algorithm is shown in Figure 4.12. The pseudocode of the encryption algorithm is given in Algorithm 6. In this algorithm, N denotes the total number of iterations that the encryption algorithm is to be performed, I denotes the I^{th} iteration

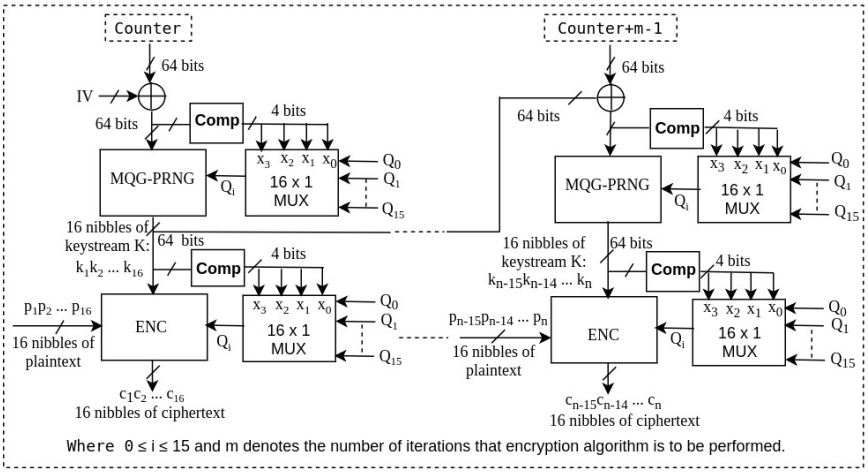


Figure 4.12: Workflow of the encryption algorithm.

Algorithm 6: Pseudocode of the encryption algorithm

Input: 1. Plaintext $P = P_{[1,\ldots,n]} = p_1, p_2, \ldots, p_n$ of n nibbles to be encrypted.
2. A non-associative quasigroup $Q = (\mathbb{Z}_{16}, *)$ of order 16.

Output: Ciphertext $C = C_{[1,\ldots,n]} = c_1, c_2, \ldots, c_n$ of n nibbles.

[1] **for** $I = 0$ *to N-1* **do**

[2] $K'_{[16\times I+1,\ldots,16\times I+16]} = \text{MQG-PRNG}(I)$;

[3] Q_i = generation of a quasigroup based on I^{th} iteration keystream $K'_{[16\times I+1,\ldots,16\times I+16]}$;

[4] $C_{[16\times I+1,\ldots,16\times I+16]} = K'_{[16\times I+1,\ldots,16\times I+16]} *_i P_{[16\times I+1,\ldots,16\times I+16]}$;

out of N, and $*_i$ is one of the quasigroup operation corresponding to the employed quasigroup Q_i of order 16, $0 \leq i \leq 15$.

4.5.5 Decryption Algorithm

The decryption algorithm is the reverse process of the encryption algorithm. It recovers the plaintext P from the ciphertext C. It uses nd^l-`transformation` along with 16 left inverse quasigroups $LIQ_i = (\mathbb{Z}_{16}, \backslash_i)$ of order 16, where $\backslash_i$ is one of the left inverse quasigroup operations corresponding the employed quasigroup LIQ_i, $0 \leq i \leq 15$. Note that these 16 left inverse quasigroups are the left inverses of the quasigroups that were used in the encryption algorithm. Also, It uses the 16 quasigroups $Q_i = (\mathbb{Z}_{16}, *_i)$ as that used in the encryption algorithm for generating the keystream K' since both encryption

and decryption use the same keystream K', where $*_i$ is one of the quasigroup operations corresponding the employed quasigroup Q_i, $0 \leq i \leq 15$. The decryption algorithm also works on one nibble (4 bits) of data. It decrypts a ciphertext $C = C_{[1,\ldots,n]} = c_1\, c_2\, c_3 \ldots c_n$ using the same keystream $K' = K'_{[1,\ldots,n]} = k_1\, k_2 \ldots k_n$ as that used in the encryption algorithm, and recovers the original plaintext $P = P_{[1,\ldots,n]} = p_1\, p_2 \ldots p_n$ as follows:

$$p_j = k_j \setminus_i c_j$$

where p_j, k_j, and c_j are characters of 4 bits (1-nibble), $1 \leq j \leq n$, and $\setminus_i$ is one of the left inverse quasigroup operations corresponding to the employed quasigroup LIQ_i, $0 \leq i \leq 15$. The workflow of the decryption algorithm is shown in Figure 4.13. Like

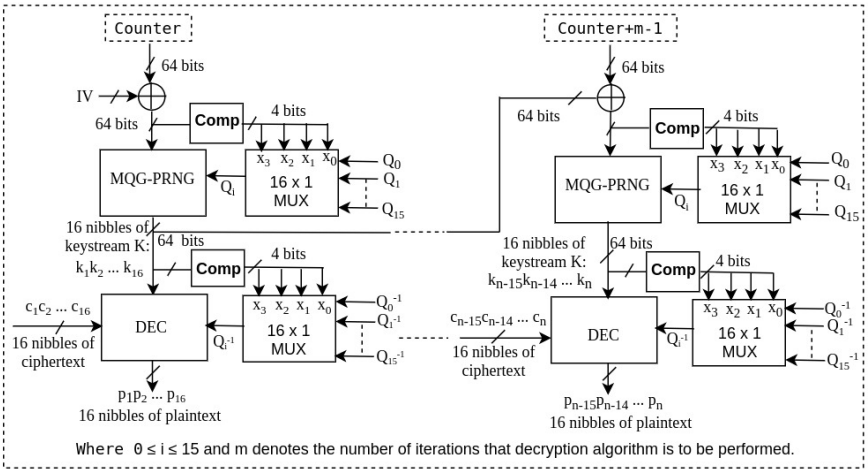


Figure 4.13: Workflow of the decryption algorithm.

the encryption algorithm, the decryption algorithm also decrypts 16 nibbles (64 bits) of ciphertext in one iteration. This is because it also uses the MQG-PRNG algorithm for generating the keystream K'. The pseudocode of the decryption algorithm is given in Algorithm 7. In this algorithm, N denotes the total number of iterations that the decryption algorithm is to be performed, I denotes the I^{th} iteration out of N, and $\setminus_i$ is one of the left inverse quasigroup operations corresponding to the employed quasigroup LIQ_i of order 16, $0 \leq i \leq 15$.

Algorithm 7: Pseudocode of the decryption algorithm

Input: 1. Ciphertext $C = C_{[1,\ldots,n]} = c_1, c_2, \ldots, c_n$ of n nibbles to be decrypted.
2. A non-associative quasigroup $Q = (\mathbb{Z}_{16}, *)$ of order 16 for generating the keystream K'.
3. A left inverse quasigroup $LIQ = (\mathbb{Z}_{16}, \backslash)$ of order 16 for decryption, where LIQ is the left inverse of Q.

Output: Plaintext $P = P_{[1,\ldots,n]} = p_1, p_2, \ldots, p_n$ of n nibbles.

[1] **for** $I = 0$ *to N-1* **do**

[2] $K'_{[16\times I+1,\ldots,16\times I+16]} = \text{MQG-PRNG}(I)$;

[3] LIQ_i = generation of a left inverse quasigroup based on I^{th} iteration keystream $K'_{[16\times I+1,\ldots,16\times I+16]}$;

[4] $P_{[16\times I+1,\ldots,16\times I+16]} = K'_{[16\times I+1,\ldots,16\times I+16]} \backslash_i C_{[16\times I+1,\ldots,16\times I+16]}$;

4.5.6 Performance analysis

The cipher is implemented in C++, and its performance is analyzed against both the space and the time (speed) complexities by comparing it with those of the previous ciphers discussed in sections 4.3 & 4.4, and some existing quasigroup based stream ciphers introduced in literature [12, 28, 43, 59, 60, 81]. To analyze the performance of this cipher, we used the same tools and system configurations as we used to analyze the previous cipher, discussed in section 4.3.7. The cipher is run 1000 times for different inputs and calculated the average execution time in seconds. Experimental results for both time and space complexities are given in Table 4.14. If we compare this result with those of the previous results shown in sections 4.3.7 & 4.4.6, it can be observed that in terms of memory requirement, except Edon-80 [28], this cipher is not only less expensive than both the previous ciphers discussed in sections 4.3 & 4.4, but also less expensive than other existing ciphers introduced in literature [12, 43, 59, 60, 81]. In contrast, it is slightly slower than our previous cipher discussed in section 4.4 and an existing cipher introduced in literature [43], but faster than our first cipher discussed in section 4.3 and other existing ciphers introduced in literature [12, 28, 59, 60, 81].

Table 4.14: Time and space complexities of the proposed cipher.

Stream cipher	Execution time in seconds			Space in KB	
	1.22 MB	2.11 MB	6.01 MB	Encryption	Decryption
Proposed cipher	0.07	0.13	0.39	0.13	0.25

4.5.7 Security analysis

The key elements of the proposed cipher are MQG-PRNG and 16 quasigroups of order 16. These 16 quasigroups are generated based on an original non-associative quasigroup Q that acts as a secret key. Note that the maximum number of non-associative quasigroups of order 16 is bounded above by 2^{456} (see Equation 2.9 of section 2.2.3 in Chapter 2). So, the encryption/decryption algorithm can use a maximum of 2^{456} secret quasigroups. Since quasigroup Q functions as the core of the proposed cipher, it is kept as a secret. If the quasigroup Q used is not secret, it is easy for the attacker to crack the cipher against a chosen-plaintext attack as illustrated later in section 4.5.7.3.

The proposed cipher keeps both the non-associative quasigroup Q and the keystream K' secret. This is because the security of the proposed cipher depends not only on the keystream K' but also on the employed quasigroup Q, which can be changed rapidly. Due to this, the proposed cipher can be seen as a family of stream ciphers parameterized by a quasigroup Q. The proposed cipher uses any 16 quasigroups out of 2^{456}. These 16 quasigroups are generated from the original non-associative quasigroup Q by a circular shift of rows by a constant amount **Comp**, see Equation 4.3. Each iteration of the proposed cipher encrypts/decrypts 64 bits of the plaintext/ciphertext and uses a quasigroup randomly out of these 16 quasigroups, which is decided by a 16×1 multiplexer employed. Note that a quasigroup may be used in more than one iteration. The selection of quasigroup depends on the current iteration 64 bits of the keystream K'. Let N be the number of iterations that the proposed cipher performs to encrypt/decrypt a $N \times 64$ bits of the plaintext/ciphertext. So, the number of ways to use any 16 quasigroups out of 2^{456} in N iteration is equal to $2^{456} \times 16^{N-1}$. For example, if $N = 16$, then a total of $2^{456} \times 16^{15}$ ways possible to use 16 quasigroup in 16 iterations to generate a 1024 bits of plaintext/ciphertext. In order to determine the employed quasigroups, an attacker can apply the brute-force methods as follows:

• *Quasigroups attack*: The proposed cipher uses 16 quasigroups of order 16, and a maximum number of quasigroups of order 16 is 2^{456}. Assuming the cipher performs N iterations, the number of possibilities that one needs to search to determine the quasigroups used in these iterations is $2^{456} \times 16^{N-1}$, where $N \geq 1$. It can be seen that as N increases, the complexity of this attack increases. For example, if $N = 16$, the attack complexity of determining the employed quasigroups that generate 1024 bits of the plaintext/ciphertext is equal to $2^{456} \times 16^{15} = 2^{516}$. That is if an attacker uses a supercomputer and tries $537.21 \times 10^{15} \approx 2^{58}$ quasigroups per second, then the attacker needs around 2^{433} years to determine the employed quasigroups. This is because, nowadays supercomputers can execute 537.21 PFLOPS $= 537.21 \times 10^{15} \approx 2^{58}$ floating-point operations per second (FLOPS) [76]

4.5.7.1 Reused key attack

This cipher is almost the same as that of our previous ciphers, discussed in sections 4.3 & 4.4. The only difference is that it uses 16 quasigroups of order 16 instead of a single quasigroup of order 256. Therefore, the analysis of this cipher, in the context of the reused key attack, is the same as that of the previous cipher, discussed in section 4.4.7.2. From the design of the cipher, we have

$$C_1 = K' *_i M_1$$
$$C_2 = K' *_i M_2$$

where C_1 and C_2 are the ciphertexts obtained using the proposed stream ciphers, and $*_i$ is one of the quasigroup operations corresponding to the employed quasigroup $Q_i = (\mathbb{Z}_{16}, *_i)$, $0 \leq i \leq 15$. If we use $\oplus$ operation instead of $*_i$, then the cipher becomes XOR-based stream cipher and the attacker in the reused key attack computes

$$C_1 \oplus C_2 = (K' \oplus M_1) \oplus (K' \oplus M_2) = M_1 \oplus M_2$$

where $\oplus$ is an XOR operation. That is, the obtained byte sequence does not depend on the keystream K'. In other words, if anyone encrypts two messages with the same key, they can recover $M_1 \oplus M_2$ which is a form of running key cipher. Even if neither message is known, as long as both the messages are in a natural language, such a cipher can

often be broken because of enough redundancy present in English and ASCII encoding. And a cryptanalyst easily can recover the original messages from

$$C_1 \oplus C_2 = M_1 \oplus M_2 \rightarrow M_1, M_2.$$

Hence, a cipher that uses the XOR operation for mixing the plaintext with the keystream would be vulnerable to reused key attack. But in our case:

$$C_1 *_i C_2 = (K' *_i M_1) *_i (K' *_i M_2) \neq M_1 *_i M_2.$$

This is because $*_i$ is one of the quasigroup operations corresponding to the employed quasigroup Q_i, $0 \leq i \leq 15$. Hence, the proposed cipher is resistant to reused key attack.

However, the proposed cipher would be vulnerable to reused key attack if the following properties hold:

(1) $t_1 \setminus (t_2 * t_3) = (t_1 \setminus t_2) * t_3$, $\forall\ t_1, t_2, t_3 \in \mathbb{Z}_n$.

(2) $(t_1 * t_2)/t_3 = t_1 * (t_2/t_3)$, $\forall\ t_1, t_2, t_3 \in \mathbb{Z}_n$

where $Q = (\mathbb{Z}_n, *)$, $LIQ = (\mathbb{Z}_n, \setminus)$ and $RIQ = (\mathbb{Z}_n, /)$ be a quasigroup, its left inverse quasigroup, and its right inverse quasigroup, respectively. This is because the attacker can recover useful information from the following equations:

$$C_1 \setminus C_2 = (K' * M_1) \setminus (K' * M_2)$$

$$(K' * M_1) \setminus (K' * M_2) = C_1 \setminus C_2$$

$$(K' * M_2) = (K' * M_1) * (C_1 \setminus C_2), \text{ see Definition 2.2.6 of Chapter 2}$$

$$M_2 = K' \setminus [(K' * M_1) * (C_1 \setminus C_2)], \text{ see Definition 2.2.6 of Chapter 2}$$

$$M_2 = [K' \setminus (K' * M_1)] * (C_1 \setminus C_2), \text{ using property (1)}$$

$$M_2 = M_1 * (C_1 \setminus C_2), \text{ see identity (2.10) of Lemma 2.2.11, defined in Chapter 2}$$

Similarly, using the property (2) and C_1/C_2, we can also arrive at the following equation: $M_1 = (C_1/C_2) * M_2$.

We verified both the above properties (1) and (2) using all possible quasigroups of order up to 6 (see Table 2.5 of Chapter 2) and found that only associative quasigroups and their left/right inverse quasigroups satisfy these properties. It is a longstanding open problem to either verify or disprove, for large order quasigroups, that only the associative quasigroups possess the above-mentioned two properties. Note that the proposed cipher is designed based on a non-associative quasigroup. So, the proposed cipher is secure against reused key attack.

4.5.7.2 Time-Memory-Data Tradeoff attack

Time-memory-data tradeoff (TMDTO) attack is a powerful attack against stream cipher. It reduces the complexity of the brute-force (exhaustive key search) attack. Let

- N denotes the number of the internal states to be covered by the attacker (or the size of the search space).
- P denotes the time complexity of the preprocessing phase (or offline time complexity).
- M denotes the space (memory) complexity.
- T denotes the time complexity of the real-time phase (or online time complexity).
- D denotes the data complexity.

Martinand and Hellman [51] introduced a time-memory tradeoff (TMTO) method for breaking block ciphers using a tradeoff curve $TM^2 = N^2$, where $N \geq T \geq 1$. Later, Babbage and Golic [2, 30] devised different time-memory-data tradeoff method for breaking stream ciphers with new tradeoff curves $TM = N$, $P = M$, $D \geq T \geq 1$. And we referred to it as the BS attack. Also, Biryukov and Shamir introduced a different tradeoff curve with better bounds to improve the attack complexity of the TMDTO against stream ciphers, which is $TM^2D^2 = N^2$, $P = \frac{N}{D}$, $T \geq D^2 \geq 1$ [10]. And we referred to it as the BS attack.

The time-memory-data tradeoff (TMDTO) attack is an extension of the time-memory tradeoff (TMTO) attack that aims to achieve better tradeoffs by increasing the number of data required. It was first successfully applied on A5/1 stream cipher [11]. Generally, a TMDTO attack is performed in two phases: (i) preprocessing phase (also called the offline phase) and (ii) the real-time phase (also called the online phase). In the preprocessing phase, the attacker recomputes several tables with memory complexity M. Each of these tables stores the mapping between different internal states (secret keys) and the corresponding keystreams with preprocessing time complexity P, which allows for reducing the online time complexity T. In the real-time phase, the attacker tries to invert the function mapping of the internal states of a stream cipher to a segment of the keystream output by intercepting D keystreams and searching them in the table with time complexity T, expecting to get some matches and recover the corresponding input (internal state).

Using TMDTO attacks, an attacker can try to reconstruct the internal state of the stream cipher to recover the secret key. That is, using the internal state update process, an attacker could obtain not only subsequently generated keystreams by running the cipher forward if he or she has reconstructed an internal state at any particular time but also recover previous states iteratively and further get the underlying secret key by running the cipher backward. Note that the secret keys are nothing but the quasigroups of order 16 that the proposed cipher uses in both the keystream generation and the encryption algorithm (see Figure 4.12). Also, each iteration of both the keystream generation and the encryption algorithm uses a different quasigroup. That is, for each of the 2^{456} possible non-associative quasigroups of order 16, the cipher has 2^{456} states (see Equation 2.9 of section 2.2.3 in Chapter 2). That is, $N = 2^{456} \times 2^{456} = 2^{912}$ (see Figure 4.12); which is 2 times the secret key length. So, the attack complexity of the proposed stream cipher against the BG attack would be $T = D = M = N^{\frac{1}{2}} = 2^{456}$, which is equivalent to the exhaustive key search attack. Hence, the proposed cipher is resistant to BG attack.

An attacker can also recover the secret key directly using the pre-computed table that stores the keystream segments for different (key, seed) pairs [22]. Note that a secret key is nothing but a non-associative quasigroup used for encryption/decryption, and the maximum number of non-associative quasigroups of order 16 is 2^{456} (see Equation 2.9 of section 2.2.3 in Chapter 2). The attacker can search the table for a collision and recover some of the secret keys if he or she has some keystream data under different secret keys corresponding to these seeds. The tradeoff curves remain the same as that used to recover internal states, but N is modified to represent the size of the collection of all possible (key, seed) pairs, i.e. $N = 2^{520}$. This is because, in the proposed stream cipher, the size of the seed and key spaces are 2^{64} and 2^{456}, respectively. So, the attack complexity against BS and BG are $T = 2^{528}$ for $D = M = 2^{128}$ and $T = M = D = N^{\frac{1}{2}} = 2^{260}$, respectively [25]. But it is still impractical.

4.5.7.3 Chosen plaintext attack

This cipher is analyzed against a chosen-plaintext attack in the same way as the previous ciphers analyzed against a known-plaintext attack in sections 4.3.8.1 & 4.4.7.1. Suppose the cryptanalyst chooses the plaintext $P = p_1\, p_2\, \ldots p_n$, obtains the ciphertext $C = c_1\, c_2\, \ldots c_n$ corresponding to the chosen-plaintext, and tries to determine the keystream

K' employed in the encryption/decryption system. The cryptanalyst, then, for the keystream K' used in the stream cipher, must solve the following system of equations:

$$\left.\begin{aligned} c_1 &= k_1 *_i p_1 \\ c_2 &= k_2 *_i p_2 \\ &\cdots \\ c_n &= k_n *_i p_n \end{aligned}\right\} \tag{4.5}$$

where $*_i$ is one of the quasigroup operations corresponding to the employed quasigroup of order 16, $0 \leq i \leq 15$, and $k_1, k_2, \ldots, k_n$ are unknowns. Let us assume that the quasigroups Q_i used are known to everyone. Then, each of these above equations has a maximum of 16 solutions since the proposed cipher uses a maximum of 16 quasigroups. In other words, each of these equations has a unique solution for each quasigroup. So, the cipher can not resist this attack. But the employed quasigroups are kept secret. So, the above system of equations has as many solutions as there are the number of quasigroups of order 16. Hence determining the quasigroups makes it practically impossible. Therefore, the proposed cipher is resistant to chosen-plaintext attack.

Using a similar argument, it can be shown that the cipher is resistant to the chosen-ciphertext and known-plaintext attacks as well.

4.5.7.4 Statistical test for randomness

The randomness of the obtained ciphertexts is tested using the NIST-STS test suite [65]. The NIST-STS test suite consists of various statistical tests. The details of each of such tests are discussed in section 4.3.8.2. We ran each test of the NIST-STS test suite using a significance level $\alpha = 0.01$ for three inputs (i) input contains all binary $0's$ `(0X00)`, (ii) input contains all binary $1's$ `(0XFF)`, and (iii) input contains `random` values. The size of each input file is 10^6 bits. We generated 1000 binary sequences (ciphertexts) for each of these three files using 1000 different keystreams so that each file uses the same keystream. Now, for 1000 binary sequences of each file, separately, we ran each test of the NIST-STS 1000 times. The results of each of these tests are shown in Table 4.15. The first column of the table lists the name of the tests carried out. The proportion of sequences that passed a statistical test at $\alpha = 0.01$ significance level for all `0X00`, all `0XFF`, and `random` inputs are listed in columns second, third, and fourth, respectively. According to the experimental results, as shown in Table 4.15, we note that on average,

99.04% of sequences pass each of these tests. so, it can be concluded that the proposed cipher produces highly random ciphertexts.

Table 4.15: NIST-STS test results for the 1000 ciphertexts.

Tests	Proportion of success out of 1000 for all `0X00` input	Proportion of success out of 1000 for all `0XFF` input	Proportion of success out of 1000 for `random` input
Frequency	0.989	0.993	0.991
Block frequency	0.989	0.997	0.992
Cumulative sum	0.990	0.992	0.994
Runs	0.990	0.990	0.989
Longest run	0.994	0.996	0.987
Rank	0.989	0.990	0.992
Discrete fourier transform	0.993	0.989	0.992
Non-overlapping template	0.990	0.990	0.991
Overlapping template	0.986	0.988	0.992
Universal statistical	0.988	0.989	0.990
Approximate entropy	0.988	0.994	0.990
Random excursion	0.989	0.990	0.991
Random excursion variants	0.989	0.989	0.990
Serial	0.990	0.994	0.989
Linear complexity	0.987	0.986	0.990

4.5.8 Summary

In this chapter, we have proposed three stream ciphers based on quasigroups. The first cipher discussed in section 4.3, uses AES-256 for generating the keystream and a single quasigroup of order 256 for encrypting/decrypting the messages, while the second and third ciphers are discussed in sections 4.4 and 4.5, respectively. The second cipher uses QG-PRNG for generating the keystream and a single quasigroup of order 256 for encrypting/decrypting the messages, while the third cipher uses MQG-PRNG for generating the keystream and 16 quasigroups for encrypting/decrypting the messages. Use of multiple quasigroups contributes to increased security. This is because a

different quasigroup is used after a certain amount of plaintext/ciphertext. QG-PRNG and MQG-PRNG are pseudo-random number generators, which are also described in this chapter. To generate the keystreams, the QG-PRNG uses a single quasigroup of order 256, while MQG-PRNG uses 16 quasigroups of order 16. The randomness of the obtained keystreams produced by the QG-PRNG and MQG-PRNG is analyzed using the NIST-STS test suite. We noted that the generated keystream sequences are highly random.

Novelty of these stream ciphers is that they are resistant to the reused key attack as against the existing XOR-based stream ciphers. Hence a keystream can be reused multiple times, thereby overcoming the major hurdle that exists in the application of the XOR-based stream ciphers. Also, the ciphers are analyzed against various attacks, including the chosen-ciphertext attack, the chosen-plaintext attack, the known-plaintext attack, the reused-key attack, the statistical attack, and the time-memory-data tradeoff (TMDTO) attack. We observed that our ciphers are resistant to these attacks as well.

The performance of the proposed ciphers is analyzed by comparing them to each other and we found that our third cipher is slightly slower than our second cipher, but overall third cipher outperforms both the first and second ciphers. This is because the third cipher uses about 99% less memory (in bytes) than the first and second ciphers. In addition, if we compare the performance of the proposed ciphers with that of the existing quasigroup based stream ciphers [12, 28, 43, 59, 60, 81]. We observed that in most cases the proposed ciphers are more efficient than the existing quasigroup based ciphers [12, 28, 43, 59, 60, 81].

The randomness of the obtained ciphertexts produced by the proposed stream ciphers is analyzed by the NIST-STS test suite. We found that the obtained ciphertexts of the proposed ciphers are highly random.

Chapter 5

Block Ciphers Based on Multiple Quasigroups

This chapter discusses two schemes of block ciphers based on multiple quasigroups. They are symmetric key ciphers and use 16 *optimal S-boxes in the form of an optimal quasigroup of order* 16*. A maximum of* 16! *optimal quasigroups can be formed using the* 16 *S-boxes, these ciphers can be seen as a family of encryption systems parameterized by an optimal quasigroup. That is, the sender and the receiver agree on a cryptosystem by first deciding on the quasigroup. This chapter gives a brief overview of the ciphers, the structure of the ciphers, the details of building elements of the ciphers, and analyzes the performance and security of the ciphers.*

While describing these ciphers, the emphasis is on how the ciphers use Substitution Permutation Networks (SPN) to achieve confusion and diffusion of bits from the plaintext to the ciphertext and why they are an excellent alternative to the existing quasigroup-based block ciphers.

5.1 Introduction

As discussed in the last chapter, a stream cipher encrypts/decrypts one data item (bit/nibble/byte) of the plaintext/ciphertext at a time. In contrast, a block cipher encrypts/decrypts a fixed amount of plaintext/ciphertext at a time called a block. One significant difference between the block ciphers and the stream ciphers is that block ciphers are stateless, whilst stream ciphers are stateful. That is, a stream cipher

maintains an internal state to generate the next part of the keystream. A block cipher is also a symmetric key cipher that uses the same key for encryption and decryption. Block diagram of a block cipher is shown in Figure 5.1. A cipher takes a fixed-size

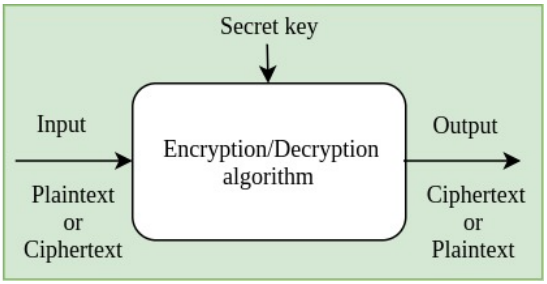


Figure 5.1: Block diagram of the block cipher.

block as input and produces an output block whose size is equal to the size of the input block. It also uses the same secret key for encrypting and decrypting messages. Block and key sizes usually depend on the cipher algorithm. For example, block and key sizes are 64 bits each in DES; whereas 128 bits or more in modern block ciphers.

A block cipher can not only be used as an encryption algorithm but it can also be used as a versatile building block for implementing a wide range of cryptographic applications, such as hash functions, message authentication codes, pseudo-random number generators, etc. Most modern-day block ciphers are designed based on permutation and substitution networks and are iterative in nature. In an iterated cipher, a round function and a key schedule must be specified, and a block of plaintext will be encrypted through N iterations of the same function. The output of the first block may be used to help encrypt the second block in what is called a *mode of operation*. For encrypting the long plaintext, a block cipher can use several modes of operation, such as ECB, CBC, CFB, etc. Different modes of operation offer different levels of protection against error propagation caused by transmission errors in the ciphertext [67].

5.2 Brief overview of the proposed block ciphers

This chapter proposes two cipher algorithms for encrypting/decrypting messages. The ciphers employ optimal quasigroups and their corresponding left inverse quasigroups. They are block ciphers and each of them uses 16 optimal quasigroups of order 16. Each of these 16 optimal quasigroups is constructed based on an original quasigroup $Q =$

$(\mathbb{Z}_{16}, *)$ of order 16. The description of the optimal quasigroup is given in section 2.2.16 of Chapter 2. The proposed ciphers use 16 S-boxes $S_0, S_1, \ldots S_{15}$ in the form of an optimal quasigroup $Q = (\mathbb{Z}_{16}, *)$ of order 16, shown in Table 2.8 of Chapter 2. Note that the size of each of these 16 S-boxes is 4×4 bits.

