

TRAINING PROJECT REPORT
ON
DEVELOPMENT OF QUASIGROUP BASED
CRYPTOGRAPHIC SCHEMES

Submitted by
ARYA TANMAY GUPTA



RAMANUJAN COLLEGE (UNIVERSITY OF DELHI)

Under the guidance of
DR. SUCHETA CHAKRABARTI

Scientist-‘F’



Scientific Analysis Group
Defence Research & Development Organisation

In fulfillment of the completion of the training program

December 2015

Acknowledgements

I thank all the personalities responsible behind the success of this project, especially the following ones.

I thank **Dr. G. Athithan**, former Director, Scientific Analysis Group (SAG), Defence Research and Development Organization (DRDO), to allow me to conduct this project and support it. I also thank **Mrs. Anu Khosla**, Director, SAG, DRDO, for supporting to continue this project. I thank **Dr. Sucheta Chakrabarti**, Scientist 'F', SAG, DRDO, to guide me all throughout this project and supported me in all its phases. I thank **Dr. S. K. Pal**, Scientist 'G', SAG, DRDO, to help me in various phases of the project, and for having various discussions fruitful for me and the project.

I thank **Dr. S. P. Agarwal**, Principal, and **Dr. Nikhil Rajput**, Assistant Professor, Department of Computer Science, Ramanujan College, University of Delhi, to support me throughout the project. I thank **Mrs. Bhavya Ahuja**, Teacher-in-Charge, Department of Computer Science, Ramanujan College, University of Delhi, to help and support me throughout the project.

I thank my classmate **Ms. Nivedita Rai**, Department of Computer Science, Ramanujan College, University of Delhi, who identified my abilities and motivated me to take this project.

I thank my **parents** and almighty **god** for everything I have and everything I am.

Arya Tanmay Gupta

College Roll No. 13/0110

Department of Computer Science,

Ramanujan College (University of Delhi)

Declaration

I hereby declare that this project report entitled **“DEVELOPMENT OF QUASIGROUP BASED CRYPTOGRAPHIC SCHEMES”** being submitted to the Department of Computer Science of Ramanujan College (University of Delhi) for the fulfillment of the completion of the training program in December 2015, is a is a bonafide record of original work carried out under the guidance and supervision of **Dr. Sucheta Chakrabarti, Scientist ‘F’, SAG, DRDO** and has not been presented elsewhere.

Arya Tanmay Gupta

College Roll No. 13/0110



CERTIFICATE

This is to certify that this project report entitled **“DEVELOPMENT OF QUASIGROUP BASED CRYPTOGRAPHIC SCHEMES”** by **Arya Tanmay Gupta**, submitted for fulfillment of the requirement for the completion of training program of B. Tech. in Computer Science, under Department of Computer science of Ramanujan College, University of Delhi, during the period, is a bonafide record of work carried out under my guidance and supervision.

Dr. Sucheta Chakrabarti

Scientist ‘F’

SAG, DRDO

December 2015

Table of Contents

1. Introduction	1
2. Mathematical Background for cryptography	4
2.1. Number Theory	4
2.1.1. Basic Number System	4
2.1.2. Some Basic Concept of Unique Factorization, Euclidean Algorithm, Gcd, Prime & Perfect Numbers; Fermat & Wilson's Theorem & Euler's Phi – Function	9
2.1.3. Euler's Phi-Function	11
2.1.4. Modular Number System	12
2.2. Algebraic Structures	17
2.2.1. Groups	17
2.2.1.1. Subgroups	20
2.2.1.2. Cyclic Groups	21
2.2.1.3. Permutation groups	23
2.2.2. Rings	26
2.2.3. Fields	28
3. Overview of Cryptography	30
3.1. Introduction	30
3.2. Historical Background	30
3.3. Modern Cryptography	33
4. Quasigroup Based Cryptography	36
4.1. Introduction	36
4.2. Quasigroups	36
4.3. Lexicographical Arrangement of quasigroups of order 4	39
4.4. Quasigroup Based String Transformations	42
4.5. Cryptographic Schemes Based on Quasigroup String Transformation	43
4.5.1. Cryptographic Schemes based on Left Quasigroup String Transformation	43

4.5.1.1. Cryptographic scheme based on left quasigroup transformation where leader is fixed	43
4.5.1.2. Cryptographic scheme based on left quasigroup transformation where leader is based on key	45
4.5.2. Iterated cryptographic schemes based on composite left quasigroup transformation	47
4.5.2.1. Iterated cryptographic scheme based on composite left quasigroup transformation where leader is fixed	47
4.5.2.2. Iterated cryptographic scheme by composite left transformation where leader is variable and is based on iteration.	49
4.5.3. Generalized iterated cryptographic scheme based on composite left quasigroup transformation	51
5. Experiments and Observations	55
6. Quasigroup based Stream Cipher Edon 80	68
6.1. Implementation of Edon 80	70
6.1.1. Edon 80 Algorithm	70
6.1.2. Modified Edon 80 Algorithm	75
6.1.3. Modified Edon 80 Algorithm for image	75
6.2. Application of Edon 80 stream cipher on strings and images	77
6.3. Statistical analysis of keystream generated by Edon80	80
7. Conclusion	81
8. Future Work	82
Appendices	83
Appendix-A	
Some general Functions used in the programs	83
Appendix-B	
List of programs	93
Appendix-C	

Statistical report of the NIST test applied on Edon 80	161
References	170

Abstract

The project entitled **“DEVELOPMENT OF QUASIGROUP BASED CRYPTOGRAPHIC SCHEMES”** mainly deals with different cryptographic schemes based on non-associative and non-commutative algebraic structures viz. quasigroups. Today one of the most important National issue is secure communication and information/data storage in both military, intelligence agencies and civilian. Therefore always there is a demand to develop new encryption schemes to achieve these goals by meeting the challenges of upcoming threats. In late 1980, mainly European crypto-community has been visualized the great potentiality of using non-associative and non-commutative algebraic structures in cryptology. It gives a new direction in cryptology. In this direction Quasigroups are very suitable algebraic structures to be used.