The first cipher [1] performs a total of 16 rounds to encrypt and decrypt a message of 128 bits. Each of these rounds uses one non-linear transformation (also called $\{e, d\}$-`transformation`) and one linear transformation. The non-linear transformation uses 8 bits of the secret key, also called a seed value or a sub-key, for processing a block of 128 bits of data. The details of the $\{e, d\}$-`transformation` are given in section 2.2.15 of Chapter 2.

The second cipher [2] performs a total of 17 rounds to encrypt and decrypt a message of 128 bits. Each of these rounds uses 128 bits round key along with one non-linear transformation (also called $\{ne^\ell, nd^\ell\}$-`transformation`) and two linear transformations. Note that the $\{ne^\ell, nd^\ell\}$-`transformation` is defined in section 2.2.15 of Chapter 2.

The algorithms of the proposed ciphers are described as well as they are implemented using the Cipher Block Chaining (CBC) mode of operation. Each iteration of the ciphers encrypts/decrypts 128 bits of plaintext/ciphertext, and the ciphers are iterated until the entire plaintext/ciphertext is encrypted/decrypted. The ciphers can also be described using other modes of operation, such as Cipher Feedback (CFB) mode, Output Feedback (OFB), and Counter (CTR) mode. Each mode of operation has its own advantages and disadvantages [67].

The non-linear transformation is nothing but a key-dependent S-box layer, which is carried out using the quasigroup operation. In each quasigroup operation, a key-dependent S-box layer chooses one S-box out of the 16 S-boxes of the quasigroup. The choice is based on the round key or sub-key. We believe that key-dependent S-box ciphers are more secure than fixed S-box ciphers. This is because key-dependent S-boxes do not offer any specific properties to the cryptanalyst. Most key-dependent S-box ciphers are effectively random. Examples of such ciphers are Blowfish [66] and SEAL [15].

[1] An efficient block cipher based on multiple optimal quasigroups and $\{e, d\}$-`transformation`
[2] A block cipher based on multiple optimal quasigroups and $\{ne^\ell, nd^\ell\}$-`transformation`

5.2.1 Quasigroup operation for encryption and decryption

A quasigroup operation is nothing but a substitution (S-box) operation. It substitutes a byte (8-bit) value with another byte (8-bit) value, depending on the round key or sub-key. Sixteen optimal quasigroups of order 16 are employed in the design of the proposed block ciphers, where each of these 16 quasigroups contains the same 16 S-boxes $S_0, S_1, \ldots S_{15}$ as an optimal quasigroup of order 16, but in a different order (permutation), one such quasigroup is shown in Table 2.8 of Chapter 2. Note that the size of each of these 16 S-boxes is 4×4 bits. So, all operations are performed in the form of 4-bit (also called nibbles) aggregations. Let each byte of data be divided into two nibbles. That is, a byte value x is represented as $x = x_1x_0$, where x_1 and x_0 are nibbles. Then, the quasigroup operations for encryption and decryption are defined as

$$\begin{gathered} x_1x_0 \star_i y_1y_0 = (x_1 *_i y_1)||(x_0 *_i y_0) \\ \text{or} \\ S_{x_1x_0}[y_1y_0] = S_{x_1}[y_1]||S_{x_0}[y_0] \end{gathered} \tag{5.1}$$

and

$$\begin{gathered} x_1x_0 \sharp_i y_1y_0 = (x_1 \setminus_i y_1)||(x_0 \setminus_i y_0) \\ \text{or} \\ S^{-1}_{x_1x_0}[y_1y_0] = S^{-1}_{x_1}[y_1]||S^{-1}_{x_0}[y_0] \end{gathered} \tag{5.2}$$

respectively, where $(*_i, \setminus_i)$ and $(\star_i, \sharp_i)$ are quasigroup operations corresponding to nibbles and bytes respectively, $0 \leq i, j \leq 15$, and $||$ is a concatenation operation that concatenates two 4-bit values into one 8-bit value. $S_j[x]$ denotes the output of j^{th} S-box, determined by looking up the row number j and the column number x of the quasigroup Q; similarly $S_j^{-1}[x]$ denotes the output of j^{th} inverse S-box, determined by looking up the row number j and the column number x of the left inverse quasigroup LIQ of the quasigroup Q, where S_j^{-1} is inverse of S_j. Also, the symbol $\sharp_i$ is called the inverse (or left inverse) quasigroup operation corresponding to the symbol $\star_i$. That is, if a quasigroup $Q_i = (\mathbb{Z}_{16}, \star_i)$ that consists of $S_1, S_2, \ldots, S_{15}$ is used for encryption then its left inverse quasigroup $LIQ_i = (\mathbb{Z}_{16}, \sharp_i)$ that consists of $S_1^{-1}, S_2^{-1}, \ldots, S_{15}^{-1}$ is used for decryption.

5.3 An efficient block cipher based on multiple optimal quasigroups and $\{e, d\}$-transformation

This section discusses the structure and the building elements of the first block cipher. The proposed cipher is an iterative cipher, and its design is based on the Substitution Permutation Network (SPN). It uses 16 optimal quasigroups of order 16 and a 128-bit secret key for encrypting/decrypting the messages. These 16 optimal quasigroups are constructed dynamically based on an original (fixed) optimal quasigroup $Q = (\mathbb{Z}_{16}, *)$ of order 16. Because of this, the cipher leverages the space of a single quasigroup and uses multiple quasigroups by generating them from an original quasigroup. That is, the space required by multiple optimal quasigroups is reduced to that of a single quasigroup. It performs a total of 16 rounds to encrypt or decrypt a block of 128 bits. Each round consists of two transformations (substitution and permutation), except the last/first round of the encryption/decryption. These transformations are an intermix of substitutions and permutations. The last/first round of the encryption/decryption only consists of the substitution.

The algorithm of the new cipher consists of three parts: (1) an algorithm to randomly select (or generate) an optimal quasigroup for each round, (2) an encryption algorithm, and (3) a decryption algorithm. The encryption algorithm consists of two transformations: (i) e-`transformation`, and (ii) bit permutation. The decryption algorithm also consists of two corresponding inverse transformations: (i) d-`transformation`, and (ii) inverse bit permutation. The e-`transformation` and the d-`transformation` are nothing but key-dependent S-box layers that depend on the sub-key. The workflow of both the encryption and the decryption algorithms of the proposed block cipher is shown in Figure 5.2. In this figure, $k_j, 0 \leq j \leq 15$, denotes a sub-key value of 8-bit (or an 8-bit of the secret key K), which is used in each round for (i) selecting a random optimal quasigroup Q_i for encryption and a left inverse quasigroup LIQ_i for decryption (where LIQ_i is the left inverse of Q_i), $0 \leq i \leq 15$, and (ii) e-transformation of the encryption and d-transformation of the decryption. The algorithm uses a **8bitTo4bitComp** compression function that compresses an 8-bit k_j to 4-bit. This 4-bit value is used by a 16×1 multiplexer for generating a random optimal quasigroup. The generation of the optimal quasigroups is discussed in the next section.

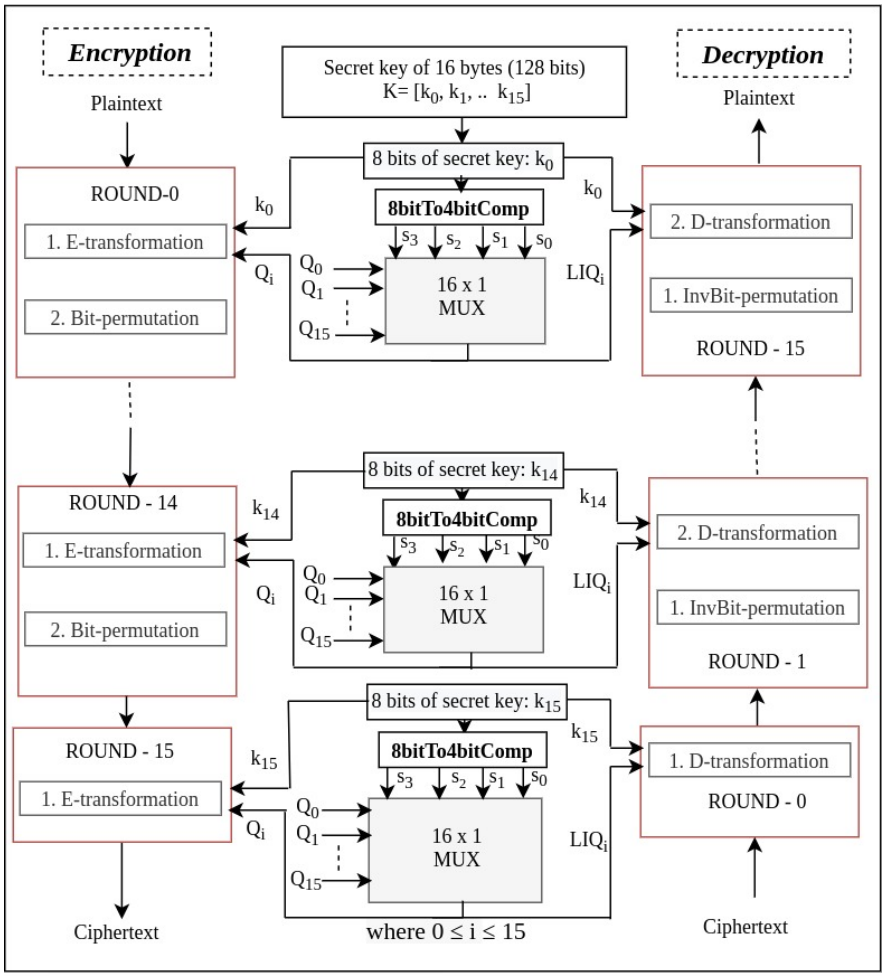


Figure 5.2: Encryption and decryption algorithms of the block cipher.

5.3.1 Generation of optimal quasigroups

The proposed cipher uses 16 optimal quasigroups of order 16 and performs a total of 16 rounds to encrypt or decrypt the messages. Also, each round of the proposed cipher uses only one optimal quasigroup out of 16 optimal quasigroups. This optimal quasigroup is selected randomly with equal probability. Note that an optimal quasigroup may be used in more than one round. Selection of the optimal quasigroup is carried out using a 16×1 multiplexer used in each round of the cipher, as shown in Figure 5.2. Note that these 16 optimal quasigroups are generated based on an original quasigroup $Q = (\mathbb{Z}_{16}, *)$. By permuting the rows of the original quasigroup Q, 16! quasigroups can be created. The proposed cipher uses only 16 quasigroups of 16! quasigroups. So,

any 16 out of the total 16! quasigroups can be selected. Let the selected quasigroups be $Q_0 = (\mathbb{Z}_{16}, *_0), Q_1 = (\mathbb{Z}_{16}, *_1), ..., Q_{15} = (\mathbb{Z}_{16}, *_{15})$, where $*_0, *_1, ..., *_{15}$ are the quasigroup operations corresponding to $Q_0, Q_1, ..., Q_{15}$, respectively. Note that all these 16 quasigroups need not be stored. This is because each quasigroup consists of the same rows but in a different order (permutation).

Based on 16×1 multiplexer, generation or selection of a total 16 quasigroups is done using Equation 4.3 of section 4.5.1 of Chapter 4, which is reproduced here in Equation 5.3 for easy reference:

$$Q_i = (R_{(0+\textbf{constval}) \, mod \, 16}, R_{(1+\textbf{constval}) \, mod \, 16}, \ldots, R_{(15+\textbf{constval}) \, mod \, 16}) \tag{5.3}$$

where $R_0, R_1, \ldots, R_{15}$ denote row numbers of the original quasigroup $Q = (\mathbb{Z}_{16}, *)$, $0 \leq \textbf{constval} \leq 15$, $Q_i \in \{Q_0, Q_1, \ldots, Q_{15}\}$. The value of **constval** is determined as follows: For selecting the r^{th} round quasigroup, we consider r^{th} round seed value or sub-key $k_r, 0 \leq r \leq 15$. This is a byte (8-bit) value of secret key K. So, divide k_r into two 4-bit values (nibbles), that is $k_r = k_{r_1} k_{r_0}$, where k_{r_0} and k_{r_1} are nibbles. Then,

$$\textbf{constval} = \textbf{8bitTo4bitComp}(k_r) = k_{r_1} \oplus k_{r_0} \tag{5.4}$$

where $\oplus$ is a bitwise addition modulo 2 operation. The `constval` is 4 bits and they are denoted by s_0, s_1, s_2, and s_3, where s_0 and s_3 are the least and the most significant bits, respectively. These s_i, $0 \leq i \leq 3$, are considered as the selection lines of the 16×1 multiplexer. If $s_3 = 0, s_2 = 0, s_1 = 0, s_0 = 0$, then the multiplexer selects or generates a quasigroup Q_0. If $s_3 = 0, s_2 = 0, s_1 = 0, s_0 = 1$, then the multiplexer selects or generates a quasigroup Q_1, and so on. Also, the same permutations of the rows $R_0, R_1, \ldots, R_{15}$ are used in the decryption algorithm, but these permutations are generated based on the $LIQ = (\mathbb{Z}_{16}, \backslash)$, where LIQ is the left inverse quasigroup of the original optimal quasigroup Q. The correctness of this is proven by Theorem 1 in Chapter 2.

5.3.2 Encryption

Encryption algorithm of the proposed block cipher is carried out in a total of 16 rounds. Each of these rounds, except the last round, consists of two transformations to encrypt a 128 bits block of data. The last (15^{th}) round performs only the E-transformation.

5.3.2.1 E-transformation

The E-transformation is nothing but e-`transformation`, defined in section 2.2.12 of Chapter 2. This constitutes a non-linear layer of the proposed cipher. It uses 16 S-boxes $S_0, S_1, \ldots, S_{15}$ in the form of an optimal quasigroup $Q = (\mathbb{Z}_{16}, *)$ of order 16. These 16 S-boxes are given in Table 2.8 of Chapter 2. It is a key-dependent S-box layer that substitutes a byte for a byte. It increases the confusion power of the cipher and hides the relationship between the ciphertext and the key; thereby making it difficult to find the key from the ciphertext. Let P=$\{p_0, p_1, \ldots, p_{15}\}$, k_r, and $C = \{c_0, c_1, \ldots, c_{15}\}$ denote input to a round, 8-bit seed value (sub-key) for the r^{th} round, and the output of a round, respectively. Then, the way of using the E-transformation on P with seed value k_r to obtain the corresponding C is as follows:

$$\left.\begin{aligned} c_0 &= k_r \star_i p_0, \\ c_j &= c_{j-1} \star_i p_j \end{aligned}\right\} \tag{5.5}$$

where p_0 p_j, c_j and k_r are byte values, $1 \leq j \leq 15$, $0 \leq r \leq 15$, and $\star_i$ is one of the quasigroup operations corresponding to the quasigroup $Q_i = (\mathbb{Z}_{16}, \star_i)$, defined in Equation 5.1, $0 \leq i \leq 15$.

5.3.2.2 Bit permutation

This is a linear transformation used after E-transformation to increase the diffusion power of the block cipher. It spreads the non-zero bits so as to increase the number of active S-boxes in the differential and linear trails; thereby getting the maximum impact of the substitution layer. It has the ability to hide the relationship between the ciphertext and the plaintext. The permutation of the bits used in the proposed cipher is given row-wise in Table 5.1. It maps bits from bit position x to bit position $\sigma(x)$, defined by the following equation:

$$\sigma(x) = \left(32\left(\left(3\left\lfloor\frac{x \; mod \; 16}{4}\right\rfloor + (x \; mod \; 4)\right) \; mod \; 4\right) + 4\left\lfloor\frac{x}{16}\right\rfloor + 1 + (x \; mod \; 4)\right) mod \; 128.$$

The following observations can be made by looking at the permutation layer:

(i) The four output bits of a particular round S-box enter into four different S-boxes of the next round.

(ii) The four input bits to an S-box of a particular round come from four different S-boxes of the previous round.

(iii) The y^{th} output bit of an S-box of a particular round becomes the $((y+1)\ mod\ 4)^{th}$ input bit of a different S-box of the next round, where $0 \leq y \leq 3$, 0^{th} and 3^{th} bits are the least and most significant bits, respectively, of the S-box.

(iv) The y^{th} input bit to an S-box of a particular round comes from the $((y-1)\ mod\ 4)^{th}$ output bit of a different S-box of the previous round.

According to observation (i); the four output bits of an S-box in one round will affect four S-boxes in the next round, and then 16 S-boxes together in the round after that. Therefore, it can be demonstrated that this permutation will affect all the 32 S-boxes in three rounds using similar reasoning for the subsequent rounds. That is, this bit permutation needs four rounds to achieve the full diffusion, that is, an input bit to an S-box of a particular round will influence all the 128 bits in four rounds, which is optimal [3].

Table 5.1: Bit permutation for a 128-bit block.

1	34	67	100	97	2	35	68	65	98	3	36	33	66	99	4
5	38	71	104	101	6	39	72	69	102	7	40	37	70	103	8
9	42	75	108	105	10	43	76	73	106	11	44	41	74	107	12
13	46	79	112	109	14	47	80	77	110	15	48	45	78	111	16
17	50	83	116	113	18	51	84	81	114	19	52	49	82	115	20
21	54	87	120	117	22	55	88	85	118	23	56	53	86	119	24
25	58	91	124	121	26	59	92	89	122	27	60	57	90	123	28
29	62	95	0	125	30	63	96	93	126	31	64	61	94	127	32

5.3.2.3 Encryption algorithm using CBC mode of operation

The encryption algorithm of the proposed block cipher is implemented using the Cipher Block Chaining (CBC) mode of operation. The pseudocode of the encryption algorithm is given in Algorithm 8. In this algorithm, Q_i denotes a generated optimal quasigroup based on the k_r for the r^{th} round of P_j block, $0 \leq i, r \leq 15$, $0 \leq j \leq N-1$, where N is the total number of plaintext blocks to be encrypted. The encryption algorithm

Algorithm 8: Pseudocode of the encryption algorithm

Input: 1. Plaintext in the form of 128-bit (16 bytes) blocks. Let $P_0, P_1, \ldots, P_{N-1}$ be N number of blocks to be encrypted.
2. Initial value (IV) of 128 bits (16 bytes).
3. Secret key K of 128 bits in the form of a sequence of 16 bytes such as $K = k_0, k_1, \ldots, k_{15}$, where k_r is a 8-bit value for $0 \leq r \leq 15$.
4. An optimal quasigroup of order 16.

Output: Ciphertext whose size is equal to the size of the plaintext.

```
[1] for j = 0 to N-1 do
[2]     P_j = XOR(P_j, IV);
[3]     for r = 0 to 14 do
[4]         Q_i = Generated optimal quasigroup based on k_r;
[5]         P_j = E-transformation(P_j, k_r, Q_i);
[6]         P_j = Bit-permutation(P_j);
[7]     Q_i = Generated optimal quasigroup based on k_15;
[8]     P_j = E-transformation(P_j, k_15, Q_i);
[9]     IV = P_j;
```

updates block P_j many times during the encryption process, and the results are stored in the same block P_j , $0 \leq j \leq N-1$.

5.3.3 Decryption

The decryption process is the reverse of the encryption process. It recovers the original plaintext from the ciphertext. It uses the same sequence of the round seed values but in reverse order. Let $K = k_0, k_1, \ldots, k_{15}$ be 16 bytes of seed values, and if k_r is used in the r^{th} round of the encryption algorithm, then k_t is used in the t^{th} round of the decryption algorithm, where $t = 15 - r, 0 \leq r \leq 15$. The decryption algorithm of the proposed block cipher is not the same as the encryption algorithm. Also, it uses the left inverses of the quasigroups that were used in the encryption algorithm. It also performs a total of 16 rounds to decrypt a 128-bit block of data. Each round, except the initial (0^{th}) round, consists of two transformations. In the initial round, only the D-transformation takes place.

5.3.3.1 D-transformation

The D-transformation is nothing but d-transformation, defined in section 2.2.12 of Chapter 2. It is the inverse of the E-transformation. It uses the inverse S-boxes $S_0^{-1}, S_0^{-1}, \ldots, S_{15}^{-1}$ as a left inverse quasigroup $LIQ = (\mathbb{Z}_{16}, \backslash)$ of the quasigroup $Q = (\mathbb{Z}_{16}, *)$ that was used in the E-transformation, where S_α^{-1} is the inverse of S_α, $0 \leq \alpha \leq 15$. A method to find the left inverse quasigroup LIQ of the quasigroup Q is given in Chapter 2. Let $C = \{c_0, c_1, \ldots, c_{15}\}$, k_t, and P=$\{p_0, p_1, \ldots, p_{15}\}$ denote round input, 8-bit seed value (sub-key) for the t^{th} round, and the round output, respectively. Then, the way of using the D-transformation on C with seed value k_t to recover P is as follows:

$$\left.\begin{aligned} p_0 &= k_t \sharp_i c_0, \\ p_j &= c_{j-1} \sharp_i c_j, \end{aligned}\right\} \tag{5.6}$$

where p_0 p_j, c_j and k_t are byte values for $1 \leq j \leq 15$, $0 \leq t \leq 15$, and $\sharp_i$ is one of the left inverse quasigroup operations corresponding to the left inverse quasigroup $LIQ_i = (\mathbb{Z}_{16}, \sharp_i)$, defined in Equation 5.2, $0 \leq i \leq 15$.

5.3.3.2 Inverse bit permutation

The inverse bit permutation of the decryption algorithm is the reverse of the bit permutation used in the encryption algorithm, shown in Table 5.2.

Table 5.2: Inverse bit permutation for a 128-bit block.

115	0	5	10	15	16	21	26	31	32	37	42	47	48	53	58
63	64	69	74	79	80	85	90	95	96	101	106	111	112	117	122
127	12	1	6	11	28	17	22	27	44	33	38	43	60	49	54
59	76	65	70	75	92	81	86	91	108	97	102	107	124	113	118
123	8	13	2	7	24	29	18	23	40	45	34	39	56	61	50
55	72	77	66	71	88	93	82	87	104	109	98	103	120	125	114
119	4	9	14	3	20	25	30	19	36	41	46	35	52	57	62
51	68	73	78	67	84	89	94	83	100	105	110	99	116	121	126

5.3.3.3 Decryption algorithm based on CBC mode

The decryption algorithm of the proposed block cipher is also implemented using the Cipher Block Chaining (CBC) mode of operation. Each iteration of the decryption

algorithm decrypts 128 bits of the ciphertext and is repeated until the entire ciphertext is decrypted. The pseudocode of the decryption algorithm is given in Algorithm 9. In this algorithm, LIQ_i denotes a generated left inverse quasigroup based on the seed

Algorithm 9: Pseudocode of the decryption algorithm

Input: 1. Ciphertext in the form of 128-bit (16 bytes) blocks. Let $C_0, C_1, \ldots, C_{N-1}$ be the N number of blocks to be decrypted.
2. Initial value (IV) of 128 bits (16 bytes).
3. Secret key K of 128 bits in the form of a sequence of 16 bytes such as $K = k_0, k_1, \ldots, k_{15}$, where k_t is a 8-bit value for $0 \leq t \leq 15$.
4. A left inverse quasigroup of order 16, this quasigroup is the left inverse of a quasigroup that was used in the encryption algorithm.

Output: Plaintext whose size is equal to the size of the ciphertext.

```
[1]  for j = 0 to N − 1 do
[2]      CopyOfC_j = C_j;
[3]      LIQ_i = Generated left inverse quasigroup based on k_0;
[4]      C_j = D-transformation (C_j, k_0, LIQ_i);
[5]      for t = 1 to 15 do
[6]          LIQ_i = Generated left inverse quasigroup based on k_t;
[7]          C_j = Inverse bit permutation (C_j);
[8]          C_j = D-transformation (C_j, k_t, LIQ_i);
[9]      C_j = XOR (C_j, IV);
[10]     IV = CopyOfC_j;
```

value for the t^{th} round of C_j block, $0 \leq i, t \leq 15$, $0 \leq j \leq N-1$, where N is the total number of ciphertext blocks to be decrypted. The `CopyofC`$_j$ is a temporary variable used to store the value of C_j before starting the decryption process. The decryption algorithm updates block C_j many times during the decryption process, and the results are stored in the same block C_j, $0 \leq j \leq N-1$.

5.4 A block cipher based on multiple optimal quasigroups and $\{ne^\ell, nd^\ell\}$-transformation

The notation employed in this section is given in Table 5.3. Here, we discuss another block cipher different from the one discussed in the previous section 5.3. It is an

Table 5.3: Some important notations.

Notation	Meaning
XOR or $\oplus$	:bitwise addition modulo 2 operation
IV	:an initial value of 128 bits
N	:a total number of blocks to be encrypted/decrypted
K_0	:initial (0^{th}) round key or secret key
RK_r	:r^{th} round key for encryption, $0 \leq r \leq 16$
RK_t	:t^{th} round key for decryption, where $t = 16 - r$
$RK_r : w_p$	:p^{th} word of the r^{th} round key for encryption, $0 \leq r \leq 16, 0 \leq p \leq 67$
$RK_t : w_p$	:p^{th} word of the t^{th} round key for decryption, where $t = 16 - r$
W	:a total of 68 words of the round key
r & t	:round number for encryption & decryption respectively, where $0 \leq r, t \leq 16$
B & C	:block number for encryption & decryption respectively, where $0 \leq B, C \leq N - 1$

iterative cipher, and its design is also based on the Substitution Permutation Network (SPN). It also uses a 128 bits secret key and 16 optimal quasigroups of order 16. These 16 optimal quasigroups are generated based on an original optimal quasigroup $Q = (\mathbb{Z}_{16}, *)$ of order 16. The description of the optimal quasigroup is specified in Chapter 2. Note that this cipher uses the same set of 16 S-boxes used by the previous cipher. These 16 S-boxes $S_0, S_1, \ldots, S_{15}$ are given in Table 2.8 of Chapter 2. It performs a total of 17 rounds to encrypt or decrypt a block of 128 bits. Each round consists of a sequence of transformations. These transformations are an intermix of substitutions and permutations. Each round, except the initial/last round of encryption/decryption of the proposed cipher, uses an optimal quasigroup by randomly picking one out of the 16 optimal quasigroups. This optimal quasigroup is selected based on the previous/next round key of encryption/decryption. The proposed cipher leverages the space of a single optimal quasigroup and employs 16 optimal quasigroups by generating them from a single optimal quasigroup. That is, the space required by 16 optimal quasigroups is reduced to that of a single optimal quasigroup. The Theorem 1 of Chapter 2 is useful in proving the correctness of the proposed cipher. The workflow of encryption and decryption of the proposed block cipher is shown in Figure 5.3. In the figure, $w_p, 0 \leq p \leq 67$, denotes a 32-bit word of the round key. The word representation of the round key is described in the next subsection. Every round uses a quasigroup Q_i

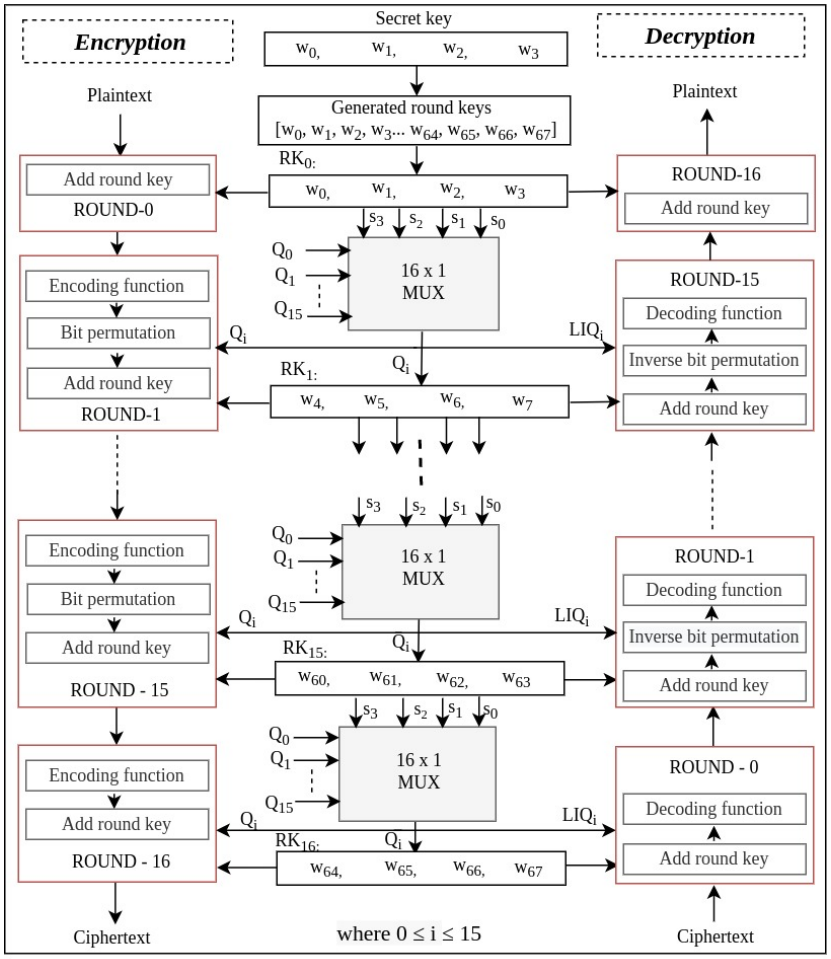


Figure 5.3: Workflow of encryption and decryption of new block cipher.

for encryption and an inverse quasigroup LIQ_i for decryption (where LIQ_i is the left inverse of Q_i), each of which is selected by a 16×1 multiplexer.

The algorithm of the proposed cipher consists of four parts: (1) an algorithm to randomly select an optimal quasigroup for each round of a block, (2) an algorithm to generate a round key, (3) an encryption algorithm, and (4) a decryption algorithm. The encryption algorithm employs three different transformations: (i) Encoding function, (ii) Bit permutation, and (iii) Add round key. Similarly, the decryption algorithm has three corresponding inverse transformations: (i) Decoding function, (ii) Inverse bit permutation, and (iii) Add round key.

5.4.1 Generation of round key

The proposed block cipher uses a round key generation algorithm to encrypt or decrypt a data block. It uses a 128-bit round key for each round to encrypt or decrypt a block of 128 bits. These round keys are generated based on the secret key of 128 bits along with 16 optimal quasigroups of order 16. The generation of each round key uses an optimal quasigroup out of 16 optimal quasigroups, selected by a 16×1 multiplexer. Now, the secret key of 128 bits is partitioned into four words, and these, in turn, are arranged as four columns of a matrix. Let $K_0 = (k_{(0,0)}, k_{(0,1)}, k_{(0,2)}, k_{(0,3)}, \ldots, k_{(3,0)}, k_{(3,1)}, k_{(3,2)}, k_{(3,3)})$ be a secret key of 128 bits (16 bytes), where each $k_{(i,j)}$ is a byte value for $0 \leq i, j \leq 3$, which are organized as a 4×4 matrix of bytes as shown in Figure 5.4 (a). In this matrix, p^{th} word (column) is denoted by w_p, where $0 \leq p \leq 3$ and size of each w_p is 32 bits. These four words are used to create the initial (0^{th}) round key, and it is represented as ($K_0 : w_0$, $K_0 : w_1$, $K_0 : w_2$, $K_0 : w_3$) or simply, $K_0 = (w_0, w_1, w_2, w_3)$. Our proposed cipher performs a total of 17 rounds for encrypting/decrypting a block of 128 bits, and each round consists of four words as a key. Therefore, a total of 68 words ($W = \{RK_0 : w_0, RK_0 : w_1, RK_0 : w_2, RK_0 : w_3, RK_1 : w_4, \ldots, RK_{15} : w_{63}, RK_{16} : w_{64}, RK_{16} : w_{65}, RK_{16} : w_{66}, RK_{16} : w_{67}\}$) are required as shown in Figure 5.4. These

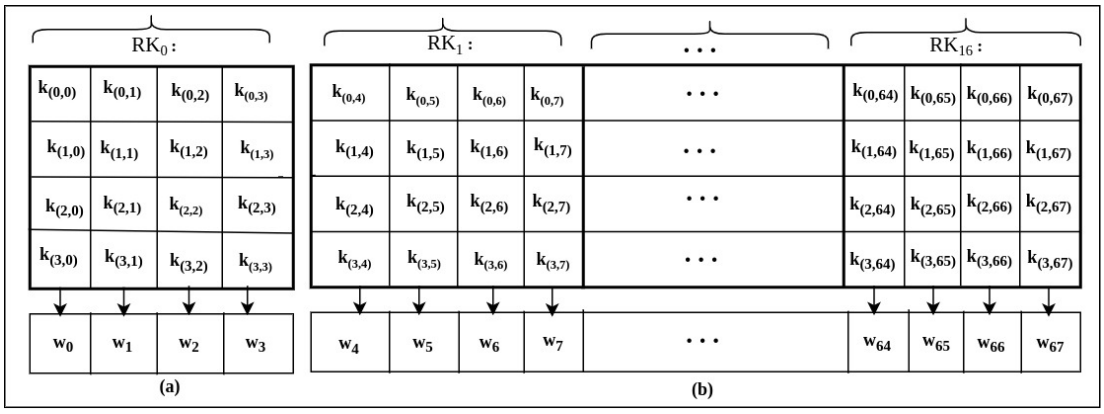


Figure 5.4: Representation of round keys.

68 words are generated based on K_0, including four words of the initial (0^{th}) round key K_0. For $1 \leq r \leq 16$, $4 \leq p \leq 67$, a p^{th} word of the r^{th} round key is generated based on the previous words $p-4$ and $p-3$. The generation of the round key algorithm is given in Algorithm 10. In this algorithm, $Q_i, 0 \leq i \leq 15$, is a generated quasigroup. And $\bigstar_i$

Algorithm 10: Generation of the round key

Input: 1. A 128-bit secret key in the form of a 4×4 matrix of bytes as shown in Figure 5.4(a).
2. An optimal quasigroup of order 16.

Output: Generates all the 17 rounds key in the form of a 4×68 matrix of bytes as shown in Figure 5.4.

```
[1]  if r = 0 and 0 ≤ p ≤ 3 then
[2]      RK_0 = K_0;
[3]  else
[4]      for p = 4 to 67 do
[5]          if p mod 4 = 0 then
[6]              Q_i = generated quasigroup based on r^th round key;
[7]              r = r + 1;
[8]          if RK_(r−1) : w_(p−3) ∈ W then
[9]              RK_r : w_p = RK_(r−1) : w_(p−4) ★_i RK_(r−1) : w_(p−3);
[10]         else
[11]             RK_r : w_p = RK_(r−1) : w_(p−4) ★_i RK_r : w_(p−3);
```

is the binary operation (quasigroup operation) of 32-bit words defined as follows: Let $U = (u_1, u_2, u_3, u_4)$ and $V = (v_1, v_2, v_3, v_4)$ be two words of 32 bits each, where u_j and v_j are byte values ($1 \leq j \leq 4$). Then,

$$U \bigstar_i V = Z = (u_1 \star_i v_1, u_2 \star_i v_2, u_3 \star_i v_3, u_4 \star_i v_4),$$

where $\star_i$ is one of the quasigroup operations defined in Equation 5.1, $0 \leq i \leq 15$. Note that a quasigroup operation $\bigstar_i$ is a look-up table operation. So, here the resultant value Z is determined by looking up the element having the row number U and the column number V in the table representation of the quasigroup $Q = (\mathbb{Z}_{2^{32}}, \bigstar_i)$.

Note that both the encryption and the generation of the round key algorithms use the same set of optimal quasigroups. These optimal quasigroups are generated based on an original optimal quasigroup $Q = (\mathbb{Z}_{16}, *)$. But in decryption, we use a left inverse quasigroup $LIQ = (\mathbb{Z}_{16}, \backslash)$ of the quasigroup Q. Note that both the encryption and the decryption algorithms use the same round key. Therefore in decryption, the key generation algorithm has to perform the quasigroup operations corresponding to the quasigroup $Q = (\mathbb{Z}_{16}, *)$ based on the quasigroup $LIQ = (\mathbb{Z}_{16}, \backslash)$. From Definition 2.2.6 of Chapter 2, the relation between these two quasigroup operations is as follows:

$$t_2 \backslash t_3 = t_1 \Leftrightarrow t_2 * t_1 = t_3, \forall (t_1, t_2, t_3) \in \mathbb{Z}_{16} \times \mathbb{Z}_{16} \times \mathbb{Z}_{16}.$$

5.4.1.1 Avalanche effect of expanded key

The round key generation (key expansion) algorithm uses a 4-word (where the size of each word is 4 bytes) secret key and produces a total of 64 words, excluding the secret key. The round key generation algorithm uses 16 optimal quasigroups of order 16. Out of these, it uses only one optimal quasigroup for generating a round key. The selection of an optimal quasigroup for each round depends on the previous round key except for the initial round. Uniformly using each of these 16 optimal quasigroups eliminates the possibility of producing the same expanded key for two different secret keys. It also allows us to obtain high diffusion of the secret key bits during the key expansion. The current round key is used to generate the next round key. This and the use of optimal quasigroups cause the key generation algorithm to produce distinct round keys. So, key bits in every round are unique. Therefore, slide attacks are avoided. We also believe that the possibility of the existence of weak or related keys is minimal.

Key expansion and key scheduling algorithms are designed so that the knowledge of a part of the secret key or round sub-key bits will not allow determining the other round's sub-key bits. We analyzed the avalanche effect on the expanded key and compared it with that of the AES-128 key expansion for three cases, namely, (i) all 0's, (ii) all 1's, and (iii) randomly generated secret keys. Results of this analysis are given in Table 5.4. These results are obtained by changing each bit of the secret key as the size of the secret key is 128 bits. Note that more than 47% bits of the expanded key have been changed in the case of the proposed key expansion algorithm, whereas the corresponding number in the case of AES-128 is only around 31%. That is, the results of the proposed cipher are more close to the ideal value than the results of AES-128. Based on these results, we can conclude that our key expansion algorithm has a higher diffusion than AES-128 for protection from some slide and key-related attacks.

Table 5.4: Avalanche effect of expanded key, when the secret key is with all zeros, all ones, and randomly generated.

Secret key of 128 bits	Proposed cipher	AES-128
$key_1 = 0$	$min = 42.19\%$, $max = 54.25\%$, $avg = 47.73\%$, $sd = 3.21$	$min = 16.09\%$, $max = 46.25\%$, $avg = 32.26\%$, $sd = 7.12$
$key_2 = 1$	$min = 42.43\%$, $max = 53.07\%$, $avg = 47.95\%$, $sd = 2.63$	$min = 17.42\%$, $max = 46.09\%$, $avg = 31.65\%$, $sd = 7.53$
$key_3 = rand$	$min = 44.38\%$, $max = 50.24\%$, $avg = 47.87\%$, $sd = 1.01$	$min = 16.48\%$, $max = 45.00\%$, $avg = 31.49\%$, $sd = 7.58$

5.4.2 Generation of multiple quasigroups

Both the schemes (round key generation and encryption/decryption algorithms) use 16 optimal quasigroups of order 16. These 16 quasigroups are generated by permuting the rows of an original quasigroup $Q = (\mathbb{Z}_{16}, *)$, example of it is given in Table 2.8 of Chapter 2. By permuting the rows of the optimal quasigroup Q, 16! optimal quasigroups can be created. Note that the proposed schemes use only 16 optimal quasigroups. So, we select any 16 out of the total 16! optimal quasigroups. Let the selected optimal quasigroups be denoted by $Q_0 = (\mathbb{Z}_{16}, *_0), Q_1 = (\mathbb{Z}_{16}, *_1), ..., Q_{15} = (\mathbb{Z}_{16}, *_{15})$, where

$*_0, *_1, ..., *_{15}$ are the quasigroup operations corresponding to $Q_0, Q_1, ..., Q_{15}$, respectively.

The selection of an optimal quasigroup out of 16 optimal quasigroups depends on both the round key and an optimal quasigroup of the previous round of the proposed encryption system, except the 1^{st} round. This is because the 1^{st} round uses the initial (0^{th}) round key and an original optimal quasigroup Q. Using a 16×1 multiplexer, the generation or selection of a total of 16 quasigroups is discussed earlier in section 5.3.1. So, using this process, the following equation can generate 16 optimal quasigroups based on an original quasigroup Q.

$$Q_i = (R_{(0+\mathbf{Rconstval}) \, mod \, 16}, R_{(1+\mathbf{Rconstval}) \, mod \, 16}, \ldots, R_{(15+\mathbf{Rconstval}) \, mod \, 16}) \tag{5.7}$$

where $R_0, R_1, \ldots, R_{15}$ denote row numbers of the original quasigroup $Q = (\mathbb{Z}_{16}, *)$, $0 \leq$ **Rconstval** ≤ 15, $Q_i \in \{Q_0, Q_1, \ldots, Q_{15}\}$, and the value of **Rconstval** is determined as follows: For selecting or generating $(r+1)^{th}$ round optimal quasigroup, we consider r^{th} (previous) round key shown in Figure 5.4. This is a 4×4 matrix of bytes. Now, define

$$\begin{aligned}
\texttt{Temp}_0 &= k_{(0,4r)} \oplus k_{(1,4r)} \oplus k_{(2,4r)} \oplus k_{(3,4r)}, \\
\texttt{Temp}_1 &= k_{(0,4r+1)} \oplus k_{(1,4r+1)} \oplus k_{(2,4r+1)} \oplus k_{(3,4r+1)}, \\
\texttt{Temp}_2 &= k_{(0,4r+2)} \oplus k_{(1,4r+2)} \oplus k_{(2,4r+2)} \oplus k_{(3,4r+2)}, \\
\texttt{Temp}_3 &= k_{(0,4r+3)} \oplus k_{(1,4r+3)} \oplus k_{(2,4r+3)} \oplus k_{(3,4r+3)}, \\
\texttt{XORofTemp}_j &= \texttt{Temp}_0 \oplus \texttt{Temp}_1 \oplus \texttt{Temp}_2 \oplus \texttt{Temp}_3.
\end{aligned}$$

Each $\texttt{Temp}_j$ for $0 \leq j \leq 3$ and $\texttt{XORofTemp}_j$ are byte values. Let $\texttt{XORofTemp}_j$ be divided into two 4-bit values (nibbles), that is $\texttt{XORofTemp}_j = x_1 x_0$, where x_0 and x_1 are nibbles. Then,

$$\texttt{Rconstval} = x_1 *_i x_0, \tag{5.8}$$

where $*_i$ is the quasigroup operation corresponding to the r^{th} round's optimal quasigroup, $0 \leq i, r \leq 15$. The $\texttt{Rconstval}$ is 4 bits, denoted by s_3, s_2, s_1, and s_0, where s_0 and s_3 are the least significant bit and the most significant bit, respectively. These s_α, $0 \leq \alpha \leq 3$, are considered as the selection lines of the 16×1 multiplexer. If

$s_3 = 0, s_2 = 0, s_1 = 0, s_0 = 0$, then the multiplexer selects or generates a quasigroup Q_0. If $s_3 = 0, s_2 = 0, s_1 = 0, s_0 = 1$, then the multiplexer selects or generates a quasigroup Q_1, and so on. Note that all these 16 optimal quasigroups need not be stored. This is because each optimal quasigroup consists of the same rows $R_0, R_1, \ldots, R_{15}$ but in a different order (permutation). Also, the same permutations of the rows $R_0, R_1, \ldots, R_{15}$ are used in the decryption algorithm, but these permutations are applied on the $LIQ = (\mathbb{Z}_{16}, \backslash)$, where LIQ is the left inverse quasigroup of the original optimal quasigroup Q. The correctness is proven in Theorem 1 in Chapter 2.

5.4.3 Encryption

Encryption is carried out in a total of 17 rounds. Each of these rounds, except the initial and the last rounds, comprises the following three transformations and encrypts a 128-bit block of data. Initial (0^{th}) round performs only the add round key, and the last (16^{th}) round performs the encoding function and add round key.

5.4.3.1 Encoding function

The encoding function is nothing but an ne^l`-transformation`, defined in section 2.2.15 of Chapter 2. It is a non-linear transformation of the proposed cipher. It uses 16 S-boxes $S_0, S_1, \ldots, S_{15}$ as an optimal quasigroup of order 16. These 16 S-boxes are given in Table 2.8 of Chapter 2. It is a key-dependent S-box layer that substitutes a byte for a byte. It adds to the confusion property and hides the relationship between the key and the ciphertext; thereby making it difficult to find the key from the ciphertext. Let B=$\{p_0, p_1, \ldots, p_{15}\}$, K=$\{k_0, k_1, \ldots, k_{15}\}$, and $C = \{c_0, c_1, \ldots, c_{15}\}$ denote input to a round, round key, and the output of a round, respectively. Then the way of using the encoding function on B with round key K to obtain the corresponding C is as follows:

$$c_j = k_j \star_i p_j$$

where all p_j, c_j and k_j are byte values for $0 \leq j \leq 15$ and $\star_i$ is one of the quasigroup operations, defined in Equation 5.1, for $0 \leq i \leq 15$.

5.4.3.2 Bit permutation

This is a linear transformation used to increase the diffusion power of the cipher. This is the second transformation in a round function. This cipher uses the same bit permutation as that of the encryption algorithm of the previous cipher, described in section 5.3.2.2.

5.4.3.3 Add round key

This is also a linear transformation. It is the third and the last transformation in a round function. This transforms a round input B=$\{p_0, p_1, \ldots, p_{15}\}$ of 16 bytes to the corresponding round output $C = \{c_0, c_1, \ldots, c_{15}\}$ of 16 bytes by XORing the input B with a round key K=$\{k_0, k_1, \ldots, k_{15}\}$ of 16 bytes as follows:

$$c_j = k_j \oplus p_j$$

where all p_j, c_j and k_j are byte values for $0 \leq j \leq 15$.

5.4.3.4 Encryption algorithm based on CBC mode of operation

The algorithm of the proposed cipher is implemented using the Cipher Block Chaining (CBC) mode of operation. Each iteration of the proposed cipher encrypts/decrypts 128 bits of plaintext/ciphertext, and it is repeated until the entire plaintext/ciphertext is encrypted/decrypted. The encryption algorithm of the proposed block cipher is given in Algorithm 11. In this algorithm, Q_i denotes a generated quasigroup based on the $(r-1)^{th}$ round key for the r^{th} round of B_j block, $0 \leq i \leq 15$, $0 \leq r \leq 16$, $0 \leq j \leq N-1$, where N is the total number of plaintext blocks to be encrypted. The encryption algorithm updates block B_j many times during the encryption, and the results are stored in the same block B_j.

5.4.4 Decryption

The decryption process is the reverse of encryption. It obtains the corresponding plaintext from the ciphertext. The algorithm of decryption is not the same as encryption. It uses the same sequence of round keys but in reverse order. Also, it uses the inverses of the quasigroups that were used in the encryption. It also performs 17 rounds to decrypt a 128-bit block of data. Each round, except the initial and the last rounds,

Algorithm 11: Pseudocode of the encryption algorithm

Input: 1. Plaintext in the form of 128-bit (16 bytes) blocks. Let $B_0, B_1, \ldots, B_{(N-1)}$ be the N number of blocks to be encrypted.
2. Initial value (IV) of 128 bits (16 bytes).
3. Secret key of 128 bits (16 bytes).
4. An optimal quasigroup of order 16.

Output: Ciphertext whose size is equal to the size of the plaintext.

[1] $\{RK_r{:}0 \leq r \leq 16\}$=Generate all the 17 rounds key;
[2] **for** $j = 0$ *to* $N - 1$ **do**
[3] B_j =XOR(B_j, IV);
[4] B_j = Add round key(B_j, RK_0);
[5] **for** $r = 1$ *to* 15 **do**
[6] Q_i = Generated quasigroup based on RK_{r-1};
[7] B_j = Encoding function(B_j, RK_r, Q_i);
[8] B_j = Bit permutation(B_j);
[9] B_j = Add round key(B_j, RK_r);
[10] Q_i = Generated quasigroup based on RK_{15};
[11] B_j = Encoding function(B_j, RK_{16}, Q_i);
[12] B_j = Add round key(B_j, RK_{16});
[13] $IV = B_j$;

consists of the following three transformations. In the initial round, add round key and decoding function; in the last round, only the add round key takes place.

5.4.4.1 Add round key

The add round key transformation for decryption is the same as that of the encryption process described in section 5.4.3.3.

5.4.4.2 Inverse bit permutation

This cipher uses the same inverse bit permutation as that of the decryption algorithm of the previous cipher, described in section 5.3.3.2.

5.4.4.3 Decoding function

The decoding function is nothing but an nd^{l}-transformation, defined in section 2.2.15 of Chapter 2. It is the inverse of the encoding function. It uses the inverse S-boxes $S_0^{-1}, S_0^{-1}, \ldots, S_{15}^{-1}$ as a left inverse quasigroup $LIQ = (\mathbb{Z}_{16}, \backslash)$ of the quasigroup $Q = (\mathbb{Z}_{16}, *)$ that was used in the encoding function, where S_{α}^{-1} is the inverse of S_{α}, $0 \leq \alpha \leq 15$. Let $C = \{c_0, c_1, \ldots, c_{15}\}$, K=$\{k_0, k_1, \ldots, k_{15}\}$, and B=$\{p_0, p_1, \ldots, p_{15}\}$ denote round input, round key, and the round output, respectively. Then the decoding function on C with the round key K to recover B is as follows:

$$p_j = k_j \sharp_i c_j$$

where all p_j, c_j and k_j are byte values for $0 \leq j \leq 15$ and $\sharp_i$ is one of the left inverse quasigroup operations, defined in Equation 5.2, for $0 \leq i \leq 15$.

5.4.4.4 Decryption algorithm based on CBC mode

The decryption algorithm of the proposed cipher is also implemented using the Cipher Block Chaining (CBC) mode of operation. Each iteration of the decryption algorithm decrypts 128 bits of the ciphertext and is repeated until the whole ciphertext is decrypted. The algorithm of decryption algorithm is given in Algorithm 12. In this algorithm, LIQ_i, $0 \leq i \leq 15$, is a generated left inverse quasigroup based on the $(t+1)^{th}$ (next) round key for the t^{th} round of C_j block, $0 \leq t \leq 16$, $0 \leq j \leq N-1$, where N is the total number of ciphertext blocks to be decrypted. The CopyofC_j is a

Algorithm 12: Pseudocode of the decryption algorithm

Input: 1. Ciphertext in the form of 128-bit (16 bytes) blocks. Let $C_0, C_1, \ldots,$ $C_{(N-1)}$ be the N number of blocks to be decrypted.
2. Initial value (IV) of 128 bits (16 bytes).
3. Secret key of 128 bits (16 bytes).
4. A left inverse quasigroup of order 16, this quasigroup is the left inverse of a quasigroup that was used in the encryption algorithm.

Output: Plaintext whose size is equal to the size of the ciphertext.

[1] $\{RK_t{:}0 \leq t \leq 16\}$=Generate all the 17 rounds key;
[2] **for** $j = 0$ *to* $N - 1$ **do**
[3] `CopyOf`C_j= C_j;
[4] LIQ_i = Generated quasigroup based on RK_1;
[5] C_j = Add round key (C_j, RK_0);
[6] C_j = Decoding function (C_j, RK_0, LIQ_i);
[7] **for** $t = 1$ *to* 15 **do**
[8] LIQ_i = Generated quasigroup based on RK_{t+1};
[9] C_j = Add round key (C_j, RK_t);
[10] C_j = Inverse bit permutation (C_j);
[11] C_j =Decoding function (C_j, RK_t, LIQ_i);
[12] C_j = Add round key (C_j, RK_{16});
[13] C_j = `XOR` (C_j, IV);
[14] IV = `CopyOf`C_j;

temporary variable used to store the value of C_j before starting the decryption process. The decryption algorithm updates block C_j many times during the decryption, and the results are stored in the same block C_j, $0 \leq j \leq N-1$.

5.5 Performance analysis

In this section, we analyzed the performance of the proposed block ciphers in terms of time (speed) and space complexities. Also, their performances are compared with AES-128, DES, and the existing quasigroup-based block ciphers presented in the literature [5, 6, 83].

The proposed ciphers have been implemented in C++ on a system with the following configuration: Intel(R) Core(TM) i5-2400 CPU @3.40 GHz processor with 8 GB RAM and 64-bit Linux operating system. The source code of the proposed ciphers is run 10^3 times for different samples, and we calculated the average execution time in seconds. We have used the C++ standard <chrono> library to measure the execution time [38]. Note that the space complexity is determined based on the S-boxes or quasigroups required for all ciphers. The performance of the proposed cipher is compared with that of the existing quasigroup-based block ciphers [5, 6, 83], DES and AES-128. The results of this analysis are shown in Table 5.5. According to the results, as shown in Table 5.5, it can be observed that the proposed ciphers are faster than the existing ciphers, except for AES-128. However, the proposed ciphers use only 50% of the space compared to that of the AES-128.

5.6 Security analysis

The proposed ciphers can be seen as a family of encryption systems parameterized by an optimal quasigroup of order 16. Since our system can create a maximum of 16! optimal quasigroups of order 16; it follows that there are $C_1^{16!}$ ways to select an optimal quasigroup out of 16! optimal quasigroups. That is, the family consists of $C_1^{16!}$ cryptosystems. The sender and the receiver agree on a cryptosystem by first deciding on an optimal quasigroup. Once a cryptosystem is decided, the set of 16 optimal quasigroups of order 16 it uses is fixed, and each round uses only one of these 16 optimal quasigroups with equal probability, depending on the round key or sub-key

Table 5.5: Comparison of the time and space complexities.

Block ciphers	Time complexity			Space complexity
	Execution time in seconds			Space in bytes
	1 MB	2.11 MB	4.21 MB	
Proposed block ciphers				
1^{st} cipher discussed in section 5.3	0.96	1.93	3.87	128
2^{nd} cipher discussed in section 5.4	1.23	2.47	4.95	128
Existing block ciphers				
Battey et al. [5, 6]	2.51	4.61	10.57	65536
Zhao and Xu [83]	3.32	6.71	13.47	128
DES	10.42	20.79	39.82	180
AES-128	0.67	1.21	2.32	256

value. To access these 16 quasigroups, a cryptanalyst must first determine the secret key to be used.

- ***Exhaustive key search attack***:- The proposed ciphers use a secret key of 128 bits. Therefore, the number of the possible keys is $2^{128} \approx 3.4 \times 10^{38}$. So, the running time of this attack is $T = O(u) = O(2^{128})$, where u is the size of the key space. Note that it is exponential in the size of the secret key. Let us assume a cryptanalyst uses a supercomputer and tries 5.37×10^{17} keys per second, then the cryptanalyst needs around 2.01×10^{13} years to determine the employed key. This is because these days, supercomputers can perform 5.37×10^{17} FLOPS[1] [76].

5.6.1 Linear cryptanalysis

Linear cryptanalysis is one of the powerful attacks against block ciphers. This attack model works based on the known-plaintext attack. It creates linear approximations of the plaintext bits, ciphertext bits, and sub-keys that hold with a suitably high probability. Let P_l be the probability of a linear approximation, then its probability bias (denoted by ϵ) can be defined as $|P_l - \frac{1}{2}|$. The higher the magnitude of the probability bias ϵ, the fewer known plaintexts are required to mount a linear attack. The encryption

[1] floating point operations per second

systems of the proposed ciphers are slightly different from each other. This is because, each round of the first cipher uses only two transformations ((i) E-transformation, and (ii) bit-permutation); whereas the second cipher uses three transformations ((i) encoding function, (ii) bit permutation, and (iii) add round key). The E-transformation and encoding function are key-dependent substitution (S-box) layers that use the round key value. The first cipher discussed in section 5.3 consists of 16 rounds, and each round uses a substitution (or an E-transformation) layer. The second cipher discussed in section 5.4 consists of 17 rounds, in which only the 16 rounds (from the 1^{st} round to the 16^{th} round of the encryption algorithm) use substitution (or encoding functions) layers. In both ciphers, a substitution layer is nothing but an S-box layer that uses 16 optimal S-boxes as an optimal quasigroup of order 16, where the size of each S-boxes is 4×4 bits.

We investigate the linear probability bias ϵ of $r+1$ round cipher by constructing the linear approximations of $r+1$ rounds, $r \geq 0$. This is because once a $r+1$-round linear approximation of the $r+2$ rounds cipher is discovered with a suitably high linear probability bias ϵ, then it is conceivable to attack the cipher [32]. Let u bits of the plaintext or input to the 0^{th} round (denoted by I_0), v bits of the output of the r^{th} round (denoted by O_r), and a total of w bits of the round keys used from the 0^{th} round to the r^{th} round (denoted by $K_{(0,r)}$). Then an approximation of r rounds is defined as follows

$$\left(\bigoplus_{i=0}^{u} I_0^{x_i}\right) \oplus \left(\bigoplus_{j=0}^{v} O_r^{x_j}\right) = \bigoplus_{k=0}^{w} K_{(0,r)}^{x_k} \tag{5.9}$$

where x_i, x_j and x_k denote bit positions, and $\oplus$ is a bitwise addition modulo 2 operation. For the first cipher, the right side value of Equation 5.9 would be zero since it does not consist of an add-round-key transformation. For the second cipher, the right side value of Equation 5.9 would be either 0 or 1, depending on the round key bits involved in the add-round-key transformation. These bits are fixed but unknown (as they are determined by the key under attack). This kind of linear relation is obtained by concatenating the appropriate linear approximations of S-boxes from round to round of the cipher. This is because S-boxes are the only non-linear components of the proposed cipher. These linear approximations hold a relation between the input and output bits of the S-boxes with a certain probability. The linear approximation of

the $r+1$ rounds cipher represents a linear trail (also called a path from the 0^{th} round up to the r^{th} round of the encryption system) that consists of active S-boxes. A linear trail is optimal if it contains a minimum number of active S-boxes.

We used the Linear Approximation Tables (LATs) of the active S-boxes to form the linear trails. A LAT shows the probability bias values of all the possible linear approximations of an S-box. Note that the LATs of all the 16 S-boxes have the same magnitude linear probability bias values ($|P_l - \frac{1}{2}|$). Because of this property, the key-dependent S-box (non-linear) layer does not need to differentiate among LATs of all the 16 S-boxes while arriving at the optimal linear trail. A LAT of one of the S-boxes is given in Table 5.7. In this table, α denotes the input mask (row number), and β denotes the output mask (column number) of the linear approximation in hexadecimal. Dividing each table element by 16 gives the probability bias ϵ for that particular linear approximation. That is, for all possible input values and hence the output values of the S-box, a linear approximation is represented as $a_0 \cdot y_0 \oplus a_1 \cdot y_1 \oplus a_2 \cdot y_2 \oplus a_3 \cdot y_3 = b_0 \cdot z_0 \oplus b_1 \cdot z_1 \oplus b_2 \cdot z_2 \oplus b_3 \cdot z_3$, where "$\cdot$" denotes bitwise AND operation, both $a_i, b_i \in \{0, 1\}$, $a_0a_1a_2a_3$ is the binary representation of α and $b_0b_1b_2b_3$ is the binary representation of β where (a_0, b_0) and (a_3, b_3) are the least and the most significant bits, respectively. For example $\alpha = A$ and $\beta = 2$, a linear approximation of the S-box with probability bias $\frac{1}{4}$ is $y_1 \oplus y_3 = z_1$. More details about LAT are given in [32].

We investigate the construction of an $r+1$-round linear trail by examining non-zero input masks corresponding to the S-boxes from the 0^{th} round to the r^{th} round, $r \geq 0$. If a particular non-zero input mask (α) occurs, then the corresponding output mask (β) with suitably high probability bias and the least hamming weight is decided using the LATs of the S-boxes in each round. As we all know, a linear trail consists of a sequence of input and output masks between the rounds so that the output masks from one round correspond to the input masks of the next round. We only consider the linear approximations of the S-boxes that have non-zero input masks and hence non-zero output masks to estimate the number of active S-boxes of a linear trail. Since the size of the input block of the proposed ciphers is 128 bits, it is impossible to find all of the possible linear trails for 2^{128} inputs. So, we divide 128 bits input block into 4 sub-blocks as P_1, P_2, P_3, and P_4, where the size of each sub-block is 32 bits. And then, we use all the possible non-zero inputs of the first sub-block P_1, keeping all the remaining sub-blocks P_2, P_3, and P_4 as zero values. This gives rise to a maximum of

2^{32} linear trails. Similarly, we repeated the same procedure for the sub-blocks P_2, P_3, and P_4. Note that both ciphers use the same 16 S-boxes and a bit permutation to achieve the confusion and diffusion properties, respectively. So, the substitution and permutation layers of both ciphers do not differentiate the number of active S-boxes in the optimal linear trail. In order to investigate the optimal linear trails corresponding to both the ciphers discussed in sections 5.3 and 5.4, we used a computer-based search to find an optimal linear trail of the $r+1$-round cipher ($r \geq 0$) by evaluating the number of active S-boxes at each round of the proposed block ciphers. The number of active S-boxes in the optimal linear trail of the $r+1$-round cipher corresponding to both ciphers is given in Table 5.6 (a) and (b). In this table, $\#r$ denotes the round number, and #S-box denotes the minimum number of active S-boxes in the optimal linear trail of the $r+1$-round cipher.

Table 5.6: Minimum number of active S-boxes in the linear trail of the $r+1$-round cipher.

1^{st} cipher, discussed in section 5.3				2^{nd} cipher, discussed in section 5.4			
$\#r$	# S-box	$\#r$	# S-box	$\#r$	# S-box	$\#r$	# S-box
0	1	8	45	1	1	9	45
1	2	9	68	2	2	10	68
2	3	10	89	3	3	11	89
3	5	11	113	4	5	12	113
4	7	12	134	5	7	13	134
5	11	13	159	6	11	14	159
6	17	14	190	7	17	15	190
7	28	15	211	8	28	16	211
(a)				(b)			

The maximum probability bias of all the 16 S-boxes employed in the proposed block ciphers is 2^{-2}. So, using the piling-up lemma, the probability bias (ϵ) of the r-round cipher can be determined as follows [52]:

$$\epsilon = 2^{\#\text{S-box}-1} \times (\texttt{MPB})^{\#\text{S-box}}$$

where MPB denotes the maximum probability bias of the S-boxes used.

The complexity of the $r+1$-round cipher against linear attack can be obtained by

using the following formula [32]:

$$N_l \approx \frac{1}{\epsilon^2},$$

where N_l denotes the number of known plaintexts needed to mount the linear attack.

Also, the complexity of the $r+2$ rounds cipher against linear cryptanalysis depends on the linear probability bias value of $r+1$ rounds [32], $r \geq 0$. According to the results as shown in Table 5.6 (a)/(b), for $\#r = 14/15$ and $\#\text{S-box} = 190$, we have $\epsilon = 2^{-191}$. So, mounting the linear attack of a 16/17-round cipher required 2^{382} known plaintexts (N_l). Hence, the proposed ciphers are resistant to linear attacks.

Table 5.7: Linear Approximation Table (LAT)

$\alpha \setminus \beta$	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**A**	**B**	**C**	**D**	**E**	**F**
0	8	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	-2	2	4	0	2	2	0	-4	2	-2	0	0	2	2	0
2	0	0	2	-2	2	2	0	4	-2	-2	0	4	0	0	-2	2
3	0	2	4	-2	2	0	2	0	2	4	-2	0	0	-2	0	-2
4	0	-2	0	2	0	2	-4	2	0	2	0	-2	0	-2	-4	-2
5	0	0	-2	-2	0	0	2	2	-4	0	2	-2	0	-4	2	-2
6	0	-2	2	0	-2	0	0	2	2	-4	-4	-2	0	-2	2	0
7	0	4	0	0	-2	2	-2	-2	-2	-2	-2	2	0	0	0	-4
8	0	0	-2	-2	0	0	2	2	0	0	-2	-2	-4	4	-2	-2
9	0	-2	0	2	4	-2	0	-2	0	-2	0	2	-4	-2	0	-2
A	0	0	4	0	-2	-2	2	-2	-2	-2	2	-2	0	0	-4	0
B	0	2	-2	0	2	0	0	-2	-2	0	-4	-2	0	-2	-2	4
C	0	2	2	0	4	2	-2	0	0	-2	2	-4	0	2	2	0
D	0	4	0	4	0	-4	0	4	0	0	0	0	0	0	0	0
E	0	2	0	2	-2	4	2	0	2	0	2	0	-4	-2	0	2
F	0	0	-2	2	2	2	4	0	2	-2	0	0	4	0	-2	-2

5.6.2 Differential cryptanalysis

Differential cryptanalysis is also one of the powerful attacks against block ciphers. This attack model works based on the chosen-plaintext attack. It reduces the complexity of the exhaustive key search attack. Like linear cryptanalysis, here we constructed the

differential trail of $r+1$-round cipher, $r \geq 0$. Let P' and P'' be the inputs (plaintexts) to the system and the corresponding outputs (outputs of the r^{th} round of the proposed ciphers) be R' and R'', respectively. The input difference is denoted by $\Delta P = P' \oplus P''$, and the output difference is denoted by $\Delta R = R' \oplus R''$, where $\oplus$ is a bit-wise addition modulo 2 operation. The pair $(\Delta P, \Delta R)$ is referred to as an expected differential trail (or differential characteristic) of an $r+1$-round cipher if a particular output difference ΔR occurs given a particular input difference ΔP with a suitably high probability. Since the proposed ciphers are of 128 bits, processing all the possible 2^{128} differential trails is practically impossible. This expected differential characteristic can be arrived at by concatenating the appropriate differential characteristics of the S-boxes from round to round. This is because S-boxes are the only non-linear components of the proposed block ciphers, and a differential trail consists of a sequence of input and output differences between the rounds; it follows that the output difference from one round corresponds to the input difference of the next round. Before combining the S-boxes to derive differential trails, we must discuss the influence of the add-round key layer on the differential trail (or on the S-box differential), which is used in the second cipher discussed in section 5.4. Consider the inputs P' and P'', and let k_r be a round key used with both the inputs P' and P'' in the add-round key layer. Then, the corresponding outputs would be $P' \oplus k_r$ and $P'' \oplus k_r$, respectively. Now, we have input difference as $(P' \oplus k_r) \oplus (P'' \oplus k_r) = P' \oplus P''$. Hence, the add-round key layer does not influence the input difference value and can be ignored to evaluate the number of active S-boxes in the optimal linear trail.

We used the Difference Distribution Table (DDT) of the active S-boxes to find the differential probabilities of the differential trails. A differential trail is said to be optimal if it contains a minimum number of active S-boxes. A DDT of the S-box shows the differential probability for all the possible pairs of the input and output differences. Note that the DDTs of all the 16 S-boxes of the proposed ciphers have the same differential probability for all the possible input and output differences pairs. Because of this property, the key-dependent non-linear (S-box) layer does not need to differentiate among DDTs of all the 16 S-boxes while arriving at the optimal differential trail. A DDT of one of the S-boxes is given in Table 2.10 of Chapter 2.

For both the ciphers discussed in sections 5.3 and 5.4, we investigate the construction of the differential trails by examining non-zero input differences corresponding to

the S-boxes from the 0^{th} round to the r^{th} round, $r \geq 0$. If a particular non-zero input difference occurs, then the corresponding output difference with suitably high probability and the least hamming weight is decided in each round using the DDTs of the S-boxes. So, we only consider the S-boxes that have non-zero input differences and hence non-zero output differences to estimate the number of active S-boxes of a differential trail. Since the size of the input block of both the proposed cipher is 128 bits, we divide 128-bit input (input difference) block ΔP into 4 sub-blocks as $\Delta P_1, \Delta P_2, \Delta P_3$, and ΔP_4, where the size of each sub-block is 32 bits. And then, we find all the differential trails for all the possible non-zero values for the first sub-block ΔP_1, keeping the remaining three sub-blocks fixed. This gives rise to a maximum of 2^{32} differential trails. Similarly, we repeated the same procedure for the sub-blocks ΔP_2, ΔP_3, and ΔP_4. Note that both the ciphers use the same 16 S-boxes and a bit permutation to achieve the confusion and diffusion properties, respectively. Therefore, the substitution and permutation layers do not differentiate the ciphers while determining the number of active S-boxes in the optimal differential trail. We used a computer-based search to find an optimal differential trail of $r+1$-round cipher ($r \geq 0$) by evaluating the number of active S-boxes at each round of the proposed ciphers. The minimum number of active S-boxes in the optimal differential trail of $r+1$-round cipher corresponding to the first and the second ciphers are given in Table 5.8 (a) and 5.8 (b), respectively. In this table, $\#r$ denotes the round number, and #S-box denotes the minimum number of active S-boxes in the optimal differential trail of the $r+1$-round cipher.

The maximum differential probability of all the 16 S-boxes employed in the proposed block ciphers is 2^{-2}. The attack complexity of the $r+1$-round cipher against differential cryptanalysis is approximately the inverse proportion to its largest differential probability and can be determined using the following formula [32]:

$$N_d \approx \frac{c}{P_d},$$

where N_d denotes the number of chosen plaintexts required to mount the differential cryptanalysis attack, c is a small constant, and P_d denotes the differential probability of $r+1$-round cipher, determined as

$$P_d = \prod_{i=1}^{\#S-box} \mathcal{P}_i,$$

Table 5.8: Minimum number of active S-boxes in the differential trail of the $r+1$-round cipher.

1^{st} cipher, discussed in section 5.3				2^{nd} cipher, discussed in section 5.4			
$\#r$	# S-box	$\#r$	# S-box	$\#r$	# S-box	$\#r$	# S-box
0	1	8	23	1	1	9	23
1	2	9	31	2	2	10	31
2	3	10	43	3	3	11	43
3	4	11	62	4	4	12	62
4	6	12	88	5	6	13	88
5	9	13	115	6	9	14	115
6	13	14	142	7	13	15	142
7	16	15	166	8	16	16	166
(a)				(b)			

where $\mathcal{P}_i$ denotes the differential probability of the i^{th} active S-box in the differential trail of the $r+1$-round cipher, $r \geq 0$.

Like linear cryptanalysis, the complexity of the $r+2$ rounds cipher against differential cryptanalysis depends on the differential probability of $r+1$ rounds [32], $r \geq 0$. According to the results shown in Table 5.8 (a)/(b), for $\#r = 14/15$ and $\#\text{S-box} = 142$, we have $P_d = (2^{-2})^{142} = 2^{-284}$. And if $c = 1$, then $N_d = 2^{284}$. So, mounting the differential attack of a 16/17-round cipher required 2^{284} chosen plaintexts. Hence, the proposed ciphers are resistant to differential attacks.

5.6.3 Avalanche effect

Avalanche effect is one of the desirable properties of the block ciphers, wherein for a small change in the input (plaintext), there should be a large change in the corresponding output (ciphertext). A good avalanche effect ensures that the diffusion power of a block cipher is at least 50%. We looked into the avalanche effects of both the proposed ciphers using the inputs of low and high hamming weights.

Let P be an input block of 128 bits with a low hamming weight that consists of all binary $0's$ (0X00). We created 128 inputs P_j that differ in 1 bit from the original input P. That is

$$P_j = P \oplus (1 << _j),$$

where $\oplus$ is a bitwise addition modulo 2, $<<_j$ is the left shift operation by j bit positions and $0 \leq j \leq 127$.

Now let C be the output of the original input P and C_j be the output of the input P_j for $0 \leq j \leq 127$. We calculated the hamming distances between C and C_j in percentages as

$$hdp_j = \frac{hd(C, C_j)}{\text{length}(C)} \times 100\%,$$

where $0 \leq j \leq 127$, $hd(C, C_j)$ denotes the hamming distance between C and C_j, and length(C) denotes the number of binary digits in the output (ciphertext) C. We repeated the same process for another 128-bit input (plaintext) with a high hamming weight that consists of all binary $1's$ (0XFF), and we calculated all the corresponding values of hdp_j for $0 \leq j \leq 127$. For both the inputs, we compared the hdp_j values of the proposed ciphers with those of the existing quasigroup based block ciphers given in [5, 6, 83] and AES-128. The results of this analysis corresponding to different ciphers are given in Table 5.9. The table shows the number of times the hamming distances (hdp_i) of the outputs $C_0, C_1, \ldots, C_{127}$ from C lie in the specified range. For example, for the input $0XFF$, 60 time the values of hdp_i of AES-128 lie in the range of $35-49.99$; while for the input $0X00$, 55 time the values of hdp_i of AES-128 lie in the range of $35 - 49.99$. The average (mean) hamming distance in percentage and the median absolute deviation (MAD) are also given in the last two columns of the table. The MAD tells us how far the hamming distances from the mean are. From these values, it can be observed that the avalanche effect of the block ciphers is approximately the same as that of AES-128, and better than those of all the other existing quasigroup based block ciphers given in [5, 6, 83].

5.6.4 Strict avalanche criterion (SAC)

A strict avalanche criterion measures the impact on each bit of the output (ciphertext) by changing the input (plaintext) bits. That is, for a slight change in the plaintext, the impact on each bit of the corresponding output should be uniform. That is, whenever a single bit of the input is changed (from 1 to 0 or from 0 to 1), each of the output's bits changes with a probability of approximately 50% [20]. In order to test whether the proposed ciphers meet this criterion, we used 128 random secret keys. Using each of these secret keys, we encrypted 1024 different randomly generated inputs (plaintexts)

Table 5.9: Number of outputs (ciphertexts) whose hamming distances from the original output C lie in the specified range.

		Range of hamming distance in percentage (hdp_j)					
		≤ 34.99	$35 - 49.99$	$50 - 64.99$	≥ 65	Mean	MAD
Proposed block ciphers							
1^{st} cipher, discussed in section 5.3	$0XFF$	0	57	71	0	50.07	3.12
	$0X00$	0	63	65	0		
2^{nd} cipher, discussed in section 5.4	$0XFF$	0	66	62	0	50.10	3.14
	$0X00$	0	51	77	0		
Existing block ciphers							
Battey et al. [5, 6]	$0XFF$	0	60	68	0	49.37	3.94
	$0X00$	0	70	58	0		
Zhao and Xu [83]	$0XFF$	32	52	44	0	39.32	13.75
	$0X00$	23	49	56	0		
AES-128	$0XFF$	0	60	68	0	49.95	3.24
	$0X00$	0	55	73	0		

of the same length. Then, we changed a particular bit in each of these 1024 inputs (the bit with the same sequence number in all inputs). We encrypted all 1024 modified inputs using each secret key and compared them with the original outputs to see how they differed. Since the size of each randomly generated input is 128 bits, we repeated this process 128 times so that every single bit in each of these inputs is changed. The partial results of this experiment corresponding to the cipher proposed in section 5.3 and the cipher proposed in section 5.4 are shown in Tables 5.10 and 5.11, respectively. Each cell of the table represents the change percentage of the j^{th} bit of the output (ciphertext) when the i^{th} bit of the input (plaintext) is changed, where i is the row number and j is the column number of the table. For example, in the table, it can be verified that when the 120^{th} bit of the inputs is changed, then the 32^{th} bit of the outputs changed in half (50%) of the outputs.

So, from our experimental results, shown in Tables 5.10 and 5.11, it can be observed that when an arbitrary bit of the inputs is changed, each bit of the outputs is changed with the probability of approximately 50%. This implies that if a single bit is changed in all of the 1024 inputs, then each of the output's bits will change in approximately half of the outputs. Hence, the proposed ciphers satisfy the strict avalanche criterion

(SAC).

Table 5.10: Strict avalanche criterion of the proposed cipher, discussed in section 5.3.

		Output (ciphertext) bits							
		1	2	4	8	16	32	64	128
Input (plaintext) bits	1	49.94%	50.08%	49.74%	50.04%	49.97%	49.91%	50.08%	50.02%
	2	50.03%	49.97%	49.89%	50.01%	49.88%	50.01%	50.13%	49.92%
	3	50.00%	49.96%	50.01%	49.82%	49.67%	49.94%	49.97%	50.00%
	4	49.98%	49.95%	49.89%	49.98%	49.93%	49.98%	50.00%	50.00%
	5	50.02%	50.03%	50.07%	49.80%	49.97%	49.76%	50.01%	49.83%
	6	50.00%	50.00%	50.04%	50.01%	50.08%	49.89%	49.93%	50.00%
	7	50.35%	50.02%	50.21%	50.30%	50.01%	49.95%	50.06%	49.93%
	8	50.21%	49.94%	49.99%	50.03%	49.71%	50.26%	50.12%	49.95%
	16	49.93%	50.05%	50.04%	50.00%	50.06%	49.91%	49.86%	49.95%
	20	49.89%	49.80%	50.34%	49.94%	50.14%	49.94%	49.87%	50.01%
	25	50.11%	50.04%	49.90%	50.13%	49.97%	50.00%	50.07%	50.00%
	32	50.24%	50.08%	50.01%	49.94%	49.76%	49.95%	49.78%	49.74%
	40	50.06%	49.89%	49.97%	49.96%	50.07%	50.07%	50.05%	50.06%
	41	50.00%	50.07%	49.99%	50.07%	49.97%	49.93%	50.00%	49.99%
	64	50.09%	50.14%	50.06%	49.99%	50.03%	49.90%	50.01%	49.95%
	100	50.05%	50.14%	50.02%	49.90%	50.20%	50.26%	49.82%	49.91%
	110	50.11%	49.87%	49.86%	50.01%	50.00%	50.10%	49.98%	49.95%
	120	50.00%	50.00%	50.02%	49.88%	49.94%	50.00%	50.13%	49.92%
	127	50.02%	49.89%	50.08%	50.06%	50.00%	49.84%	50.13%	49.95%
	128	49.96%	49.87%	50.00%	50.03%	49.92%	50.00%	50.04%	50.00%

5.6.5 Statistical test for randomness

The ciphertexts created using the proposed ciphers pass various statistical tests of NIST-STS [1]. We evaluated the randomness of the obtained ciphertexts using the NIST-STS test suite. Each test of the NIST-STS package gives a p-value and Success/Fail status. The p-value is the probability that a perfect random number generator would have produced a less random sequence than the one being tested [65]. We have used NIST Spec. Publ. 800-22 rev. 1a package with significance level $\alpha = 0.01$ that consists

[1] National Institute of Standards and Technology - Statistical Test Suite

Table 5.11: Strict avalanche criterion of the proposed cipher, discussed in section 5.4.

		Output (ciphertext) bits							
		1	2	4	8	16	32	64	128
Input (plaintext) bits	1	49.97%	50.07%	50.04%	50.14%	49.87%	49.90%	50.08%	50.04%
	2	49.83%	49.94%	49.87%	50.04%	49.87%	50.03%	50.17%	49.95%
	3	50.09%	49.93%	50.02%	49.82%	49.67%	49.94%	49.97%	49.92%
	4	49.97%	49.94%	49.87%	49.78%	49.94%	49.88%	50.00%	49.76%
	5	50.02%	50.05%	50.07%	49.80%	49.87%	49.76%	50.04%	49.73%
	6	49.65%	50.00%	50.04%	50.01%	50.08%	49.89%	49.93%	50.00%
	7	50.35%	50.02%	50.21%	50.30%	50.00%	49.95%	50.06%	49.93%
	8	50.21%	49.89%	49.99%	50.09%	49.71%	50.26%	50.12%	49.95%
	16	49.83%	50.07%	50.04%	50.30%	50.06%	49.91%	49.86%	49.79%
	20	49.88%	49.80%	50.34%	49.95%	50.14%	49.94%	49.87%	50.00%
	25	50.17%	50.04%	49.90%	50.13%	49.87%	50.00%	50.07%	49.58%
	32	50.24%	50.08%	50.01%	49.94%	49.76%	49.95%	49.78%	49.74%
	40	50.09%	50.17%	49.97%	49.90%	50.07%	50.07%	50.05%	50.10%
	41	50.00%	50.17%	49.89%	50.07%	49.98%	49.91%	50.06%	49.89%
	64	50.11%	50.15%	50.06%	49.99%	50.13%	49.90%	50.01%	49.93%
	100	50.05%	50.14%	50.02%	49.90%	50.20%	50.26%	49.82%	49.91%
	110	50.20%	49.87%	49.86%	50.00%	50.08%	50.10%	49.98%	49.90%
	120	50.03%	50.00%	50.02%	49.85%	49.94%	49.96%	50.13%	49.90%
	127	50.03%	49.79%	50.06%	50.00%	50.08%	49.84%	50.13%	49.89%
	128	49.87%	49.80%	50.16%	50.08%	49.92%	50.00%	50.04%	50.17%

of 15 types of statistical tests [65]. The details of each of these tests are described in section 4.3.8.2 of Chapter 4.

Various data types are defined in [13, 71]. In which, we have chosen a Cipher Block Chaining Mode. This is because the proposed ciphers are implemented based on CBC mode. We randomly chose 128 bits IV, 128 bits secret key K, and 8192 128 bits plaintext blocks for each of these tests. A binary sequence of 1048576 bits is constructed using the ciphertext obtained in the CBC mode. That is, it is a binary sequence obtained by concatenating the 8192 ciphertext blocks of 128 bits each. We generated 1000 such binary sequences for the same plaintext blocks using different random 128 bits keys. We ran each of these tests on the outputs of both the proposed

ciphers and AES-128 1000 times and compared the randomness of the proposed ciphers with that of the AES-128 for each binary sequence of 1048576 bits. The experimental results of this analysis corresponding to the cipher discussed in sections 5.3 and the cipher discussed in section 5.4 are given in Table 5.12. In this table, column A lists the names of the tests carried out. The number of accepted binary sequences corresponding to the first cipher discussed in section 5.3, the second cipher discussed in section 5.4, and the AES-128 that passed a statistical test at the $\alpha = 0.01$ significance level are given in columns B, C, and D, respectively. As a result, the randomness of the proposed ciphers is comparable to that of the AES-128. Hence, from the NIST-STS's point of view, both the proposed ciphers are random.

Table 5.12: For 1000 random keys, results of the NIST test for the proposed encryption systems as compared to the AES-128 encryption system when the same key is used for all cryptosystems with CBC mode of operation.

A	B	C	D
Tests	Proportion of success out of 1000 samples for the 1^{st} cipher, discussed in section 5.3	Proportion of success out of 1000 samples for 2^{nd} cipher, discussed in section 5.4	Proportion of success out of 1000 samples for the AES-128
Frequency	0.993	0.991	0.991
BF	0.983	0.990	0.991
CS	0.994	0.993	0.992
Runs	0.990	0.987	0.989
LR	0.990	0.989	0.992
Rank	0.996	0.985	0.991
DFT	0.982	0.988	0.986
NOT	0.985	0.987	0.980
OT	0.992	0.992	0.994
US	0.985	0.991	0.988
AE	0.991	0.988	0.993
RE	0.989	0.990	0.985
REV	0.991	0.993	0.990
Serial	0.993	0.987	0.994
LC	0.995	0.989	0.981

5.7 Summary

This chapter proposed two block ciphers to encrypt or decrypt data in the form of a block of 128 bits. The proposed ciphers use 16 optimal S-boxes as an optimal quasigroup of order 16, where the size of each S-box is 4×4 bits. The design of the proposed ciphers is based on the concept of multiple quasigroups. They utilize the functionality of the 16 optimal quasigroups derived from an original optimal quasigroup of order 16. In other words, they leverage the space of a single quasigroup and use 16 quasigroups by generating them from an original quasigroup. So, the space required by 16 optimal quasigroups is reduced to that of a single quasigroup. The second cipher discussed in section 5.4 is more standard than the first cipher discussed in section 5.3. This is because each round of the second cipher uses three transformations (substitution, permutation, and add round key), while the first cipher uses only two transformations (substitution and permutation). The proposed ciphers have been analyzed against several attacks, including linear cryptanalysis and differential cryptanalysis, and found that the ciphers are resistant to these attacks. Also, we have analyzed the software performance (time complexity), space complexity, and avalanche effect (diffusion effect) of the proposed ciphers by comparing them with AES-128 and other existing quasigroup based block ciphers [5, 6, 83]. We noted that the avalanche effect of our ciphers is almost the same as that of AES-128 and due to more computations our ciphers are slightly slower than AES-128, but our cipher uses half the space compared to AES-128. Also, the proposed block ciphers use the same amount of space as that used by [83] but 512 times lesser than [5, 6]. We also noted that our ciphers are more efficient than DES. In addition, our ciphers are more than 2 times faster and give a better avalanche effect than other existing quasigroup based block ciphers [5, 6, 83]. Hence, we concluded that our ciphers appear to be an excellent alternative for the quasigroup based proposals. We have also analyzed our ciphers against the strict avalanche criterion (SAC). The results showed that when a random bit of plaintext is changed the proposed ciphers change each bit of the ciphertext with a probability of approximately 50%. Hence the proposed ciphers satisfy the SAC.

Remember that our ciphers can be seen as a family of encryption systems parameterized by an optimal quasigroup of order 16. So, if required, the security of the ciphers can be enhanced by keeping the optimal quasigroup secret along with the secret key.

That is, the security of the proposed ciphers depends not only on the secret key but also on the optimal quasigroup employed. Note that our ciphers use 16 optimal quasigroups in all the 16 rounds, and these 16 quasigroups are generated from the initial optimal quasigroup by circularly shifting the rows. Since an optimal quasigroup is constructed using the 16 optimal S-boxes, we, therefore, can form a maximum of 16! optimal quasigroups by permuting the rows. So, the total key space of our ciphers would then be $C_1^{16!} \times 16^{16} \times 2^{128} \approx 2^{236}$. Therefore, the proposed ciphers can be seen to be more secure than AES-128 against quantum attack since in quantum computing [31], the best quantum attack against any symmetric-key cryptosystem is proportional to the square root of the key space. And, the attack complexity of our cipher against quantum attack is about 2^{118}, while in the case of AES-128 is only 2^{64}.

The randomness of the obtained ciphertexts produced by the proposed ciphers is tested using the NIST statistical test suite. We ran our encryption systems for a random plaintext of 1048576 bits with 1000 different keys and generated 1000 ciphertexts. The results of the proposed ciphers are compared with that of AES-128 for the same plaintext and the same keys. We observed that the randomness of the outputs of our ciphers and AES-128 are comparable to each other.

Chapter 6

Hash Functions and HMACs bsed on quasigroup

This chapter introduces two variants of cryptographic hash functions and their corresponding message authentication codes (HMACs). Both hash functions can be seen as extensions of the MD5 hash function. The underlying structure of the new hash functions is based on MD5 and a quasigroup of order 16 *or* 256*. This chapter gives a brief overview of the proposed hash functions and HMACs, describes the structure and building elements of the proposed schemes, and analyzes the performance and security of the proposed schemes.*

6.1 Introduction

As we know, the encryption/decryption method is used to achieve confidential communication. It protects against passive attacks, where the cryptanalyst only observes messages transmitted between sender and receiver. For active attacks, on the other hand, the cryptanalyst can also change the content of messages during the transmission. To mitigate this type of threat, encryption itself is not sufficient. So, we need another cryptographic tool called a hash function that will enable us to detect when a modification has occurred. Modern hash functions can be divided into two types: (i) a hash function without a key and (ii) a hash function with a key, also called HMAC. A cryptographic hash function (or HMAC) is a one-way compression function that compresses a variable-length message to a fixed-length hash value (or MAC value). The

block diagram of the hash and HMAC algorithms is shown in Figure 6.1. In this figure, H denotes the hash function that takes a variable-length message M and produces a fixed length hash value $H(M)$; the size of $H(M)$ depends on the hash algorithm. The value of $H(M)$ is used to detect the integrity of the transmitted message M. On the other hand, H_k denotes the HMAC that takes two inputs (i) a variable-length message M and (ii) a secret key k and produces a fixed length MAC value $H_k(M)$, also called an authentication tag; the size of $H_k(M)$ depends on the HMAC algorithm. The value of $H_k(M)$ is used to simultaneously verify the authenticity and the integrity of the transmitted message M when two authorized parties communicate in an insecure channel. More details about the cryptographic hash functions and HMACs are given in Chapter 1.

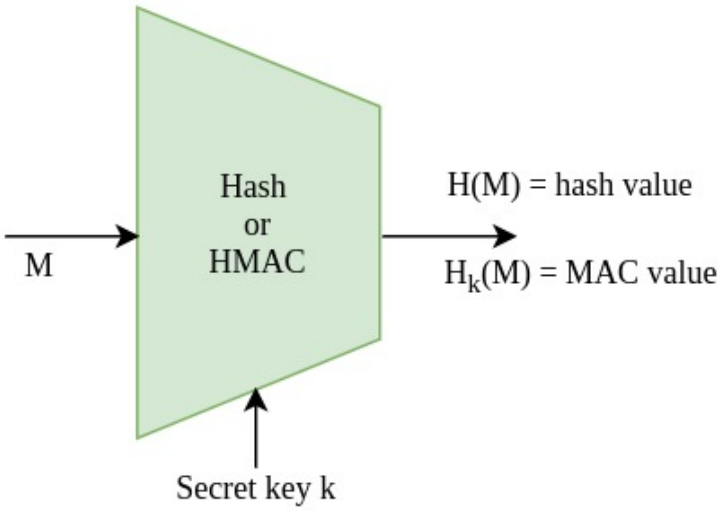


Figure 6.1: Hash function and HMAC.

6.2 Overview of the proposed hash functions and HMACs.

This chapter proposes two extended versions of the MD5 and HMAC-MD5 based on quasigroup. The first hash function and HMAC, named QGMD5-224 and QGMAC-224, generate a 224-bit hash value and MAC value, respectively. And the second hash function and HMAC, named QGMD5-384 and QGMAC-384, generate a 384-bit hash value and MAC value, respectively. All the schemes are designed based on the quasigroup. Note that the underlying structure of both the hash function and

message authentication code is similar. The only difference between the two is that the quasigroup used in the hash function is publicly known, while the quasigroup used in the message authentication code acts as a secret key. Also, depending on the algorithms, the proposed schemes use a quasigroup of order 16 or 256. Note that the use of a quasigroup of order 256 will provide more security than the use of a quasigroup of order 16. But to store a quasigroup of order 256 more space is required than that of a quasigroup of order 16, and this may be a challenge for small computing devices. So, initially, we prefer to use the order 16 quasigroups, and later, we can use order 256 if needed.

All the proposed schemes are iterative in nature. And for each iteration, they take a 512-bit input block of the message M, and produce 224 or 384 bits as hash (MAC) value. Let input message M be divided into ℓ blocks $B_1, B_2, \ldots, B_\ell$, where the size of each block B_i is 512 bits, $1 \leq i \leq \ell$. All proposed schemes are implemented using the cipher block chaining (CBC) mode of operation. Each scheme processes the input message M block by block and produces an output of 224 or 384 bits hash (MAC) value depending on the algorithm used is either QGMD5-224 (QGMAC-224) or QGMD5-384 (QGMAC-384), respectively. Processing of each of these blocks is as follows: Let $i \geq 1$ be a fixed integer less than ℓ. Each of the proposed schemes takes block B_i together with the initial value IV_{i-1} as input, performs four rounds, and outputs a 128-bit IV_i. Note that IV_0 is the initial value chosen at the beginning. j^{th}, $1 \leq j \leq 4$, round of the algorithms consists of j^{th} round of MD5, followed by the quasigroup based expansion and compression operations. The processing of the last block B_ℓ is exactly the same as that of the previous blocks $B_1, B_2, \ldots, B_{\ell-1}$ except that the last (4^{th}) round of the algorithms consists of Round 4 of MD5 followed by only the expansion operation.

6.2.1 Brief description of MD5

MD5 is one of the most widely used hash functions in cryptography since it requires the least number of computations. It is iterative in nature and designed based on Merkle-Damgard construction. As input, it takes a variable-length message M and produces an output with a fixed length of 128 bits as the hash value. In order to begin the process, the entire message M is divided into 512-bit blocks. There is padding applied if the length of message M is not a multiple of 512 bits, and it is padded by adding a sufficient number of 0's after a bit 1 to bring the length of message M to a

multiple of 512 bits minus 64 bits. Once the padding has been applied, append 64 bits representation of the length of the original message M if the original message length is less than or equal to 2^{64}; otherwise, the lower order 64 bits of the representation of the original message M are used so that the resulting message is an exact multiple of 512 bits as shown in Figure 6.2. Now each of these 512-bit blocks of a message M is

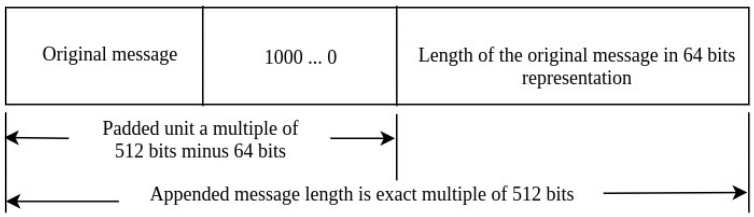


Figure 6.2: Length of the message after padding

processed by dividing it into sixteen 32-bit words. The algorithm of MD5 has 4 rounds, each of which has 16 steps, for a total of 64 steps. Each 512 bits block of M passes through the following 4 round functions:

$$\begin{aligned} R_{(1,p)}(X,Y,Z) &= (X \wedge Y) \vee (\neg X \wedge Z), \quad 1 \leq p \leq 16 \\ R_{(2,p)}(X,Y,Z) &= (X \wedge Z) \vee (Y \wedge \neg Z), \quad 17 \leq p \leq 32 \\ R_{(3,p)}(X,Y,Z) &= (X \oplus Y \oplus Z), \qquad 33 \leq p \leq 48 \\ R_{(4,p)}(X,Y,Z) &= Y \oplus (X \vee \neg Z), \qquad 49 \leq p \leq 64 \end{aligned} \tag{6.1}$$

where, X, Y, Z are 32 bit words and $\wedge, \vee, \oplus$ and $\neg$ are AND, OR, XOR, and NOT operations, respectively. The $R_{(r,p)}$ denotes the r^{th} round function in the p^{th} step, $1 \leq r \leq 4$, $1 \leq p \leq 64$. Each step of the MD5 operates on four 32 bits words W, X, Y, and Z, and performs the operation as follows:

$$\begin{aligned} W &= Z \\ X &= ((W + R_{(r,p)}(X,Y,Z) + m_i + k_j) \lll_s) + X \\ Y &= X \\ Z &= Y \end{aligned} \tag{6.2}$$

where the operation + denotes the addition modulo 2^{32}, m_i denotes a message word of 32 bits, k_j denotes a step-specific constant, and $\lll_s$ is a specific constant that specifies a left-circular shift by s-bit position. The graphical representation of one step operation of MD5 is shown in Figure 6.3.

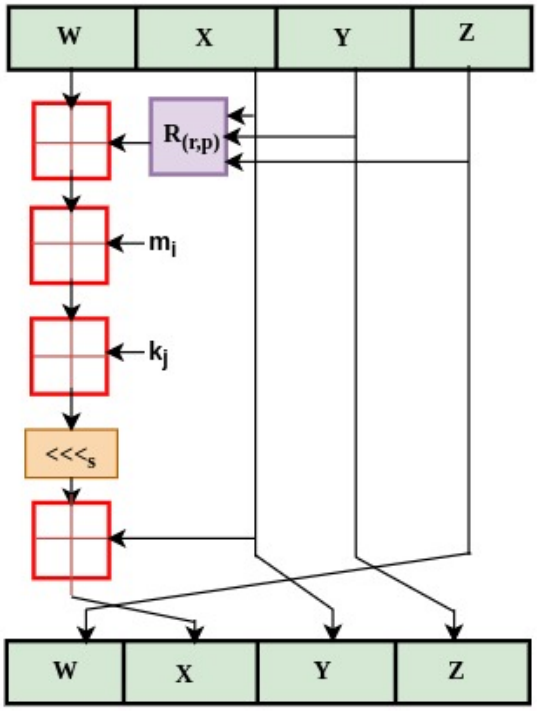


Figure 6.3: One step operation of MD5 hash function

6.3 A QGMD5-224 bits hash function and a QGMAC-224 bits message authentication code based on a quasigroup

This section proposes two new schemes based on a quasigroup: (i) a cryptographic hash function, named here as QGMD5-224, and (ii) a message authentication code based on QGMD5-224, named here as QGMAC-224. The QGMD5-224 hash function expands the hash size of the MD5 hash function by converting 128 bits into 224 bits. The QGMAC-224 expands the MD5 based message authentication code (HMAC-MD5) by converting 128 bits into 224 bits. Both expansions are carried out using the quasigroup expansion (QGExp128To224) and the quasigroup compression (QGComp224To128) layers. Note that the underlying structure of both the schemes QGMD5-224 and QGMAC-224 is similar. The only difference between the two is that the quasigroup used in QGMD5-224 is publicly known, while the quasigroup used in QGMAC-224 acts as a secret key. The workflow of both the QGMD5-224 and the QGMAC-224 is shown

in Figure 6.4. In order to start the process of each of these schemes, an arbitrary length

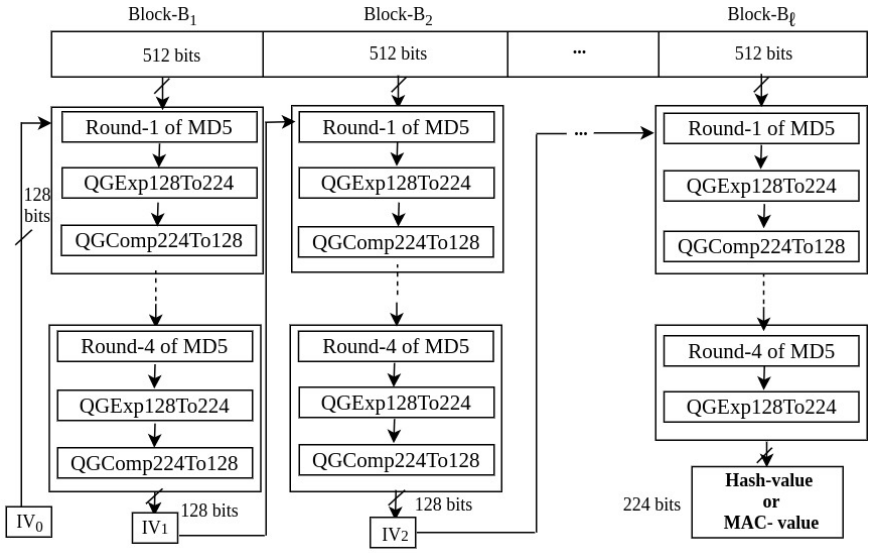


Figure 6.4: Workflow of QGMD5-224 and QGMAC-224

message M is first divided into ℓ fixed-size blocks, where the size of each block is 512 bits. If the length of the message M is not a multiple of 512 bits, then the padding will be required, and it is padded as in the case of the MD5 hash function, such as discussed earlier in section 6.2.1. Observe that each round, except the last round of the last block of MD5, is followed by a QGExp128To224 layer that expands 128 bits into 224 bits by inserting 96 bits and a QGComp224To128 layer that compresses back to 128 bits by deleting 96 bits. The last round of the last block of MD5 is followed by only a QGExp128To224 layer. Both layers QGExp128To224 and QGComp224To128 are defined using the quasigroup expansion (QGExp) and the quasigroup compression (QGComp) operations, respectively. These operations are defined by confining to the rules of the selected quasigroup. Depending on the algorithm, the proposed schemes use quasigroups of orders 16 or 256. The functioning of QGExp and QGComp operations with these order quasigroups is explained separately in detail.

6.3.1 Quasigroup expansion (QGExp) operation

The QGExp operation works byte-by-byte, and for each expansion operation, it takes two bytes of data and produces a sequence of three bytes of data. Let each 8-bit (one

byte) value be divided into two 4-bit values. That is, a character (one-byte value) p is represented as $p = p_1p_0$, where p_0 and p_1 are 4-bit values (hexadecimal digits or nibble values). The proposed schemes use quasigroups of order 16 and 256. Therefore, for the quasigroup of order 256, it is defined as follows:

$$p_1p_0 \circledast_1 q_1q_0 = (p_1p_0, q_1q_0, r_1r_0) \tag{6.3}$$

where $r_1r_0 = p_1p_0 *_1 q_1q_0$, and $*_1$ and $\circledast_1$ denote the quasigroup operation and the QGExp operation corresponding to order 256, respectively. Note that the resultant element r_1r_0 is determined by looking up the element having the row index of p_1p_0 and the column index of q_1q_0 in the table representation of the quasigroup of order 256.

Now, for the quasigroup of order 16, the QGExp operation is defined as follows:

$$p_1p_0 \circledast_2 q_1q_0 = (p_1p_0, q_1q_0, r_1||r_0) \tag{6.4}$$

where $r_1 = p_1 *_2 q_1$, $r_0 = p_0 *_2 q_0$, and $*_2$ and $\circledast_2$ denote the quasigroup operation and the QGExp operation corresponding to order 16, respectively and $||$ is the concatenation operation that concatenates two 4-bit value to make one 8-bit value. Note that r_0 is determined by looking up the element having the row index of p_0 and the column index of q_0 in the table representation of the quasigroup of order 16. Similarly, r_1 is determined by looking up the element having the row index of p_1 and the column index of q_1 in the table representation of the quasigroup of order 16.

A general application of the QGExp operation for a pair of sequences of elements can be defined as follows:
Let $P = (p_1^1p_0^1, p_1^2p_0^2, \ldots, p_1^tp_0^t)$ and $Q = (q_1^1q_0^1, q_1^2q_0^2, \ldots, q_1^tq_0^t)$, where $p_1^ip_0^i$ and $q_1^jq_0^j$ are byte values whereas p_0^i, p_1^i, q_0^j, and q_1^j are nibble (4-bit) values, for $1 \leq i, j \leq t$, then

$$(P \circledast_1 Q) \text{ or } (P \circledast_2 Q) = ((p_1^1p_0^1, q_1^1q_0^1, r_1^1r_0^1), (p_1^2p_0^2, q_1^2q_0^2, r_1^2r_0^2), \ldots, (p_1^tp_0^t, q_1^tq_0^t, r_1^tr_0^t))$$

where $r_1^jr_0^j = p_1^jp_0^j *_1 q_1^jq_0^j$, $*_1$ is the quasigroup operation of order 256 with respect to the QGExp operation $\circledast_1$ or $r_1^jr_0^j = (p_1^j *_2 q_1^j)||(p_0^j *_2 q_0^j)$, $*_2$ is the quasigroup operation of order 16 with respect to the QGExp operation $\circledast_2$ and $||$ is the concatenation operation.

Similarly if $P = ((p_1^{11}p_0^{11}, p_1^{12}p_0^{12}, \ldots, p_1^{1k}p_0^{1k}), (p_1^{21}p_0^{21}, p_1^{22}p_0^{22}, \ldots, p_1^{2k}p_0^{2k}), \ldots,$
$(p_1^{t1}p_0^{t1}, p_1^{t2}p_0^{t2}, \ldots, p_1^{tk}p_0^{tk}))$ and $Q = (q_1^1q_0^1, q_1^2q_0^2, \ldots, q_1^tq_0^t)$, where $p_1^{ij}p_0^{ij}$ is a byte value,

p_0^{ij} and p_1^{ij} are nibble (4-bit) values for $1 \leq i \leq t$, $1 \leq j \leq k$, $q_1^l q_0^l$ is a byte value, q_0^l and q_1^l are nibble (4-bit) values for $1 \leq l \leq t$, then

$$\begin{aligned}(P \circledast_1 Q) \; or \; (P \circledast_2 Q) = ((&p_1^{11}p_0^{11}, p_1^{12}p_0^{12}, \ldots, p_1^{1k}p_0^{1k}, q_1^1q_0^1, r_1^1r_0^1),\\ &(p_1^{21}p_0^{21}, p_1^{22}p_0^{22}, \ldots, p_1^{2k}p_0^{2k}, q_1^2q_0^2, r_1^2r_0^2),\\ &\ldots,\\ &(p_1^{t1}p_0^{t1}, p_1^{t2}p_0^{t2}, \ldots, p_1^{tk}p_0^{tk}, q_1^tq_0^t, r_1^tr_0^t))\end{aligned}$$

where $r_1^i r_0^i = p_1^{ik}p_0^{ik} *_1 q_1^i q_0^i$, $*_1$ is the quasigroup operation of order 256 with respect to the QGExp operation $\circledast_1$ or $r_1^i r_0^i = (p_1^{ik} *_2 q_1^i)||(p_0^{ik} *_2 q_0^i)$, $*_2$ is the quasigroup operation of order 16 with respect to the QGExp operation $\circledast_2$ and $||$ is the concatenation operation.

6.3.1.1 QGExp128To224 layer

The quasigroup expansion layer (QGExp128To224) uses the QGExp operation for expanding the intermediate result of 128 bits into 224 bits by inserting 96 bits. It works as follows:

Let $P = (p_1^1p_0^1, p_1^2p_0^2, p_1^3p_0^3, p_1^4p_0^4)$, $Q = (q_1^1q_0^1, q_1^2q_0^2, q_1^3q_0^3, q_1^4q_0^4)$, $U = (u_1^1u_0^1, u_1^2u_0^2, u_1^3u_0^3, u_1^4u_0^4)$, and $V = (v_1^1v_0^1, v_1^2v_0^2, v_1^3v_0^3, v_1^4v_0^4)$, where $p_1^ip_0^i, q_1^iq_0^i, u_1^iu_0^i$, and $v_1^iv_0^i$ are 8-bit (byte) values, whereas $p_0^i, q_0^i, u_0^i, v_0^i, p_1^i, q_1^i, u_1^i$, and v_1^i are 4-bit (nibble) values, $1 \leq i \leq 4$. Then

$$\begin{aligned}(P \circledast Q) = ((&p_1^1p_0^1, q_1^1q_0^1, r_1^1r_0^1), (p_1^2p_0^2, q_1^2q_0^2, r_1^2r_0^2),\\ &(p_1^3p_0^3, q_1^3q_0^3, r_1^3r_0^3), (p_1^4p_0^4, q_1^4q_0^4, r_1^4r_0^4))\\ ((P \circledast Q) \circledast U) = ((&p_1^1p_0^1, q_1^1q_0^1, r_1^1r_0^1, u_1^1u_0^1, r_1^5r_0^5),\\ &(p_1^2p_0^2, q_1^2q_0^2, r_1^2r_0^2, u_1^2u_0^2, r_1^6r_0^6),\\ &(p_1^3p_0^3, q_1^3q_0^3, r_1^3r_0^3, u_1^3u_0^3, r_1^7r_0^7),\\ &(p_1^4p_0^4, q_1^4q_0^4, r_1^4r_0^4, u_1^4u_0^4, r_1^8, r_0^8))\\ (((P \circledast Q) \circledast U) \circledast V) = ((&p_1^1p_0^1, q_1^1q_0^1, r_1^1r_0^1, u_1^1u_0^1, r_1^5r_0^5, v_1^1v_0^1, r_1^9r_0^9),\\ &(p_1^2p_0^2, q_1^2q_0^2, r_1^2r_0^2, u_1^2u_0^2, r_1^6r_0^6, v_1^2v_0^2, r_1^{10}r_0^{10}),\\ &(p_1^3p_0^3, q_1^3q_0^3, r_1^3r_0^3, u_1^3u_0^3, r_1^7r_0^7, v_1^3v_0^3, r_1^{11}r_0^{11}),\\ &(p_1^4p_0^4, q_1^4q_0^4, r_1^4r_0^4, u_1^4u_0^4, r_1^8, r_0^8, v_1^4v_0^4, r_1^{12}r_0^{12}))\end{aligned}$$

where the symbol $\circledast$ would be either $\circledast_1$ or $\circledast_2$ depending on the order of the quasigroup employed. $r_1^j r_0^j$ is the resultant or inserted byte based on the quasigroup operation, $1 \leq j \leq 12$. The QGExp128To224 converts 128 bits (16 bytes) to 224 bits (28 bytes). Working of this layer is illustrated by the following example.

Example 6.3.2. *Consider the quasigroup of order* 16 *given in Table 2.8 of Chapter 2. And let* $P = (34, AF, A0, 48)$, $Q = (42, 2F, 72, A8)$, $U = (0B, A3, 38, 4C)$, *and* $V = (30, 28, BC, D8)$ *be a sequence of* 128 *bits (16 bytes) of data in hexadecimal digits to be converted to* 224 *bits (28 bytes). Then, the QGExp128To224 layer converts* 128 *bits to* 224 *bits as follows:*

$$\begin{aligned}
(P \circledast_2 Q) &= ((34, 42, \underline{37}), (AF, 2F, \underline{46}), (A0, 72, \underline{CB}), (48, A8, \underline{CA})) \\
((P \circledast_2 Q) \circledast_2 U) &= ((34, 42, 37, 0B, \underline{A9}), (AF, 2F, 46, A3, \underline{CA}), \\
&\quad (A0, 72, CB, 38, \underline{11}), (48, A8, CA, 4C, \underline{DE})) \\
(((P \circledast_2 Q) \circledast_2 U) \circledast_2 V) &= ((34, 42, 37, 0B, A9, 30, \underline{F7}), (AF, 2F, 46, A3, CA, 28, \underline{C0}), \\
&\quad (A0, 72, CB, 38, 11, BC, \underline{3C}), (48, A8, CA, 4C, DE, D8, \underline{7B}))
\end{aligned}$$

where $\circledast_2$ *is the QGExp operation of order* 16. *The inserted bytes are indicated by underlining them. After performing the QGExp128To224, the resulting sequence of* 224 *bits (28 bytes) is* $3442370BA930F7AF2F46A3CA28C0A072CB3811BC3C48A8CA4CDED87B$.

6.3.3 Quasigroup compression (QGComp) operation

The QGComp operation is nothing but a quasigroup operation. It takes two bytes of data as input and produces one byte of data as output. Like the QGExp operation, it is also defined for both quasigroups of orders 16 and 256 as follows:
Let p_0, p_1, q_0, q_1, r_0, and r_1 be 4-bit (nibble) values. Then, for the quasigroup of order 256. It is defined as

$$r_1 r_0 = p_1 p_0 *_1 q_1 q_0 \tag{6.5}$$

where $*_1$ denotes a quasigroup operation of order 256.

And, for a quasigroup of order 16, it is defined as

$$r_1 = p_1 *_2 q_1 || r_0 = p_0 *_2 q_0 \tag{6.6}$$

where $*_2$ denotes a quasigroup operation of order 16. and $||$ is the concatenation operation that concatenates two 4-bit values to make one 8-bit value.

6.3.3.1 QGComp224To128 layer

The quasigroup compression (QGComp224To128) layer is the inverse of the QGExp128To224 layer. It takes 224 bits as input and converts them into 128 bits as output. In other words, QGComp224To128 compresses the partial hash-value (or MAC-value) of 224 bits into 128 bits. This 128-bit output is then fed into the next round of the MD5 algorithm. It works as follows: First, it divides a block of 224 bits (28 bytes) into four sub-blocks of 56 bits (7 bytes) each. Let $P_j = (p_1^1p_0^1, p_1^2p_0^2, p_1^3p_0^3, p_1^4p_0^4, p_1^5p_0^5, p_1^6p_0^6, p_1^7p_0^7)$ be a sub-block of 7 byte, where p_0^t and p_1^t are 4-bit values, and $p_1^tp_0^t$ is a byte value, $1 \leq t \leq 7, 1 \leq j \leq 4$. Then,

$$QGComp(P_j) = (r_1^1r_0^1, r_1^2r_0^2, r_1^3r_0^3, r_1^4r_0^4)$$

where, for a quasigroup of order 256, $r_1^ir_0^i = p_1^ip_0^i *_1 p_1^{8-i}p_0^{8-i}$, $*_1$ is the quasigroup operation of order 256 for $1 \leq i \leq 3$ and $r_1^4r_0^4 = p_1^4p_0^4$.

And, for a quasigroup of order 16, $r_1^ir_0^i = (p_1^i *_2 p_1^{8-i})||(p_0^i *_2 p_0^{8-i})$, $*_2$ is the quasigroup operation of order 16 for $1 \leq i \leq 3$ and $r_1^4r_0^4 = p_1^4p_0^4$. This is illustrated by the following example.

Example 6.3.4. *Consider the quasigroup Q of order* 16 *used in Example* 6.3.2*, which is given in Table* 2.8 *of Chapter* 2*. Also, from Example* 6.3.2*, we consider the sequence of* 224 *bits (*28 *bytes) $P =$* 3442370*BA*930*F*7*AF*2*F*46*A*3*CA*28*C*0*A*072*CB*3811*BC*3*C*48*A*8*C A*4*CDED*87*B, represented in hexadecimal digits, which is to be compressed. These bytes are divided into four sub-blocks as $P_1 =$* 3442370*BA*930*F*7*, $P_2 =$ *AF*2*F*46*A*3*CA*28*C*0*, $P_3 =$ *A*072*CB*3811*BC*3*C, and $P_4 =$* 48*A*8*CA*4*CDED*87*B. Now using the quasigroup of order* 16 *given in Table* 2.8 *of Chapter* 2

$$QGComp224To128(P) = QGComp(P_1)||QGComp(P_2)||QGComp(P_3)||QGComp(P_4)$$

where $QGComp(P_1) =$ B262670B, $QGComp(P_2) =$ ED08D9A3, $QGComp(P_3) =$ F594A738, and $QGComp(P_4) =$ 2C9A064C. After performing the QGComp224To128 on P, the resulting sequence of 128 *bits (16 bytes) is* B262670*BED*08*D*9*A*3*F*594*A*7382*C*9 *A*064*C*.

6.3.5 Algorithm of QGMD5-224 and QGMAC-224

The proposed schemes make use of the QGExp128To224 and QGComp224To128 layers along with the round functions of the MD5 hash function to produce a hash value

of 224 bits or a MAC value of 224 bits. Note that the QGExp128To224 and the QGComp224To128 layers are defined by confining to the rules of the selected quasigroup of order 256 or 16 depending on the algorithm. Also, note that each of the proposed schemes processes the input message block by block. So, a message $M = B_1, B_2, \therefore, B_\ell$ is divided into, say ℓ, message blocks, where the size of each message block B_i is 512 bits, $1 \leq i \leq \ell$. The pseudocode of the algorithm of QGMD5-224 and that of the QGMAC-224 is given in Algorithm 13. In this algorithm, the variables B'_i and B'_ℓ are used to store the output of the round functions of MD5. And the variables $OutPutOfExp$ and $OutPutOfComp$ are used to store the output of the layers $QGExp128To224$ and $QGComp224To128$, respectively.

6.3.6 Implementation and software performance

The proposed schemes have been implemented in C++ on a system with the following configuration: Intel(R) Core(TM) i5-2400 CPU @3.40 GHz processor with 4 GB RAM and 64 bits Linux operating system. The performance of the proposed schemes (QGMD5-224 and QGMAC-224) is analyzed by comparing them with the standard hash functions and message authentication codes, such as MD5, SHA-224, HMAC-MD5, and SHA-HMAC-224. For this analysis, we ran each of these schemes 1000 times for a randomly chosen message M="The brown dog jumps over a lazy cat" and calculated the average execution time in microseconds (μs). For measuring the execution time in microseconds, we used a C++ standard library $< chrono >$ [38]. The results of this analysis are presented in Table 6.5. See that the performance of the QGMD5-224 hash function is compared with that of both MD5 and SHA-224 hash functions, and the performance of the QGMAC-224 is compared with that of both HMAC-MD5 and HMAC-SHA-224. It can be observed that the proposed hash function QGMD5-224 is slightly slower than the MD5 but faster than the SHA-224. And the proposed message authentication code QGMAC-224 is faster than both the HMAC-MD5 and the HMAC-SHA-224.

6.3.7 Security analysis

CrackStation and HashCracker tools were used to analyze the proposed QGMD5-224 hash function against the dictionary attack. These tools are basically intended to crack the hash value of MD4, MD5, etc. They employ massive pre-computed lookup tables to

Algorithm 13: Algorithm of QGMD5-224 and QGMAC-224

Input: 1. Message M in the form of ℓ blocks $B_1, B_2, \ldots, B_\ell$ of 512 bits each.
2. An initial value IV_0 of 128 bits.

Output: A hash value or MAC-value of 224 bits.

[1] **if** $\ell = 1$ **then**
[2] GOTO Line No. [14];
[3] **else**
[4] **for** $i = 1$ *to* $\ell - 1$ **do**
[5] **for** $j = 1$ *to* 4 **do**
[6] **if** $j = 1$ **then**
[7] B'_i =Round-1 of MD5(B_i, IV_{i-1});
[8] **else**
[9] B'_i =Round-j of MD5$(B_i, OutPutOfComp)$;
[10] $OutPutOfExp = QGExp128TO224(B'_i)$;
[11] $OutPutOfComp = QGComp224To128(OutPutOfExp)$;
[12] **if** $j = 4$ **then**
[13] $IV_i = OutPutOfCOmp$;
[14] **for** $j = 1$ *to* 3 **do**
[15] **if** $j = 1$ *AND* $\ell = 1$ **then**
[16] B'_ℓ =Round-1 of MD5(B_1, IV_0);
[17] **else if** $j = 1$ *AND* $\ell > 1$ **then**
[18] B'_ℓ =Round-1 of MD5$(B_\ell, IV_{\ell-1})$;
[19] **else**
[20] B'_ℓ =Round-j of MD5$(B_\ell, OutPutOfComp)$;
[21] $OutPutOfExp = QGExp128TO224(B'_\ell)$;
[22] $OutPutOfComp = QGComp224To128(OutPutOfExp)$;
[23] B'_ℓ =Round-4 of MD5$(B_\ell, OutPutOfComp)$;
[24] $OutPutOfExp = QGExp128TO224(B'_\ell)$. Output these 224 bits as the final hash value or MAC-value of the message M;

Table 6.1: Performance analysis of hash functions and HMACs.

Hash Functions	Avg Execution time in microseconds (μs)	Message Authentication Codes	Avg Execution time in microseconds (μs)
Existing schemes			
MD5	7.94	HMAC-MD5	10.13
SHA-224	10.28	HMAC-SHA-224	15.72
Proposed schemes			
QGMD5-224	9.85	QGMAC-224	9.85

crack password hashes. The QGMD5-224 is also analyzed against various other attacks, including the brute force attack, and it is found to be resistant to these attacks as well. A hash value produced by a hash function determines its strength against brute force attack and the QGMD5-224 produces a 224-bit hash value instead of 128-bit, as in the case of MD5. Typically, for an n-bit hash value, a brute force attack requires 2^n effort to compute the (i) pre-image and (ii) second pre-image attacks, and to find a collision, it requires $2^{n/2}$ effort. Since the size of the hash value of QGMD5-224 is 224 bits as against 128 bits of MD5, the QGMD5-224 can be seen to be more secure than the MD5.

6.3.7.1 Collision Resistance

Collision resistance is one of the important properties of a hash function. That is, a hash function must be collision resistant. This is because the mapping of a hash function between message space and the set of hash values is many-to-one, meaning different messages may have the same hash value. For testing the collision resistance of the proposed QGMD5-224 hash function, we randomly select pairs of messages M_1 and M_2 with hamming distance 1. For each pair of messages M_1 and M_2, we computed the corresponding hash values h_1 and h_2 and stored them in ASCII format (ASCII representation is a sequence of bytes in which each byte value lies from 0 to 255). Now, we perform the following two experiments, which are defined in [82].

In the first experiment, we compare h_1 and h_2 as byte sequences, and a number of bytes that have the same value at the same position, namely the number of hits, is counted as follows:

$$v = \sum_{p=i}^{s} f(d(x_p), d(x^{'}{}_p)), \text{ where } f(x,y) = \begin{cases} 1, & x = y \\ 0, & x \neq y. \end{cases} \tag{6.7}$$

The function $d(.)$ converts the entries to their equivalent decimal values, and s denotes the number of bytes in a hash value. Smaller v characterizes the stronger hash function against collision resistance.

Theoretically, for N independent experiments, the following equation specifies the expected number of times v hits for an s-byte hash value.

$$W_N(v) = N \times Prob\{v\} \tag{6.8}$$

where $Prob\{v\} = \frac{s!}{v!(s-v)!}\left(\frac{1}{256}\right)^v\left(1-\frac{1}{256}\right)^{s-v}$, $v = 0, 1, 2, \ldots, s$. If $v = 0$, a collision will never happen, and if $v = s$, a collision will happen. Using equation (6.8) and for $N = 2048$, we computed the expected values of $W_N(v)$ for $s = 28$ byte hash-values. These results are presented in Table 6.2 (a). The experimental results of SHA-224 and QGMD5-224 are presented in Table 6.2 (b). If we compare the experimental results of SHA-224 and QGMD5-224 with the corresponding expected results, which are tabulated in Table 6.2 (a) and (b). It can be observed that the experimental results

Table 6.2: Results of expected and experimental.

	Expected results ($W_N(v)$)	Experimental results	
v	$s = 28$	SHA-224 ($s = 28$)	QGMD5-224 ($s = 28$)
0	1835.42	1828	1841
1	201.54	212	199
2	10.67	8	8
$v \geq 3$	0	0	0
	(a)	(b)	

of the proposed QGMD5-224 not only coincide very well with the theoretical ones but also it has better collision resistance than that of the SHA-224.

In the second experiment, we calculated the absolute difference (AD) between each pair of h_1 and h_2 as

$$AD = \sum_{p=1}^{s} |d(u_p) - d(u'_p)| \tag{6.9}$$

where $d(u_p)$ and $d(u'_p)$ are the p^{th} byte value of the h_1 and h_2, respectively. The larger value of AD implies a stronger hash function against collision resistance. For each of

the hash functions SHA-224 and QGMD5-224, the simulation of this experiment is also run 2048 times and calculated the minimum, maximum, mean, and mean/char of AD. The results of this experiment are given in Table 6.3. Note that the ideal value of mean/char, as defined in [17], is 85.33. According to the results as shown in Table 6.3, it can be observed that the obtained mean/char value of the proposed QGMD5-224 hash function is closer to the ideal value of 85.33 than that of the SHA-224. Hence, the second experiment also ensures that QGMD5-224 is more secure than the SHA-224.

Table 6.3: Results of the absolute differences.

Hash Functions	Minimum	Maximum	Mean	Mean/Char
SHA-224	707	2417	2308.60	82.45
QGMD5-224	1203	4351	2377.48	84.91

6.3.7.2 Prefix and suffix attacks

A prefix attack is one of the alternative ways to find a collision in the cryptographic hash function. In this attack, an attacker creates a false message by choosing an arbitrary message and appends it to the original message so that the false (appended) message and the original message have the same hash value.

Mathematically, for a given quasigroup Q and an initial value IV, let $M = (m_1, m_2, \ldots, m_k)$ be a message to be hashed, where $m_i \in Q, 1 \leq i \leq k$. Let $P = (p_1, p_2, \ldots, p_u)$ be a prefix to be appended to M where $p_j \in Q, 1 \leq j \leq u$. The attacker can then create a false message $PM = (p_1, p_2, \ldots, p_u, m_1, m_2, \ldots, m_k)$ by adding the prefix P to the original message M, so that $H_{IV}(PM) = H_{IV}(M)$. $H_{IV}(P) = IV$ is the only condition for this to happen. In other words, this attack can be applied if a hash function is vulnerable to a pre-image attack. The proposed QGMD5-224 hash function uses MD5 along with quasigroup-based expansion (QGExp128To224) and compression (QGComp224T128) layers and MD5 is resistant to this pre-image attack. That is, the security of the QGMD5-224 is not only dependent on the MD5 but also on the QGExp128To224 and QGComp224T128 layers. Hence, the QGMD5-224 is resistant to prefix attack.

A similar argument can be used to show that the proposed hash function is resistant to the suffix attack as well. For this attack, first, an attacker chooses a suf-

fix $S = (s_1, s_2, \ldots, s_t)$, $s_j \in Q$, $1 \leq j \leq t$, and tries to create a false message $MS = (m_1, m_2, \ldots, m_k, s_1, s_2, \ldots, s_t)$ by appending the suffix S to the original message M, so that $H_{IV}(MS) = h = H_{IV}(M)$. This can happen only if $H_h(S) = h$.

6.3.7.3 Avalanche effect

An avalanche effect is one of the desirable properties of a hash function. It means a hash function should have a good avalanche effect. That is, the output of the hash function should change significantly for a slight change in input. We have analyzed the proposed QGMD5-224 hash function against this test by comparing it with those of the existing MD5 and SHA-224 hash functions. For this test, we randomly chose a message M="The brown dog jumps over a lazy cat" of 280 bits, and generated 280 messages $M = M_0, M_1, \ldots, M_{279}$ by changing the i^{th} bit (from 0 to 1 or from 1 to 0) of M, $0 \leq i \leq 279$.

Let $h = H(M)$ be the hash value of the original message M and $h_i = H(M_i)$ be the hash values of the messages M_i for $0 \leq i \leq 279$. Since the hash value size of MD5 is 128 bits, it differs from that of SHA-224 and QGMD5-224. So, the hamming distance between h_i and h is measured in percentage as follows:

$$HDP_i = \frac{D(h, h_i)}{NB(h)} \times 100\% \tag{6.10}$$

where HDP_i denotes the hamming distance between h_i and h in percentage for $0 \leq i \leq 279$, $D(h, h_i)$ denotes the hamming distance between h and h_i and $NB(h)$ denotes the total number of binary digits in hash value h. For each of the hash functions MD5, SHA-224, and QGMD5-224, the results of this test are shown in Table 6.4. In this table, the first column shows the range of hamming distances (HDP_i) in the specified range separately; the second, third, and fourth columns of the table show the number of times the hamming distances (HDP_i) of the hash values $h_0, h_1, \ldots, h_{279}$ from h lie in the specified range given in the first column of the table corresponding to MD5, SHA-224, and QGMD5-224, respectively. Also given in the last row of the table is the average (mean) hamming distance in percentage. From these values, it can be observed that the avalanche effect of QGMD5-224 is better than that of both MD5 and SHA-224.

Table 6.4: Hamming distances for MD5, SHA-224 and QGMD5-224.

Range of HDP_i	Number of hash pairs of MD5	Number of hash pairs of SHA-224	Number of hash pairs of QGMD5-224
35 - 44.99	41	19	16
45 - 54.99	206	238	246
55 - 64.99	33	23	18
	Avarage hamming distance		
Mean:	49.76	49.97	50.02

6.3.7.4 Bit variance test

Bit variance test is one of the statistical tests to measure the impact on each bit of hash value by changing the bits of the input message. If there is a slight change in the input message, then the impact of this change on each bit of the corresponding hash value should be uniform. The proposed QGMD5-224 hash function takes a variable length input message and produces a fixed-length 224 bits hash value. For each bit of the hash value, we calculate the probability of this bit being 1. Let $P_i(0)$ be the probability that the i^{th} bit of a hash value is 0. Similarly, let $P_i(1)$ be the probability that the i^{th} bit of a hash value is 1. If $P_i(0) = P_i(1) = \frac{1}{2}$ for all bits of the hash value (i.e. $i = 1, 2, \ldots, 224$), then the QGMD5-224 passes the bit variance test. Since it is computationally difficult to consider all the possible input message bit changes, we evaluated the results for the same messages $M_0, M_1, \ldots, M_{279}$, which were earlier used in the avalanche effect test, and found the following:

$$\text{Number of hash values} = 281$$
$$\text{Mean frequency of 1s (expected)} = 140.5$$
$$\text{Mean frequency of 1s (calculated)} = 140.4$$

According to these results, it can be observed that the average probability of $1s \approx 50\%$. Hence, QGMD5-224 passes the bit variance test.

6.3.7.5 Analysis of QGMAC-224

The proposed message authentication code QGMAC-224 is designed based on the hash function QGMD5-224 and uses a quasigroup of order 16 or 256 depending on the

algorithm. This quasigroup acts as a secret key. Therefore, the security of the QGMAC-224 depends on the hash function QGMD5-224 as well as on the quasigroup of order 256 or 16 that is used. Since the number of quasigroups of orders 16 and 256 is upper bounded by 0.689×10^{138} and 0.753×10^{102805}, respectively, the use of order 256 quasigroups will provide more security than the quasigroups of order 16. But to store a quasigroup of order 256, 65280 bytes extra space is required than that of a quasigroup of order 16.

As of today, a cryptosystem with keyspace 2^{128} is considered to be a secure cryptosystem. So, the use of order 16 quasigroups also provides good security in QGMAC-224. This is because the number of quasigroups of order 16 is $0.689 \times 10^{138} \approx 2^{456}$. That is, the use of order 256 quasigroups can be an alternative option. Because of the large number of quasigroups of either order, it follows that the probability of identifying the employed quasigroup is close to zero. Hence, QGMAC-224 is resistant to brute force attack. Also, QGMAC-224 is analyzed against forgery attack and found to be resistant. In this attack, an attacker chooses a fixed n number of different messages $(M_1, M_2, \ldots, M_n)$ and their corresponding MAC values (authentication tags) $(h_1, h_2, \ldots, h_n)$ and tries to solve the following equations for the secret key k :

$$h_i = H_k(M_i),\ 1 \leq i \leq n \tag{6.11}$$

where, H is the QGMD5-224 hash function and k is the quasigroup employed. If the attacker has knowledge of the secret key k, then the attacker can forge an authentication tag for any chosen message. But the above system of equations has as many solutions as there are quasigroups of order 16 or 256. Hence determining the quasigroup makes it practically impossible. Therefore, the QGMAC-224 is also resistant to forgery attack.

6.4 A QGMD5-384 bits hash function and a QGMAC-384 bits message authentication code based on a quasigroup

This section proposes an extension of the work previously described in section 6.3. In this section, we also discuss two new schemes based on a quasigroup: (i) a cryptographic hash function, named here as QGMD5-384, and (ii) a message authentication code based on QGMD5-384, named here as QGMAC-384. The primary goal of proposing

these new schemes is to obtain a 160-bit longer hash value and MAC value than the previous one by spending a little bit extra time. By which the new schemes are found to be more secure than the previous ones. Also, the algorithm of QGMD5-384 uses an optimal quasigroup of order 16, while the algorithm of QGMAC-384 uses a quasigroup of order 16 or 256, depending on the algorithm. Note that the underlying structure of both the schemes QGMD5-384 and QGMAC-384 is similar. The only difference between the two is that the quasigroup used in QGMD5-384 is publicly known, while the quasigroup used in QGMAC-384 acts as a secret key. The description of an optimal quasigroup of order 16 is given in section 2.2.16 of Chapter 2. In future, a 16 order optimal quasigroup can be replaced by a 256 order optimal quasigroup. It will only be possible if we generate an optimal quasigroup of order 256. This is because the generation of optimal quasigroups of order 2^k is a hard problem, $k \geq 4$.

The proposed schemes can be seen as an expansion of the hash value and MAC value sizes of the MD5 and HMAC-MD5, respectively. Both the expansions are done through a series of QGExp128T384 and QGComp384To128 layers. The QGExp128To384 expansion layer is implemented in two sub-expansion layers. In the first sub-expansion layer, QGExp128To384 transforms 128 bits into 224 bits and is referred to as QGExp128To224. In the second sub-expansion layer, QGExp128To384 transforms 224 bits into 384 bits and is referred to as QGExp224To384. And the QGComp384To128 compression layer compresses 384 bits into 128 bits. The workflow of both the QGMD5-384 and the QGMAC-384 is shown in Figure 6.5. In order to start the process of each of these schemes, an arbitrary length message M is first divided into ℓ fixed-size blocks, where the size of each block is 512 bits. If the length of the message M is not a multiple of 512 bits, then the padding will be required, and it is padded as in the case of the MD5 hash function, such as discussed earlier in section 6.2.1. Observe that each round, except the last round of the last block, MD5 is followed by a QGExp128To224 sub-layer that expands 128 bits to 224 bits by inserting 96 bits, QGExp224to384 sub-layer that expands 224 bits into 384 bits by inserting 160 bits, and a QGComp384To128 layer that compresses back to 128 bits by deleting 256 bits. In the last round of the last block, MD5 is followed by only the QGExp128To224 and QGExp224To384 sub-layers. Both QGExp128To384 and QGComp384To128 layers are defined using the quasigroup expansion (QGExp) and the quasigroup compression (QGComp) operations, respectively. The QGExp and QGComp operations are defined previously in sections 6.3.1

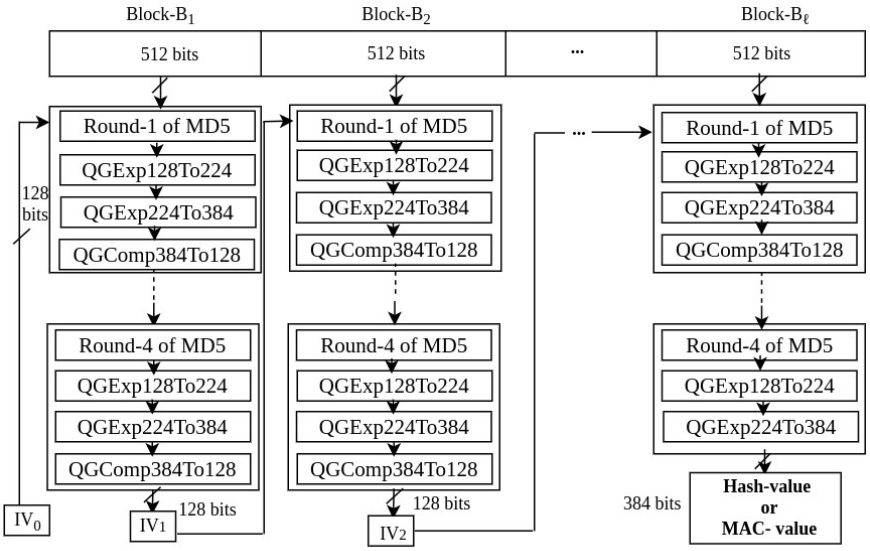


Figure 6.5: Workflow of QGMD5-384 and QGMAC-384

and 6.3.3, respectively by confining to the rules of the selected quasigroup of order 16 or 256, depending on the algorithm.

6.4.1 QGExp128To384 layer

The quasigroup-based expansion layer (QGExp128To384) expands the intermediate result of 128 bits into 384 bits by inserting 256 bits. It is carried out by employing two consecutive sub-layers (i) QGExp128To224 and (ii) QGExp224To384. Each of these sub-layers uses the QGExp operation, defined in section 6.3.1.

6.4.1.1 QGExp128To224 sub-layer

The QGExp128To224 is the first sub-layer of the QGExp128To384 that takes 128 bits as input and converts them into 224 bits as output. It is defined earlier in section 6.3.1.1. And we have the following four consecutive tuples as the output of this sub-layer:

$$P' = (p_1^1 p_0^1, q_1^1 q_0^1, r_1^1 r_0^1, u_1^1 u_0^1, r_1^5 r_0^5, v_1^1 v_0^1, r_1^9 r_0^9),$$
$$Q' = (p_1^2 p_0^2, q_1^2 q_0^2, r_1^2 r_0^2, u_1^2 u_0^2, r_1^6 r_0^6, v_1^2 v_0^2, r_1^{10} r_0^{10}),$$
$$U' = (p_1^3 p_0^3, q_1^3 q_0^3, r_1^3 r_0^3, u_1^3 u_0^3, r_1^7 r_0^7, v_1^3 v_0^3, r_1^{11} r_0^{11}),$$
$$V' = (p_1^4 p_0^4, q_1^4 q_0^4, r_1^4 r_0^4, u_1^4 u_0^4, r_1^8 r_0^8, v_1^4 v_0^4, r_1^{12} r_0^{12}).$$

where the size of each tuple is 56 bits (7 bytes), $p_1^i p_0^i, q_1^i q_0^i, u_1^i u_0^i$, $v_1^i v_0^i$, and $r_1^j r_0^j$ are 8-bit (byte) values, whereas $p_0^i, q_0^i, u_0^i, v_0^i, r_0^j, p_1^i, q_1^i, u_1^i, v_1^i$, and r_1^j are 4-bit (nibble) values, $1 \leq i \leq 4$ and $1 \leq j \leq 12$.

6.4.1.2 QGExp224To384 sub-layer

The QGExp224To384 is the second sub-layer of the QGExp128To384 that takes 224 bits as input and converts them into 384 bits as output. That is, the output of the QGExp128To224 sub-layer is used as the input to the QGExp224To384 sub-layer. It works as follows:

$$\begin{aligned}
(P' \circledast Q') &= ((p_1^1 p_0^1, p_1^2 p_0^2, r_1^{13} r_0^{13}), (q_1^1 q_0^1, q_1^2 q_0^2, r_1^{14} r_0^{14}), (r_1^1 r_0^1, r_1^2 r_0^2, r_1^{15} r_0^{15}),\\
&\quad (u_1^1 u_0^1, u_1^2 u_0^2, r_1^{16} r_0^{16}), (r_1^5 r_0^5, r_1^6 r_0^6, r_1^{17} r_0^{17}), (v_1^1 v_0^1, v_1^2 v_0^2, r_1^{18} r_0^{18}),\\
&\quad (r_1^9 r_0^9, r_1^{10} r_0^{10}, r_1^{19} r_0^{19}))\\
((P' \circledast Q') \circledast U') &= ((p_1^1 p_0^1, p_1^2 p_0^2, r_1^{13} r_0^{13}, p_1^3 p_0^3, r_1^{20} r_0^{20}), (q_1^1 q_0^1, q_1^2 q_0^2, r_1^{14} r_0^{14}, q_1^3 q_0^3, r_1^{21} r_0^{21}),\\
&\quad (r_1^1 r_0^1, r_1^2 r_0^2, r_1^{15} r_0^{15}, r_1^3 r_0^3, r_1^{22} r_0^{22}), (u_1^1 u_0^1, u_1^2 u_0^2, r_1^{16} r_0^{16}, u_1^3 u_0^3, r_1^{23} r_0^{23}),\\
&\quad (r_1^5 r_0^5, r_1^6 r_0^6, r_1^{17} r_0^{17}, r_1^7 r_0^7, r_1^{24} r_0^{24}), (v_1^1 v_0^1, v_1^2 v_0^2, r_1^{18} r_0^{18}, v_1^3 v_0^3, r_1^{25} v_0^{25}),\\
&\quad (r_1^9 r_0^9, r_1^{10} r_0^{10}, r_1^{19} r_0^{19}, r_1^{11} r_0^{11}, r_1^{26} r_0^{26}))\\
(((P' \circledast Q') \circledast U') \circledast V') &= ((p_1^1 p_0^1, p_1^2 p_0^2, r_1^{13} r_0^{13}, p_1^3 p_0^3, r_1^{20} r_0^{20}, p_1^4 p_0^4, r_1^{27} r_0^{27}),\\
&\quad (q_1^1 q_0^1, q_1^2 q_0^2, r_1^{14} r_0^{14}, q_1^3 q_0^3, r_1^{21} r_0^{21}, q_1^4 q_0^4, r_1^{28} r_0^{28}),\\
&\quad (r_1^1 r_0^1, r_1^2 r_0^2, r_1^{15} r_0^{15}, r_1^3 r_0^3, r_1^{22} r_0^{22}, r_1^4 r_0^4, r_1^{29} r_0^{29}),\\
&\quad (u_1^1 u_0^1, u_1^2 u_0^2, r_1^{16} r_0^{16}, u_1^3 u_0^3, r_1^{23} r_0^{23}, u_1^4 u_0^4, r_1^{30} r_0^{30}),\\
&\quad (r_1^5 r_0^5, r_1^6 r_0^6, r_1^{17} r_0^{17}, r_1^7 r_0^7, r_1^{24} r_0^{24}, r_1^8 r_0^8, r_1^{31} r_0^{31}),\\
&\quad (v_1^1 v_0^1, v_1^2 v_0^2, r_1^{18} r_0^{18}, v_1^3 v_0^3, r_1^{25} v_0^{25}, v_1^4 v_0^4, v_1^{32} v_0^{32}),\\
&\quad (r_1^9 r_0^9, r_1^{10} r_0^{10}, r_1^{19} r_0^{19}, r_1^{11} r_0^{11}, r_1^{26} r_0^{26}, r_1^{12} r_0^{12}, r_1^{33} r_0^{33}))
\end{aligned}$$

where the symbol $\circledast$ would be either $\circledast_1$ or $\circledast_2$ depending on quasigroup of order 256 or 16 employed. The symbols $\circledast_1$ and $\circledast_2$ are defined in Equation (6.3) and Equation (6.4), respectively. $r_1^t r_0^t$, $13 \leq t \leq 33$ is the resulting or inserting byte based on quasigroup operation. Note that the QGExp224To384 sub-layer converts 224 bits (28 bytes) to 392 bits (49 bytes). The last byte $r_1^{33} r_0^{33}$ is deleted as the proposed schemes only need 384 bits (48 bytes). The following example illustrates these expansion sub-layers.

Example 6.4.2. *Consider the quasigroup of order* 16 *given in Table 2.8 of Chapter 2. And let* $P = (34, AF, A0, 48)$, $Q = (42, 2F, 72, A8)$, $U = (0B, A3, 38, 4C)$, *and* $V = (30, 28, BC, D8)$ *be a sequence of* 128 *bits (*16 *bytes) of data in hexadecimal digits to be converted to* 384 *bits (*48 *bytes). Then, the first expansion sub-layer (QGExp128To224) of QGExp128To384 converts* 128 *bits to* 224 *bits as follows:*

$$\begin{aligned}
(P \circledast_2 Q) &= ((34, 42, \underline{37}), (AF, 2F, \underline{46}), (A0, 72, \underline{CB}), (48, A8, \underline{CA})) \\
((P \circledast_2 Q) \circledast_2 U) &= ((34, 42, 37, 0B, \underline{A9}), (AF, 2F, 46, A3, \underline{CA}), \\
&\quad (A0, 72, CB, 38, \underline{11}), (48, A8, CA, 4C, \underline{DE})) \\
(((P \circledast_2 Q) \circledast_2 U) \circledast_2 V) &= ((34, 42, 37, 0B, A9, 30, \underline{F7}), (AF, 2F, 46, A3, CA, 28, \underline{C0}), \\
&\quad (A0, 72, CB, 38, 11, BC, \underline{3C}), (48, A8, CA, 4C, DE, D8, \underline{7B}))
\end{aligned}$$

The output of QGExp128To224 becomes the input to the second expansion sub-layer (QGExp224To384) of the expansion layer QGExp128To384, where $(34, 42, 37, 0B, A9, 30, F7)$, $(AF, 2F, 46, A3, CA, 28, C0)$, $(A0, 72, CB, 38, 11, BC, 3C)$, *and* $(48, A8, CA, 4C, DE, D8, 7B)$ *are considered as* P', Q', U', *and* V', *respectively. Now, the second expansion sub-layer (QGExp224To384) of QGExp128To384 converts* 224 *bits to* 384 *bits as follows:*

$$\begin{aligned}
(P' \circledast_2 Q') &= ((34, AF, \underline{64}), (42, 2F, \underline{75}), (37, 46, \underline{33}), (0B, A3, \underline{F9}), (A9, CA, \underline{E5}), \\
&\quad (30, 28, \underline{1D}), (F7, C0, \underline{91})) \\
((P' \circledast_2 Q') \circledast_2 U') &= ((34, AF, 64, A0, \underline{99}), (42, 2F, 75, 72, \underline{D2}), (37, 46, 33, CB, \underline{07}), \\
&\quad (0B, A3, F9, 38, \underline{4C}), (A9, CA, E5, 11, \underline{48}), (30, 28, 1D, BC, \underline{36}), \\
&\quad (F7, C0, 91, 3C, \underline{EC})) \\
(((P' \circledast_2 Q') \circledast_2 U') \circledast_2 V') &= ((34, AF, 64, A0, 99, 48, \underline{BC}), (42, 2F, 75, 72, D2, A8, \underline{04}), \\
&\quad (37, 46, 33, CB, 07, CA, \underline{5E}), (0B, A3, F9, 38, 4C, 4C, \underline{88}), \\
&\quad (A9, CA, E5, 11, 48, DE, \underline{34}), (30, 28, 1D, BC, 36, D8, \underline{87}), \\
&\quad (F7, C0, 91, 3C, EC, 7B, \underline{A6}))
\end{aligned}$$

where $\circledast_2$ *is the QGExp operation of order* 16. *The inserted bytes are indicated by underlining them. After performing the QGExp128To384, the resulting sequence of* 384 *bits (*48 *bytes) is* $34AF64A09948BC422F7572D2A804374633CB07CA5E0BA3F9384C4C8$ $8A9CAE51148DE3430281DBC36D887F7C0913CEC7B$.

6.4.3 QGComp384To128 layer

The quasigroup-based compression layer (QGExp384To128) compresses the intermediate result of 384 bits into 128 bits by deleting 256 bits. It is carried out using the

QGComp operation, defined in section 6.3.3. The application of QGComp384To128 is as follows: First, it divides the 384 bits (48 bytes) into 3 sub-blocks of 128 bits (16 bytes) each. Let $P = (p_1^1 p_0^1, p_1^2 p_0^2, \ldots, p_1^{16} p_0^{16})$, $Q = (q_1^1 q_0^1, q_1^2 q_0^2, \ldots, q_1^{16} q_0^{16})$, and $U = (u_1^1 u_0^1, u_1^2 u_0^2, \ldots, u_1^{16} u_0^{16})$ be a sequence 3 sub-blocks of 48 bytes each, where $p_1^i p_0^i, q_1^i q_0^i$, and $u_1^i u_0^i$ are bytes values, whereas $p_1^i, q_1^i, u_1^i, p_0^i, q_0^i$, and u_0^i are nibble values, $1 \leq i \leq 16$. Then

$$(P \odot_1 Q) \text{ or } (P \odot_2 Q) = (x_1^1 x_0^1, x_1^2 x_0^2, \ldots, x_1^{16} x_0^{16})$$

where $x_1^i x_0^i = p_1^i p_0^i *_1 q_1^i q_0^i$ or $x_1^i x_0^i = (p_1^i *_2 q_1^i) || (p_0^i *_2 q_0^i)$, and

$$((P \odot_1 Q) \odot_1 U) \text{ or } ((P \odot_2 Q) \odot_2 U) = (y_1^1 y_0^1, y_1^2 y_0^2, \ldots, y_1^{16} y_0^{16})$$

where $y_1^i y_0^i = x_1^i x_0^i *_1 u_1^i u_0^i$ or $y_1^i y_0^i = (x_1^i *_2 u_1^i) || (x_0^i *_2 u_0^i)$. The symbols $\odot_1$ and $\odot_2$ denote the quasigroup-based compression operations corresponding to the quasigroup of orders 256 and 16, respectively. And the symbols $*_1$ and $*_2$ are the quasigroup operations corresponding to the symbols $\odot_1$ and $\odot_2$ of the orders 256 and 16, respectively. The following example illustrates the functioning of this compression layer.

Example 6.4.4. *Consider the quasigroup Q of order 16 used in Example 6.4.2. This is given in Table 2.8 of Chapter 2. Also, from Example 6.4.2, we consider the sequence of 384-bit (48 bytes) $P = 34AF64A09948BC422F7572D2A804374633CB07CA5E0BA3F9384C4C88A9CAE51148DE3430281DBC36D887F7C0913CEC7B$ represented in hexadecimal digits. To compress these bytes are divided into three sub-blocks as $P_1 = 34AF64A09948BC422F7572D2A8043746$, $P_2 = 33CB07CA5E0BA3F9384C4C88A9CAE511$, and $P_3 = 48DE3430281DBC36D887F7C0913CEC7B$. Now using the quasigroup of order 16 given in Table 2.8 of Chapter 2, the QGComp384To128 layer works as follows:*

$$(P_1 \odot_2 P_2) = C6E2C2EF2D9CA14F884A4124255C2FD1,$$

and then

$$(P_1 \odot_2 P_2) \odot_2 P_3) = D76A163D0260BC6CDA5C4EC9F8B8A953,$$

where $\odot_2$ is the quasigroup-based compression operation of order 16. The obtained result $D76A163D0260BC6CDA5C4EC9F8B8A953$ consists of 16 bytes and hence is 128 bits.

6.4.5 Algorithm of QGMD5-384 and QGMAC-384

The proposed schemes make use of the QGExp128To384 and QGComp384To128 layers along with the round functions of the MD5 to produce a hash value of 384 bits or a MAC value of 384 bits. The algorithm of QGMD5-384 uses an optimal quasigroup of order 16, and the algorithm of QGMAC-384 uses a quasigroup of order 16 or 256, depending on the algorithm. Each of the proposed schemes processes the input message block by block. So, a message $M = B_1, B_2, \therefore, B_\ell$ is divided into, say ℓ, message blocks, where the size of each message block B_i is 512 bits, $1 \leq i \leq \ell$. The pseudocode of the algorithm of QGMD5-384 and that of the QGMAC-384 is given in Algorithm 14. In this algorithm, the variables B'_i and B'_ℓ are used to store the output of the round functions of MD5. And the variables $OutPutOfFirstSubExpL$, $OutPutOfSecondSubExpL$, and $OutPutOfCompL$ are used to store the output of the layers $QGExp128To224$, $QGExp224To384$, and $QGComp384To128$, respectively.

6.4.6 Implementation and software performance

The proposed schemes have been implemented using the same system configuration and software tools used to implement the previous schemes. The details of these are given in section 6.3.6. Inputs to these schemes are also the same as those used in the previous schemes. The performance of the proposed schemes (QGMD5-384 and QGMAC-384) is compared with those of the standard hash functions and message authentication codes, such as MD5, SHA-384, HMAC-MD5, and HMAC-SHA-384. The results of this analysis are presented in Table 6.5. See that the performance of the QGMD5-384 hash function is compared with that of both MD5 and SHA-384 hash functions, and the performance of the QGMAC-384 is compared with that of both HMAC-MD5 and HMAC-SHA-384. It can be observed that the performance of QGMD5-384 and QGMAC-384 is slightly slower than that of MD5 and HMAC-MD5 but faster than that of SHA-384 and HMAC-SHA-384, respectively.

6.4.7 Security analysis

The security of the proposed QGMD5-384 hash function is analyzed against various attacks, including brute force, collision, prefix, and suffix attacks. And it is found to be resistant to these attacks.

Algorithm 14: Algorithm of QGMD5-384 and QGMAC-384

Input: 1. Message M in the form of ℓ blocks $B_1, B_2, \ldots, B_\ell$ of 512 bits each.
2. An initial value IV_0 of 128 bits.

Output: A hash value or MAC-value of 384 bits.

[1] **if** $\ell = 1$ **then**
[2] GOTO Line No. [15];
[3] **else**
[4] **for** $i = 1$ *to* $\ell - 1$ **do**
[5] **for** $j = 1$ *to* 4 **do**
[6] **if** $j = 1$ **then**
[7] B'_i =Round-1 of MD5(B_i, IV_{i-1});
[8] **else**
[9] B'_i =Round-j of MD5$(B_i, OutPutOfComp)$;
[10] $OutPutOfFirstSubExpL = QGExp128TO224(B'_i)$;
[11] $OutPutOfSecondSubExpL =$
$QGExp224TO384(OutPutOfFirstSubExpL)$;
[12] $OutPutOfCompL =$
$QGComp384To128(OutPutOfSecondSubExpL)$;
[13] **if** $j = 4$ **then**
[14] $IV_i = OutPutOfCompL$;
[15] **for** $j = 1$ *to* 3 **do**
[16] **if** $j = 1$ *AND* $\ell = 1$ **then**
[17] B'_ℓ =Round-1 of MD5(B_1, IV_0);
[18] **else if** $j = 1$ *AND* $\ell > 1$ **then**
[19] B'_ℓ =Round-1 of MD5$(B_\ell, IV_{\ell-1})$;
[20] **else**
[21] B'_ℓ =Round-j of MD5$(B_\ell, OutPutOfCompL)$;
[22] $OutPutOfFirstSubExpL = QGExp128TO224(B'_\ell)$;
[23] $OutPutOfSecondSubExpL =$
$QGExp224TO384(OutPutOfFirstSubExpL)$;
[24] $OutPutOfCompL = QGComp384To128(OutPutOfSecondSubExpL)$;
[25] B'_ℓ =Round-4 of MD5$(B_\ell, OutPutOfCompL)$;
[26] $OutPutOfFirstSubExpL = QGExp128TO224(B'_\ell)$;
[27] $OutPutOfSecondSubExpL = QGExp224TO384(OutPutOfFirstSubExpL)$.

Output these 384 bits as the final hash value or MAC-value of the message M;

Table 6.5: Performance analysis of hash functions and HMACs.

Hash Functions	Avg Execution time in microseconds (μs)	Message Authentication Codes	Avg Execution time in microseconds (μs)
Existing schemes			
MD5	7.94	HMAC-MD5	10.13
SHA-384	17.92	HMAC-SHA-384	22.04
Proposed schemes			
QGMD5-384	16.52	QGMAC-384	16.52

6.4.7.1 Brute force attack

The QGMD5-384 produces a 384 bits hash value instead of 128 bits and 224 bits, as in the case of MD5 and QGMD5-224, respectively. Typically, for an n-bit hash value, a brute force attack requires 2^n effort to compute (i) pre-image and (ii) second pre-image attacks, and to find a collision, it requires $2^{n/2}$ effort. Since the size of the hash value of QGMD5-384 is 384 bits as against 128 bits and 224 bits of MD5 and QGMD5-224, respectively, the QGMD5-384 can be seen to be more secure than both the MD5 and the previous QGMD5-224 hash functions.

6.4.7.2 Collision attack

We have used the same procedure to analyze the QGMD5-384 hash function against this attack that was used to analyze the previous QGMD5-224 hash function; details are given in section 6.3.7.1. That is, we perform two experiments by randomly choosing $N = 2048$ pairs of messages M_1^i and M_2^i with hamming distance 1, $1 \leq i \leq 2048$. And for each pair of messages M_1^i and M_2^i, we computed the corresponding hash values h_1^i and h_2^i and stored them in ASCII format.

In the first experiment, for all pairs of hash values h_1^i and h_2^i, $1 \leq i \leq 2048$, we computed the experimental results of both the SHA-384 and the QGMD5-384 using Equation (6.7); and the expected results of an s-byte hash value using Equation (6.8). The expected and experimental results are given in Table 6.6 (a) and (b), respectively. In this table, v denotes the number of bytes that have the same value at the same position if we compare a pair of h_1^i and h_2^i as byte sequences, $1 \leq i \leq 2048$; s denotes the number of bytes in a hash value, and $W_N(v)$ denotes the number of times v hits in

N number of experiments. Smaller v characterizes the stronger hash function against collision resistance. If we compare the experimental results of SHA-384 and QGMD5-384 with the corresponding expected results, which are tabulated in Table 6.6 (a) and (b), It can be observed that the experimental results of the proposed QGMD5-384

Table 6.6: Results of expected and experimental.

	Expected result ($W_N(v)$)	Experimental result	
v	$s = 48$	SHA-384 ($s = 48$)	QGMD5-384 ($s = 48$)
0	1697.23	1676	1683
1	319.48	348	335
2	29.44	20	27
3	1.77	4	3
$v \geq 4$	0	0	0
	(a)	(b)	

not only coincide very well with the theoretical ones but also it has better collision resistance than that of SHA-384.

In the second experiment, we calculated the absolute difference (AD) between each pair of h_1^i and h_2^i using Equation (6.9), $1 \leq i \leq 2048$. In this case, the larger value of AD characterizes the stronger hash function against collision resistance. For each of the hash functions SHA-384 and QGMD5-384, the simulation of this experiment is also run 2048 times and calculated the minimum, maximum, mean, and mean/char of AD. The results of this experiment are given in Table 6.7. Note that the ideal value of mean/char, as defined in [17], is 85.33. According to the results as shown in Table 6.7, it can be observed that the obtained mean/char value of the proposed QGMD5-384 hash function is closer to the ideal value of 85.33 than that of the SHA-384. Hence, the second experiment also ensures that QGMD5-384 is more secure than SHA-384.

Table 6.7: Results of the absolute differences.

Hash Functions	Minimum	Maximum	Mean	Mean/Char
SHA-384	2821	5209	3913.43	81.52
QGMD5-384	2800	5395	4046.27	84.31

6.4.7.3 Prefix and suffix attacks

Prefix and suffix attacks are common methods for finding collisions in cryptographic hash functions. The details of these attacks against the QGMD5-224 hash function are described in section 6.3.7.2. For the prefix attack, an attacker creates a false message $PM = (p_1, p_2, \ldots, p_u, m_1, m_2, \ldots, m_k)$ by adding the prefix $P = (p_1, p_2, \ldots, p_u)$ to the original message $M = (m_1, m_2, \ldots, m_k)$ so that $H_{IV}(PM) = H_{IV}(M)$, where H is the hash function, $m_i \in Q$ for $1 \leq i \leq k$, $p_j \in Q$, for $1 \leq j \leq u$, and IV and Q denote an initial value and an employed quasigroup. This attack is successful only if $H_{IV}(P) = IV$. In other words, this attack can happen if a hash function is vulnerable to a pre-image attack. But the proposed QGMD5-384 hash function is resistant to pre-image attack. This is because it uses MD5 and an optimal quasigroup of order 16. Since MD5 is already resistant to the pre-image attack and the optimal quasigroup consists of 16 optimal S-boxes of 4×4. That is, the security of the QGMD5-384 is not only dependent on the MD5 but also on the 16 optimal S-boxes as an optimal quasigroup of order 16. Hence, the QGMD5-384 is resistant to prefix attack.

A similar argument can be used to show that the proposed QGMD5-384 hash function is resistant to the suffix attack. To mount this attack, first, an attacker chooses a suffix $S = (s_1, s_2, \ldots, s_t)$, $s_j \in Q$, $1 \leq j \leq t$, and tries to create a false message $MS = (m_1, m_2, \ldots, m_k, s_1, s_2, \ldots, s_t)$ by appending the suffix S to the original message M, so that $H_{IV}(MS) = h = H_{IV}(M)$. This attack is successful only if $H_h(S) = h$.

6.4.7.4 Avalanche effect

A hash function must exhibit a good avalanche effect since it is one of the desirable properties of a hash function. We have analyzed the proposed QGMD5-384 hash function against this test by comparing it with those of the existing MD5 and SHA-384 hash functions. For this test, we used the same process and inputs that were used to analyze the QGMD5-224 hash function. The details are described in section 6.3.7.3. The results of the MD5, SHA-384, and QGMD5-384 against this test are given in Table 6.8. In this table, HDP_i denotes the hamming distance between two hash values in percentage, which is obtained using Equation (6.10). The first column shows the range of hamming distances (HDP_i) in the specified range separately; the second, third, and fourth columns show the number of times the hamming distances (HDP_i) of the hash

values lie in the specified range given in the first column of the table corresponding to MD5, SHA-384, and QGMD5-384, respectively. Also given in the last row of the table is the average (mean) hamming distance in percentage. From these values, it can be observed that the avalanche effect of QGMD5-384 is better than that of both MD5 and SHA-384.

Table 6.8: Hamming distances for MD5, SHA-384 and QGMD5-384.

Range of HDP_i	Number of hash pairs of MD5	Number of hash pairs of SHA-384	Number of hash pairs of QGMD5-384
35 - 44.99	41	7	7
45 - 54.99	206	265	268
55 - 59.99	33	8	5
	Avarage hamming distance		
Mean:	49.76	49.97	50.12

6.4.7.5 Bit variance test

Bit variance test is one of the statistical tests to measure the impact on each bit of hash value by changing the bits of the input message. If there is a slight change in the input message, then the impact of this change on each bit of the corresponding hash value should be uniform. For this test, we used the same inputs that were used to analyze the QGMD5-224 hash function, given in section 6.3.7.4. The proposed QGMD5-384 hash function takes a variable length input message and produces a fixed-length 384 bits hash value. So, for each of these 384 bits, we calculate the probability of a bit being 1. Let $P_i(1)$ be the probability that the i^{th} bit of a hash value is 1. Similarly, let $P_i(0)$ be the probability that the i^{th} bit of a hash value is 0. If $P_i(1) = P_i(0) = \frac{1}{2}$ for all bits of the hash value, i.e. for $i = 1, 2, \ldots, 384$, then the QGMD5-384 passes the bit variance test. The obtained results are as follows:

$$\text{Number of hash values} = 281$$
$$\text{Mean frequency of 1s (expected)} = 140.5$$
$$\text{Mean frequency of 1s (calculated)} = 140.3$$

According to these results, it can be observed that the average probability of $1s \approx 50\%$. Hence, QGMD5-384 passes the bit variance test.

6.4.7.6 Analysis of QGMAC-384

The proposed message authentication code QGMAC-384 is designed based on the hash function QGMD5-384 and uses a secret quasigroup of order 16 or 256, depending on the algorithm. So, the security of the QGMAC-384 depends on both the QGMD5-384 and the quasigroup of order 16 or 256 that is used. Since the number of quasigroups of orders 16 and 256 is upper bounded by 0.689×10^{138} and 0.753×10^{102805}, respectively, the use of order 256 quasigroups will provide more security than the quasigroups of order 16. But to store a quasigroup of order 256 is required 65280 bytes extra space than that of a quasigroup of order 16.

As of today, a cryptosystem with at least a keyspace of 2^{128} is considered to be a secure cryptosystem. Therefore, the use of order 16 quasigroups also provides good security in QGMAC-384. This is because the number of quasigroups of order 16 is $0.689 \times 10^{138} \approx 2^{456}$. That is, the use of order 256 quasigroups can be an alternative option. Because of the large number of quasigroups of either order, it follows that the probability of identifying the employed quasigroup is close to zero. Hence, QGMAC-384 is resistant to brute force attack. Also, QGMAC-384 is analyzed against forgery attack and found to be resistant. In this attack, an attacker chooses a fixed number, say n, of different messages $(M_1, M_2, \ldots, M_n)$ and their corresponding MAC values (authentication tags) $(h_1, h_2, \ldots, h_n)$ and tries to solve the following equations for the secret key k :

$$h_i = H_k(M_i),\ 1 \leq i \leq n \tag{6.12}$$

where, H is the QGMD5-384 hash function and k is the quasigroup employed. If the attacker has knowledge of the secret key k, then the attacker can forge an authentication tag for any chosen message. But here, k is secret. So, the above system of equations has as many solutions as there are quasigroups of order 16 or 256. Hence determining the quasigroup makes it practically impossible. Therefore, the QGMAC-384 is also resistant to forgery attack.

6.5 Summary

This chapter proposed two hash functions and, based on them, proposed the corresponding message authentication codes. Each of these schemes is designed based on a

quasigroup. The first hash function (QGMD5-224) and message authentication code (QGMAC-224) produce 224 bits hash value and MAC value, respectively. They are discussed in section 6.3. The second hash function (QGMD5-384) and message authentication code (QGMAC-384) produce 384 bits hash value and MAC value, respectively. They are discussed in section 6.4. Both the hash functions and the message authentication codes can be viewed as the extended version of MD5, and they use the MD5 along with a quasigroup of order 16 or 256, depending on the algorithms. The proposed schemes mask the weaknesses of MD5 and adds extra security. We have analyzed both the QGMD5-224 and the QGMD5-384 against various attacks, including brute force attack, collision resistance, and prefix and suffix attacks, and found that they are resistant to these attacks. The randomness properties of the proposed hash functions are analyzed using the avalanche effect and bit variance test, and they satisfied all the properties that are needed for the ideal hash functions.

We compared our schemes with the existing quasigroup-based hash functions that are discussed in the literature survey of Chapter 3 and are introduced in the literature [23, 24, 70]. We found that the proposed hash functions (QGMD5-224 and QGMD5-384) are resistant to prefix and suffix attacks [69]; while the existing hash functions [23, 24, 70] are vulnerable to these attacks. Hence, It can be concluded that the new quasigroup based hash functions (QGMD5-224 and QGMD5-384) appear to be a good alternative to the existing quasigroup based hash functions.

Also, the proposed QGMAC-224 and QGMAC-384 can be viewed as an extended version of HMAC-MD5. They use a quasigroup of order 16 or 256 as a secret key to calculate the MAC values. Since the number of quasigroups of order 16 or 256 is practically infinite, it is computationally infeasible to determine the employed quasigroup. Hence the QGMAC-224 and QGMAC-384 are resistant to brute force attacks. Also, they are analyzed against forgery attacks, and found that both of them are resistant to forgery attacks as well.

The software performance of both QGMD5-224 and QGMAC-224 is compared to that of the corresponding algorithms of MD5, SHA-224, HMAC-MD5, and HMAC-SHA-224. Similarly, the software performance of both QGMD5-384 and QGMAC-384 is compared to that of the corresponding algorithms of MD5, SHA-384, HMAC-MD5, and HMAC-SHA-384. We found that QGMD5-224 is slightly slower than MD5 but

faster that SHA-224. And QGMAC-224 is found to be always faster than both HMAC-MD5 and HMAC-SHA-224. On the other hand, we found that QGMD5-384 is slightly slower than MD5 but faster than SHA-384. And QGMAC-384 is slightly slower than HMAC-MD5 but faster than HMAC-SHA-384.

Chapter 7

Conclusions and Future work

7.1 Conclusions

The major contribution of this thesis is the design of new cryptosystems based on quasigroup. Contributions are broadly categorized into three parts. In the first part three variants of stream ciphers, in the second part two variants of block ciphers, and in the third part two variants of hash functions and HMACs are proposed. All these proposals are based on the concept of quasigroup.

The three proposed stream ciphers are described in Chapter 4. The first cipher discussed in section 4.3, uses AES-256 for generating the keystream and a quasigroup of order 256 for encrypting/decrypting the messages. The second cipher discussed in section 4.4, uses QG-PRNG for generating the keystream and a quasigroup of order 256 for encrypting/decrypting the messages. The QG-PRNG is a quasigroup based pseudo-random number generator described in section 4.4.1. Note that the second cipher is the revised version of the first cipher. This is because the second cipher uses QG-PRNG for generating the keystream instead of AES-256 as used in the first cipher, thereby the second cipher is more efficient than the first cipher. The third cipher is discussed in section 4.5. It uses MQG-PRNG for generating the keystream and 16 quasigroups of order 16 for encrypting/decrypting the messages. The MQG-PRNG is a multiple quasigroups-based pseudo-random number generator, uses 16 quasigroups, and it is described in section 4.4.2. The algorithms of both the MQG-PRNG and the encryption/decryption use the same set of 16 quasigroups but may be in different orders (permutations). These 16 quasigroups are dynamically generated based on an original

quasigroup of order 16. The use of multiple quasigroups contributes to increased security since a different quasigroup is used after a certain amount of time. Note that the third cipher is the revised version of both the first and second ciphers. This is because, the third cipher uses a quasigroup of order 16 instead of a quasigroup of order 256 as used in both the first and second ciphers, thereby the third cipher needs around 99% lesser space than both the first and second ciphers. The novelty of the proposed stream ciphers is that they are resistant to the reused key attack as against the existing XOR-based stream ciphers. Hence a keystream can be reused multiple times, thereby overcoming the major hurdle that exists in the application of the stream ciphers. The proposed ciphers are analyzed against the most common attacks, including the chosen-ciphertext attack, the chosen-plaintext attack, the known-plaintext attack, the reused-key attack, and the time-memory-data tradeoff (TMDTO) attack. We observed that our ciphers are resistant to these attacks. Also, the performance of the proposed ciphers is analyzed by comparing them to some of the existing quasigroup based stream ciphers [12, 28, 43, 59, 60, 81]. We observed that in most cases the proposed ciphers are more efficient than the existing quasigroup based proposals. The randomness of the obtained ciphertexts produced by the proposed stream ciphers is analyzed using the NIST-STS test suite. We found that the obtained ciphertexts of the proposed ciphers are highly random.

The proposed two block ciphers to encrypt or decrypt messages in the form of a block of 128 bits are described in Chapter 5. Both the ciphers are designed based on the Permutation Substitution Network (PSN) and use 16 optimal S-boxes as an optimal quasigroup of order 16, where the size of each S-box is 4×4 bits. The design of the proposed ciphers is based on the key-dependent S-box, where the operations are carried out using the quasigroup operation. In each quasigroup operation, a key-dependent S-box layer chooses one S-box out of the 16 S-boxes, where the choice is based on the round key or sub-key. We believe that key-dependent S-box ciphers are more secure than fixed S-box ciphers. This is because key-dependent S-boxes do not offer any specific properties to the cryptanalyst. Most key-dependent S-box ciphers are effectively random. Examples of such ciphers are Blowfish [66] and SEAL [15]. The second cipher discussed in section 5.4 is more standard than the first cipher discussed in section 5.3. This is because each round of the second cipher uses three transformations (substitution, permutation, and add round key), while the first cipher uses only two

transformations (substitution and permutation). The proposed ciphers are analyzed against several attacks, including linear cryptanalysis and differential cryptanalysis, and found that the ciphers are resistant to these attacks. Also, we have analyzed the software performance (time complexity), space complexity, and avalanche effect (diffusion effect) of the proposed ciphers by comparing them with AES-128 and other existing quasigroup based block ciphers [5, 6, 83]. We noted that the avalanche effect of our ciphers compares to that of the AES-128 and due to more computations our ciphers are slightly slower than AES-128, but our cipher uses half the space compared to AES-128. Also, the proposed block ciphers use the same amount of space as that used by [83] but 512 times lesser than [5, 6]. We also noted that our ciphers are more efficient than DES. In addition, our ciphers are more than 2 times faster and give a better avalanche effect than other existing quasigroup based block ciphers [5, 6, 83]. Hence, we concluded that the proposed ciphers appear to be an excellent alternative to the quasigroup based proposals. The randomness of the obtained ciphertexts produced by the proposed ciphers is tested using the NIST statistical test suite. We ran our encryption systems for a random plaintext of 1048576 bits with 1000 different keys and generated 1000 ciphertexts. The results are compared with that of AES-128 for the same plaintext and the same keys. We observed that the randomness of the outputs of our ciphers and AES-128 are comparable to each other.

In Chapter 6, we proposed two hash functions and, based on them we also proposed, the corresponding message authentication codes (HMACs). Each of these schemes is designed using the quasigroup. The first hash function, named QGMD5-224, and the corresponding message authentication code, named QGMAC-224 produce 224 bits hash value and MAC value, respectively. They are described in section 6.3. The second hash function, named QGMD5-384, and the corresponding message authentication code, named QGMAC-384 produce 384 bits hash value and MAC value, respectively. They are described in section 6.4. The proposed hash functions (QGMD5-224 and QGMD5-384) can be seen as the extended version of MD5. They are designed using the MD5 and a quasigroup of order either 16 or 256. They mask the weaknesses of MD5 and add extra security. We have analyzed both the QGMD5-224 hash function and the QGMD5-384 hash function against several attacks, including brute force attack, collision resistance, and prefix and suffix attacks, and found that the proposed hash functions are resistant to these attacks. The randomness properties of the proposed hash functions are

analyzed using the avalanche effect and bit variance test, and they satisfied all the properties that are needed for the ideal hash functions.

The proposed HMACs (QGMAC-224 and QGMAC-384) can be seen as an extended version of HMAC-MD5. They use a quasigroup of order either 16 or 256 as a secret key and calculate the MAC values. Since the number of quasigroups of order 16 or 256 is practically infinite, it is impossible to determine the employed quasigroup. Hence the QGMAC-224 and QGMAC-384 are resistant to brute force attack. Also, they are analyzed against forgery attacks and found that both are resistant to forgery attacks as well.

The software performance of both QGMD5-224 and QGMAC-224 is compared to that of the corresponding algorithms of MD5, SHA-224, HMAC-MD5, and HMAC-SHA-224. Similarly, the software performance of both QGMD5-384 and QGMAC-384 is compared to that of the corresponding algorithms of MD5, SHA-384, HMAC-MD5, and HMAC-SHA-384. We found that QGMD5-224 is slightly slower than MD5 but faster than SHA-224. And QGMAC-224 is found to be always faster than both HMAC-MD5 and HMAC-SHA-224. On the other hand, we found that QGMD5-384 is slightly slower than MD5 but faster than SHA-384. And QGMAC-384 is slightly slower than HMAC-MD5 but faster than HMAC-SHA-384. Also, we observed that the performance of the proposed hash functions and their corresponding HMACs are the same. This is because the underlying structure of both the proposed hash functions and their corresponding HMACs is similar. The only difference between the two is that the quasigroup used in the hash functions is publicly known, while the quasigroup used in HMACs acts as a secret key.

7.2 Future work

In this thesis, we have designed several cryptosystems based on quasigroups. Each of the proposed cryptosystems comes under the category of symmetric key cryptography. So, one of our intentions is to design public key cryptosystems based on quasigroup. In addition, we have the following intentions:

- To analyze the proposed stream ciphers against several attacks such as Algebraic attack, Correlation attack, Fault attack, Guess and determine attack, etc.

- To analyze the proposed block ciphers against more cryptanalytic attacks such as Boomerang attack, Meet-in-the-middle attack, Key recovery attack, Impossible differential attack, etc.
- To analyze the proposed keystream generation algorithms (QG-PRNG and MQG-PRNG) against several attacks such as Related-key attack, Slide attack, etc.
- To analyze the proposed hash functions against several attacks such as Rainbow table attack, Side-channel attack, Length extension attack, etc.

Last but not the least, we would like to extend the proposed hash functions and their corresponding HMACs to produce 512 or 728 bits hash and MAC values. We also would like to explore the possibility of designing other crypto-primitives such as digital signature and authentication systems using the proposed hash functions and HMACs.

References

[1] Alter, R. **How many Latin squares are there?**. *The American Mathematical Monthly*, **82**(6), pp.632-634, 1975.

[2] Babbage, S. H. **Improved exhaustive search attacks on stream ciphers**. *In European Convention on Security and Detection*, IET:161–166, 1995.

[3] Banik, S., Pandey, S. K., Peyrin, T., Sasaki, Y., Sim, S. M. and Todo, Y. **GIFT: A small present: Towards reaching the limit of lightweight encryption**. *In Cryptographic Hardware and Embedded Systems–CHES 2017: 19th International Conference Proceedings*, pp. 321-345. Springer International Publishing, 2017.

[4] Barker E. B. and Kelsey J. M. **Recommendation for random number generation using deterministic random bit generators**. *Technical report, NIST, (revised)*, 2007.

[5] Battey, M. and Parakh, A. **Efficient quasigroup block cipher for sensor networks**. *In 21st international conference on computer communications and networks*, pp. 1-5, IEEE, 2012.

[6] Battey, M. and Parakh, A. **An efficient quasigroup block cipher**. *Wireless personal communications*, **73**(1), pp.63-76, 2013.

[7] Bayer, R. and Metzger, J. K. **On the encipherment of search trees and random access files**. *ACM Transactions on Database Systems*, **1**(1), pp. 37-52, 1976.

[8] Bellare, M., Canetti, R., and Krawczyk, H. **Keying hash functions for message authentication**. *In Advances in Cryptology—CRYPTO'96: 16th Annual International Cryptology Conference Proceedings*, pp. 1-15, Springer, 1996.

[9] Biham E. and Shamir A. **Differential cryptanalysis of the data encryption standard**. *Springer Science Business Media*, 2012.

[10] Biryukov, A. and Shamir, A. **Cryptanalytic time/memory/data tradeoffs for stream ciphers**. 6^{th} *International Conference on the Theory and Application of Cryptology and Information Security*, pp. 1-13, Springer, 2000.

[11] Biryukov, A., Shamir, A., and Wagner, D. **Real time cryptanalysis of A5/1 on a PC**. 7^{th} *International Workshop Proceedings*, pp. 1-18, Springer, 2001.

[12] Chakrabarti, S., Pal, S. K., and Gangopadhyay, S. **An improved 3-quasigroup based encryption scheme**. *ICT Innovations Web Proceedings ICT*, pp. 173-184, 2012.

[13] Chew, L. C. N., Shah, I. N. M., Abdullah, N. A. N., Zawawi, N. H. A., Rani, H. A., and Zakaria, A. A. **Randomness analysis on Speck family of lightweight block cipher**. *International Journal of Cryptology Research*, **5**(1), pp. 44-60, 2015.

[14] Comtet, L., D´enes J., A. D. Keedwell, R. A. Fisher, and et. al. **Sequence #A000315: Number of reduced Latin squares of order n; also number of labeled loops with a fixed identity element**. *The On-Line Encyclopedia of Integer Sequences.*

[15] Coppersmith, D. and Rogaway, P.W. **International Business Machines Corp.** *Software-efficient pseudorandom function and the use thereof for encryption. U.S.* Patent 5,454,039, 1995.

[16] Dénes, J., and Keedwell, A.D. **Latin squares: New developments in the theory and applications**, Vol. 46, Elsevier, 1991.

[17] Deng, S., Li, Y. and Xiao, D. **Analysis and improvement of a chaos-based Hash function construction**. *Communications in Nonlinear Science and Numerical Simulation*, **15**(5), pp. 1338-1347, 2010.

[18] Diffie, W. and Hellman, M. E. **Special feature exhaustive cryptanalysis of the NBS data encryption standard**. *Computer*, **10**(6), pp. 74-84, 1977.

[19] Dimitrova, V., and Markovski, J. **On quasigroup pseudo random sequence generator**. *In Proc. of the* 1^{st} *Balkan Conference in Informatics,*, pp. 21-23, 2004.

[20] Dimitrova, V., Kostadinoski, M., Trajcheska, Z., Petkovska, M. and Buhov, D. **Some Cryptanalysis of the Block Cipher BCMPQ**. *ICT Innovations 2014 Web Proceedings*, pp. 201-209, 2014.

[21] Dimitrova, V., Markovski, S. and Gligoroski, D. **Classification of quasigroups as boolean functions, their algebraic complexity and application of gröbner bases in solving systems of quasigroup equations**. *RICS book series Springer-Groebner bases, Coding and Cryptography*, 2007.

[22] Dunkelman, O. and Keller, N. **Treatment of the initial value in time-memory-data trade- off attacks on stream ciphers**. *Information Processing Letters*, **107**(5), pp. 133-137, 2008.

[23] Dvorsky, J., Ochodkova, E., and Snasel, V. **Hash functions based on quasigroups**. *Proc. Milkulasska kryptobesidka, Praha (in Czech)*, pp. 27-36, 2001.

[24] Dvorsky, J., Ochodkova, E., and Snasel, V. **Hash functions based on large quasigroups**. *(Czech), Proc. of Velikonocni kryptologie, Brno*, pp. 1–8, 2002.

[25] Ekdahl, P., Johansson, T., Maximov, A. and Yang, J. **A new SNOW stream cipher called SNOW-V**. *Cryptology ePrint Archive*, 2018.

[26] Gligoroski, D., Markovski, S., and Kocarev, L. **Edon-R, An Infinite Family of Cryptographic Hash Functions**. *International Journal of Netw. Secur*, **8**(3), pp. 293-300, 2006.

[27] Gligoroski, D., and Knapskog, S. J. **Edon-R(256,384,512)- An efficient implementa- tion of Edon-R family of cryptographic hash functions**. *Commentationes Mathematicae Universitatis Carolinae*, **49**(2), pp. 219-39, 2008.

[28] Gligoroski, D., Markovski, S., and Knapskog, S. J. **The stream cipher Edon80**. *New stream cipher designs: The eSTREAM finalists*, pp. 152-169, 2008.

[29] Gligoroski, D. and Markovski, S. **Cryptographic potentials of quasigroup transformations Category**.

[30] Golic., J. D. **Cryptanalysis of alleged A5 stream cipher**. *In International Conference on the Theory and Applications of Cryptographic Techniques*, pp. 239-255, Springer, 1997.

[31] Grover, L. K. **A fast quantum mechanical algorithm for database search**. *In Proceedings of the twenty-eighth annual ACM symposium on Theory of computing*, pp. 212-219, 1996.

[32] Heys, H. M. **A tutorial on linear and differential cryptanalysis**. *Cryptologia*, **26**(3), pp. 189–221, 2002.

[33] Ilaiyaraja, M., BalaMurugan, P. and Jayamala, R. **Securing Cloud Data using Cryptography with Alert System**. *International Journal of Engineering Research*, **3**(3), pp. 98-101, 2014.

[34] Van Lint, J. H., Wilson, R. M., and Wilson, R. M. **A course in combinatorics**. *Cambridge university press*, 2001.

[35] Jiao, S., Lei, T., Gao, Y., Xie, Z., and Yuan, X. **Known-plaintext attack and ciphertext-only attack for encrypted single-pixel imaging**. *IEEE Access*, Vol. 7, pp. 119557-119565.

[36] Hell, M. and Johansson, T. **A key recovery attack on edon80**. 13^{th} *International Conference on the Theory and Application of Cryptology and Information Security*, pp. 568-581, Springer, 2007.

[37] Joux, A. **Multicollisions in iterated hash functions. Application to cascaded constructions**. 24^{th} *Annual International Cryptology Conference Proceedings*, Vol. 24, pp. 306-316, Springer, 2004.

[38] Josuttis, N. M. **The C++ standard library: a tutorial and reference**, 2^{nd} Edition, 2012.

[39] Khovratovich, D., Nikolic, I. and Weinmann, R. P. **Cryptanalysis of Edon-R**. *Retrieved*, 2013.

[40] Kelsey, J. and Schneier, B. **Second preimages on n-bit hash functions for much less than 2^n work**. 24^{th} *Annual International Conference on the Theory and*

Applications of Cryptographic Techniques Proceedings, Vol 24 pp. 474-490, Springer, 2005.

[41] Koscielny, C. **A method of constructing quasigroup-based stream-ciphers**. *Applied mathematics and computer science*, Val. 6, pp. 109-122, 1996.

[42] Koscielny, C. **Generating quasigroups for cryptographic applications**. *International Journal of Applied Mathematics and Computer Science*, **12**(4), pp. 559-570, 2002.

[43] Lakshmi, S., Srinivasan, C., Lakshmy, K.V. and Sindhu, M. **A quasigroup based synchronous stream cipher for lightweight applications**. *International Symposium on Security in Computing and Communication*, Springer, pp. 205–214, 2017.

[44] Leander, G. and Poschmann, A. **On the classification of 4 bit S-Boxes**. *In proceedings of the* 1^{st} *international workshop on arithmetic of finite fields*, pp. 159-176, Springer, 2007.

[45] Malyutina N. N. **Cryptanalysis of some stream ciphers**. *Quasigroups and related systems*, pp. 281-292, 2019.

[46] Markovski S., Gligoroski D. and Andova S. **Using quasigroups for one-one secure encoding**. *In: Proc. VIII Conf. logic and computer science LIRA*, Vol. 97, pp. 157-162, CiteSeer, 1997.

[47] Markovski, S. and A. Mileva. **NaSHA**. *Submission to NIST*, 2008.

[48] Markovski, S. and Mileva, A. **Nasha - cryptographic hash functions**. *NIST The First SHA-3 Candidate Conference*, Leuven, pp. 25–28, 2009.

[49] Markovski, S., Gligoroski, D. and Bakeva, V. **On infinite class of strongly collision resistant hash functions *Edon-F* with variable length of output**. 1^{st} *International Conference on Mathematics and Informatics for Industry*, Thessaloniki, pp. 302–308, 2003.

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

[51] Martin E. and Hellman, A. **Cryptanalytic Time-Memory Tradeoff**. *IEEE Transactions on Information Theory*, **26**(4), pp. 401–406, 1980.

[52] Matsui M. **Linear cryptanalysis method for DES cipher**. *Advances in cryptology, Proc. Eurocrypt'93*, LNCS 765, Springer pp. 386-397, 1994.

[53] McKay, B. D. and Wanless, I. M. **On the number of Latin squares**. **Annals of combinatorics**, **9**(3), pp. 335-344, 2005

[54] Meyer, K. A. **A new message authentication code based on the non-associativity of quasigroups**. *Iowa State University*, 2006.

[55] Mihajloska, H. and Gligoroski, D. **Construction of Optimal 4-bit S-boxes by Quasigroups of Order 4**. *In the sixth international conference on emerging security information, systems and technologies*, pp. 163-168, 2012.

[56] Mohan, M., Devi, M. K. K., Prakash, V.J. **Security analysis and modification of classical encryption scheme**. *Indian journal of science and technology*, **8**(8), pp. 542-548, 2015.

[57] Nikolic, I. and Knovratovich, D. **Free–start attacks on NaSHA**, 2008.

[58] Paar, C. and Pelzl, J. **Understanding cryptography: a textbook for students and practitioners**. Springer Science & Business Media, 2009.

[59] Petrescu, A. **A 3-quasigroup stream cipher**. *International Conference Interdisciplinarity in Engineering INTER-ENG*, Elsevier, p. 168, 2009.

[60] Petrescu A. **n-quasigroup cryptographic primitives: stream ciphers**. *Stud. Univ. Babeş-Bolyai inform.* **55**, [On table of contents: Anul LIV], pp. 27-34, 2010.

[61] Pickover, C. A., Ryser, H. J., Sloane, J. A. and et. al. **Sequence #A002860: Number of latin squares of order n; or labeled quasigroups**. The On-Line Encyclopedia of Integer Sequences.

[62] Pindar, Z., Jamel, S.H., Disina, A. and Deris, M.M. **Compression function based on permutations quasigroups**. *ARPN Journal of Engineering and Applied Sciences*, **11**(12), pp. 1-8, 2015.

[63] Parakh, A. and Kak, S. **Online data storage using implicit security**. *Information sciences*, **179**(19), pp. 3323–3331, 2009.

[64] Rijmen V. and Daemen J. **Advanced encryption standard**. *In: Proc. of federal information processing standards publications*, NIST, pp 19-22, 2001.

[65] Rukhin, A., Soto, J., Nechvatal, J., Smid, M. and Barker E. **A statistical test suite for random and pseudorandom number generators for cryptographic applications**, NIST, special publication, pp. 800-22, revision 1a, 2001.

[66] Schneier, B. **Description of a new variable-length key, 64-bit block cipher (Blowfish)**. *In International Workshop on Fast Software Encryption*, pp. 191-204, Springer, 1993.

[67] Schneier, B. **Applied cryptography protocols, algorithms, and source code in C**, John Wiley & Sons, 2007.

[68] Shcherbacov, V. **Elements of quasigroup theory and some its applications in code theory and cryptology**. *Lectures in Prague, Czech Republic.*

[69] Slaminkova, I. and Vojvoda, M. **Cryptanalysis of a hash function based on isotopy of quasigroups**. *Tatra Mountains Mathematical Publications*, **45**(1), pp. 137-149, 2010.

[70] Snasel, V., Abraham A., Dvorsy J., Kromer P. and Platos J. **Hash function based on large quasigroups**. LNCS 5544, pp. 521-529, 2009.

[71] Soto, J. and Bassham, L. **Randomness testing of the advanced encryption standard finalist candidates**. Report md20899-8930, NIST, Gaithersburg, 2000.

[72] Stallings W. **Cryptography and network security**. 4^{th} ed. Pearson education, Inc. India, 2006.

[73] Stevens, M. **MASTER'S THESIS, On collisions for MD5**, 2007.

[74] Stinson D. **Cryptography: Theory and practice.** *CRC Press, CRC Press LLC*, 1995.

[75] Stones, R.J., Su, M., Liu, X., Wang, G. and Lin, S. **A Latin square autotopism secret sharing scheme**. *Designs, Codes and Cryptography*, **80**(3), pp.635-650, 2016.

[76] Supercomputer, F. **A64FX 48C 2.2GHz, Tofu interconnect D, Fujitsu RIKEN Center for Computational Science Japan**, `https://www.top500.org/lists/top500/2021/11/`, 2021.

[77] Teseleanu, G. **Quasigroups and substitution permutation networks: a failed experiment**. *Cryptologia*, **45**(3), pp. 266-281, 2020.

[78] Turner, J. M. **The keyed-hash message authentication code**. *Federal Information Processing Standards Publication*, **198**(1), pp. 1-13, 2008.

[79] Velammal, D.S.G. and Arockiadoss, T. **Modified method of generating randomized Latin squares**. *IOSR Journal of Computer Engineering*, Vol. 16, pp. 76-80, 2014.

[80] Vojvoda M. **Stream ciphers and hash functions-analysis of some new design approaches**. *PhD thesis, Slovak University of technology*, 2004.

[81] Zhang, J. and Xu, Y. **Stream cipher based on quasigroups with keystream period of arbitrary length**. *International Conference on Computer Science and Service System*, IEEE, pp. 834–836, 2012.

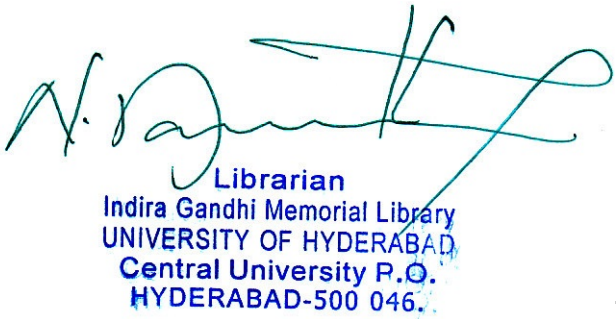
[82] Zhang J., Wang X. and Zhang W. **Chaotic keyed hash function based on feedforward–feedback nonlinear digital filter**. *Physics Letters A*, **362**(5-6), pp. 439-448, 2007.

[83] Zhao, Y. and Xu, Y. **A Lightweight block cipher based on quasigroups**. *In* 6^{th} *International Conference on Advanced Materials and Computer Science*, 2017a.

[84] Zhao, Y. and Xu, Y. **Optimal S-boxes based on 3-quasigroups of order 4**. *In* 6^{th} *International Conference on Advanced Materials and Computer Science*, 2017b.

Design and Analysis of Cryptographic Primitives based on Quasigroup

by Umesh Kumar



Submission date: 24-May-2023 04:54PM (UTC+0530)
Submission ID: 2100773401
File name: Umesh_Kumar.pdf (2.49M)
Word count: 58211
Character count: 269119

Design and Analysis of Cryptographic Primitives based on Quasigroup

ORIGINALITY REPORT

36%	33%	22%	3%
SIMILARITY INDEX	INTERNET SOURCES	PUBLICATIONS	STUDENT PAPERS

PRIMARY SOURCES

overall similarity index = 36 − (17 + 7 + 4 + 1) = 07%

1	eprint.iacr.org Internet Source	17%
2	www.igi-global.com Internet Source	7%
3	www.researchgate.net Internet Source	4%
4	Umesh Kumar, V. Ch. Venkaiah. "An Efficient Message Authentication Code Based on Modified MD5-384 Bits Hash Function and Quasigroup", International Journal of Cloud Applications and Computing, 2022 Publication	1%
5	Dimpy Chauhan, Indivar Gupta, Rashmi Verma. "Quasigroups and their applications in cryptography", Cryptologia, 2020 Publication	1%
6	en.wikipedia.org Internet Source	<1%

This publication is by the student.

Professor
School of CIS
Prof. C.R. Rao Road,
Central University
Hyderabad-46. (India)

25/5/2023

7	Umesh Kumar, Aayush Agarwal, V. Ch. Venkaiah. "Chapter 8 New Symmetric Key Cipher Based onQuasigroup", Springer Science and Business Media LLC, 2022 Publication	<1 %
8	dokumen.pub Internet Source	<1 %
9	acikbilim.yok.gov.tr Internet Source	<1 %
10	digitalcommons.unomaha.edu Internet Source	<1 %
11	www.yumpu.com Internet Source	<1 %
12	Lecture Notes in Computer Science, 2015. Publication	<1 %
13	faculty.ist.unomaha.edu Internet Source	<1 %
14	Rhouma, R.. "Cryptanalysis of a chaos-based cryptosystem on DSP", Communications in Nonlinear Science and Numerical Simulation, 201102 Publication	<1 %
15	Lecture Notes in Computer Science, 2012. Publication	<1 %

16	Umesh Kumar, V. Ch. Venkaiah. "Chapter 1 A New Modified MD5-224 Bits Hash Function andanEfficient Message Authentication Code Based onQuasigroups", Springer Science and Business Media LLC, 2022 Publication	<1 %
17	Jun Zheng, Hanping Hu. "A highly secure stream cipher based on analog-digital hybrid chaotic system", Information Sciences, 2022 Publication	<1 %
18	proceedings.ictinnovations.org Internet Source	<1 %
19	Lecture Notes in Computer Science, 2011. Publication	<1 %
20	Sijia Li, Zhiyi Liao, Zhengyang Wu, Zheng Wu, Lin Ding. "(Quantum) Time-Memory-Data Tradeoff Attacks on the SNOW-V Stream Cipher", Symmetry, 2022 Publication	<1 %
21	archive.org Internet Source	<1 %
22	www.clausiuspress.com Internet Source	<1 %
23	Submitted to University of Glamorgan Student Paper	<1 %

24	Submitted to Symbiosis International University Student Paper	<1%
25	Arpita Sarkar, Binod Kumar Singh. "A cancelable fingerprint biometric based session key establishment protocol", Multimedia Tools and Applications, 2019 Publication	<1%
26	Ramesh Karri. "Power optimization for universal hash function data path using divide-and-concatenate technique", Proceedings of the 3rd IEEE/ACM/IFIP international conference on Hardware/software codesign and system synthesis - CODES+ISSS 05 CODES+ISSS 05, 2005 Publication	<1%
27	ebin.pub Internet Source	<1%
28	Lecture Notes in Computer Science, 2009. Publication	<1%
29	etd.lib.metu.edu.tr Internet Source	<1%
30	"Security, Privacy, and Applied Cryptography Engineering", Springer Science and Business Media LLC, 2014 Publication	<1%

31	repositorio.uniandes.edu.co Internet Source	<1 %
32	ipfs.io Internet Source	<1 %
33	www.jestr.org Internet Source	<1 %
34	Lecture Notes in Computer Science, 2016. Publication	<1 %
35	Submitted to La Trobe University Student Paper	<1 %
36	bnewtoncis185.blogspot.com Internet Source	<1 %
37	docplayer.pl Internet Source	<1 %
38	www3.iam.metu.edu.tr Internet Source	<1 %
39	"Advances in Cryptology – ASIACRYPT 2007", Springer Nature, 2007 Publication	<1 %
40	www.hooklee.com Internet Source	<1 %
41	Lecture Notes in Computer Science, 2014. Publication	<1 %
42	www.mdpi.com Internet Source	

		<1%
43	pdffox.com Internet Source	<1%
44	Submitted to Al-Nahrain University Student Paper	<1%
45	Lecture Notes in Computer Science, 2001. Publication	<1%
46	link.springer.com Internet Source	<1%
47	www.securitywatch.com Internet Source	<1%
48	D.S.L. Wei, F.P. Muga, K. Naik. "Isomorphism of degree four Cayley graph and wrapped butterfly and their optimal permutation routing algorithm", IEEE Transactions on Parallel and Distributed Systems, 1999 Publication	<1%
49	Menezes, Alfred, and Paul van Oorschot. "Coding Theory And Cryptology", Discrete Mathematics and Its Applications, 1999. Publication	<1%
50	www.ijera.com Internet Source	<1%

51	Sachin Kumar. "Image data security using Quasigroup combined with Fibonacci Q-transformation", Journal of Information Security and Applications, 2021 Publication	<1 %
52	www.quasigroups.eu Internet Source	<1 %
53	"New Stream Cipher Designs", Springer Nature, 2008 Publication	<1 %
54	Chao-Jung Cheng, Chi-Bin Cheng. "An asymmetric image cryptosystem based on the adaptive synchronization of an uncertain unified chaotic system and a cellular neural network", Communications in Nonlinear Science and Numerical Simulation, 2013 Publication	<1 %
55	speakerdeck.com Internet Source	<1 %
56	"Fast Software Encryption", Springer Science and Business Media LLC, 2006 Publication	<1 %
57	Vaclav Snasel. "Searching for quasigroups for hash functions with genetic algorithms", 2009 World Congress on Nature & Biologically Inspired Computing (NaBIC), 12/2009 Publication	<1 %

58	core.ac.uk Internet Source	<1%
59	Aleksandra Mileva. "chapter 12 New Developments in Quasigroup-Based Cryptography", IGI Global, 2014 Publication	<1%
60	Submitted to Queen's University of Belfast Student Paper	<1%
61	edoc.pub Internet Source	<1%
62	"Gröbner Bases, Coding, and Cryptography", Springer Science and Business Media LLC, 2009 Publication	<1%
63	Submitted to Arab Open University Student Paper	<1%
64	"Applied Cryptography and Network Security", Springer Science and Business Media LLC, 2009 Publication	<1%
65	Dario Ghersi, Abhishek Parakh, Mihaly Mezei. "Comparison of a quantum random number generator with pseudorandom number generators for their use in molecular Monte Carlo simulations", Journal of Computational Chemistry, 2017 Publication	<1%

66	Jinlu Liu, Jie Yang, Zhengyu Li, Qi Su, Wei Huang, Bingjie Xu, Hong Guo. "117 Gbits/s Quantum Random Number Generation With Simple Structure", IEEE Photonics Technology Letters, 2017 Publication	<1 %
67	Submitted to National Institute of Technology Karnataka Surathkal Student Paper	<1 %
68	Submitted to Nottingham Trent University Student Paper	<1 %
69	Submitted to Sheffield Hallam University Student Paper	<1 %
70	Submitted to Universiti Kebangsaan Malaysia Student Paper	<1 %
71	es.scribd.com Internet Source	<1 %
72	www.emsec.rub.de Internet Source	<1 %
73	Menezes, . "Stream Ciphers", Discrete Mathematics and Its Applications, 1996. Publication	<1 %
74	N. K. Sreelaja, N. K. Sreeja. "An image edge based approach for image password encryption", Security and Communication Networks, 2016	<1 %

Publication

75	Understanding Cryptography, 2010. Publication	<1 %
76	Václav Snášel, Jiří Dvorský, Eliška Ochodková, Pavel Krömer, Jan Platoš, Ajith Abraham. "Chapter 39 Genetic Algorithms Evolving Quasigroups with Good Pseudorandom Properties", Springer Science and Business Media LLC, 2010 Publication	<1 %
77	lib.dr.iastate.edu Internet Source	<1 %
78	secowinet.epfl.ch Internet Source	<1 %
79	uwspace.uwaterloo.ca Internet Source	<1 %
80	www.cisco.com Internet Source	<1 %
81	"A Lightweight block cipher based on quasigroups", 2017 6th International Conference on Advanced Materials and Computer Science (ICAMCS 2017), 2017 Publication	<1 %
82	"Advances in Networks and Communications", Springer Science and Business Media LLC, 2011 Publication	<1 %

83	Bouchra Echandouri, Fouzia Omary, Fatima Ezzahra Ziani, Anas Sadak. "SEC-CMAC A New Message Authentication Code Based on the Symmetrical Evolutionist Ciphering Algorithm", International Journal of Information Security and Privacy, 2018 Publication	<1 %
84	Submitted to Federal University of Technology Student Paper	<1 %
85	Nishant Sinha. "Internal state recovery of Espresso stream cipher using conditional sampling resistance and TMDTO attack", Advances in Mathematics of Communications, 2019 Publication	<1 %
86	Satti, M., and S. Kak. "Multilevel Indexed Quasigroup Encryption for Data and Speech", IEEE Transactions on Broadcasting, 2009. Publication	<1 %
87	hdl.handle.net Internet Source	<1 %
88	patents.google.com Internet Source	<1 %
89	technodocbox.com Internet Source	<1 %
90	vdoc.pub	

	Internet Source	<1%
91	"Advances in Information and Computer Security", Springer Science and Business Media LLC, 2014 Publication	<1%
92	"Fast Software Encryption", Springer Nature, 2001 Publication	<1%
93	"Fast Software Encryption", Springer Science and Business Media LLC, 2004 Publication	<1%
94	"Selected Areas in Cryptography", Springer Science and Business Media LLC, 2013 Publication	<1%
95	Advances in Intelligent Systems and Computing, 2013. Publication	<1%
96	Harshvardhan Tiwari, Krishna Asawa. "A secure and efficient cryptographic hash function based on NewFORK-256", Egyptian Informatics Journal, 2012 Publication	<1%
97	Mollin, . "Cryptographic Basics", Discrete Mathematics and Its Applications, 2006. Publication	<1%

98	Submitted to Nanyang Technological University, Singapore Student Paper	<1%
99	abstract.ups.edu Internet Source	<1%
100	ciit.finki.ukim.mk Internet Source	<1%
101	d-nb.info Internet Source	<1%
102	oa.upm.es Internet Source	<1%
103	scholar.sun.ac.za Internet Source	<1%
104	spectrum.library.concordia.ca Internet Source	<1%
105	www.math.unipd.it Internet Source	<1%
106	Submitted to Higher Education Commission Pakistan Student Paper	<1%
107	M. García-Martínez, E. Campos-Cantón. "Pseudo-random bit generator based on lag time series", International Journal of Modern Physics C, 2014 Publication	<1%

108	Zhuo Liu, Yong Wang, Gongkun Jiang, Leo Yu Zhang. "Design and Analysis on a Parallel Chaos-Based Hash Function", International Journal of Bifurcation and Chaos, 2020 Publication	<1 %
109	drops.dagstuhl.de Internet Source	<1 %
110	ntnuopen.ntnu.no Internet Source	<1 %
111	www.informatica.si Internet Source	<1 %
112	www.readbag.com Internet Source	<1 %
113	"Cryptographic Hardware and Embedded Systems – CHES 2008", Springer Nature, 2008 Publication	<1 %
114	"Multimedia Security Using Chaotic Maps: Principles and Methodologies", Springer Science and Business Media LLC, 2020 Publication	<1 %
115	"Software Engineering Research, Management and Applications", Springer Science and Business Media LLC, 2019 Publication	<1 %
116	Iryna Egorova. "Scattering theory for Jacobi operators with a steplike quasi-periodic	<1 %

background", Inverse Problems, 06/01/2007
Publication

117	Submitted to University of Wales, Bangor Student Paper	<1 %
118	mi.eng.cam.ac.uk Internet Source	<1 %
119	Submitted to 2642 Student Paper	<1 %
120	Amar Abdelmalek Ghehioueche, Noureddine Chikouche, Fares Mezrag. "Performance Evaluation and Analysis of Encryption Schemes for Wireless Sensor Networks", 2019 International Conference on Digitization (ICD), 2019 Publication	<1 %
121	Submitted to Loughborough University Student Paper	<1 %

Exclude quotes	On	Exclude matches	< 14 words
Exclude bibliography	On		