In this project we have implemented different quasigroup based transformations and different cryptographic schemes based on these transformations. Different experiments are carried out and some important properties have been observed. Here experiments mainly are carried out on quasigroups of order 4. Also write a program for arranging all quasigroups of order 4 in Lexicographical order. Finally we have coded the stream cipher Edon-80 and its modified version. Edon-80 is one of the selected stream cipher out of 15 selected ciphers for final round in e-stream competition. It has been implemented and verified successfully. In this project all cryptographic schemes are coded in MatLab.

All schemes are verified and tested by using the examples of encryption / decryption. Edon-80 has been applied on different type of inputs and also gives the output in different forms. It has been also applied successfully on image inputs. Some samples of Edon-80 key stream have been generated of length 10^6 bits and NIST tests have been carried out successfully.

The project plays a very important role in analyzing the pattern of output strings and choice of quasigroups and leader. It will also help to understand the insight of design principles of Edon-80. It will help to design new schemes / customized Edon-80 in future.

Chapter 1.

Introduction

The word **cryptography** comes from two Greek words, *kryptós* (means “hidden” or “secret”) and *graphien* (means “writing” or “to study”). It is a science of practicing secure communication in the presence of third parties, called adversaries. More generally, it is about constructing and analyzing protocols that block adversaries; various aspects in information security such as *data confidentiality*, *data integrity*, *authentication*, and *non-repudiation* are central to modern cryptography. Modern cryptography exists at the intersection of the disciplines of mathematics, computer science, and electrical engineering. Applications of cryptography include ATM cards, computer passwords, and electronic commerce.

The use of the science of cryptography is as old as some 4000 years ago by Egyptians. Cryptography prior to the modern age was effectively synonymous with **encryption**, the conversion of information from a readable state to apparent nonsense. The originator of an encrypted message shared the decoding technique needed to recover the original information only with intended recipients, thereby precluding unwanted persons from doing the same. Since World War I and the advent of the computer, the methods used to carry out cryptology have become increasingly complex and its application more widespread.

Modern cryptography is heavily based on mathematical theory and computer science practice. The originator converts the form of the original message using some mathematical algorithm, a decoding technique, into some form and sends it. This changed form is called **cipher**. The receiver already knows how to break this cipher into original message. What the originator does is called encryption and what the receiver does is **decryption**. A good cryptographic algorithm is that which an adversary is unable to break.

Cryptographic algorithms are designed around computational hardness assumptions, making such algorithms hard to break in practice by any adversary. It is theoretically possible to break such a system, but it is infeasible to do so by any known practical means.

These schemes are therefore termed computationally secure; theoretical advances, e.g., improvements in integer factorization algorithms, and faster computing technology require these solutions to be continually adapted. There exist information – theoretically secure schemes that provably cannot be broken even with unlimited computing power – an example is the one-time pad – but these schemes are more difficult to implement than the best theoretically breakable but computationally secure mechanisms.

In almost all cryptographic schemes, one more string or bit sequence, mathematically independent of message or cipher, is used in encryption. This is called **key**. There are two types of keys: *Private Key* and *Public Key*. **Private Key** systems are used to share secret information like at corporates, defence, etc. Private keys are not shared with anyone except the communicating parties. **Public keys** are used mostly in public communication like telephone channels, mobile phones, etc. Public keys are available to everyone through public key repository or directory.

Public key cryptography was a revolutionary concept which came in 1976. It was introduced by Diffie and Hellman through their paper “New Directions in cryptography”. This paper also provided a new and indigenous method for key exchange, the security of which is based on the intractability of discrete logarithm problems. In 1978, Rivest, Shamir and Adleman discovered the first public-key encryption and signature scheme, now referred to as RSA. The RSA scheme is based on factoring large integers.

Another class of powerful and practical public-key schemes was found by ElGamal in 1985. These are also based on the discrete logarithm problem. One of the most significant contributions provided by public-key cryptography is the digital signature. In 1991 the first international standard for digital signatures (ISO/IEC9796) was adopted. It is based on the RSA public-key scheme. In 1994 the U.S. Government adopted the Digital Signature Standard, a mechanism based on the ElGamal public key scheme.

The search for new public-key schemes, improvements to existing cryptographic mechanisms, and proofs of security continues at a rapid pace. Various standards and infrastructures involving cryptography are being put in place. Security products are being developed to address the security needs of an information intensive society.

Our main methods of encryption were based on cryptography. In this project, we have used the incredible mathematical properties of quasigroups to encipher and then decipher the encrypted texts.

We have subjected various plaintexts to many methods of encryptions using quasigroups. We made minor changes to those algorithms to observe the effect on the ciphertexts. We have made a program to sort a collection of quasigroups lexicographically.

Then we implemented Edon 80 in Matlab. We have enciphered texts and images using Edon 80. We have tested five one-million bit keys using NIST test protocol.

The details of all what we implemented, the observations in those experiments & statistical analyses are included in this report.

Chapter 2.

Mathematical Background for Cryptography

2.1. Number Theory

In this section we will discuss the fundamental concepts and properties of number systems [5] mainly used for cryptography.

2.1.1. Basic Number System

The number system evolved over time by expanding the notion of what we mean by the word “number”. At first, “number” meant something you could count, like how many cows a farmer owns. These are called the natural numbers or sometimes counting numbers.

Natural Numbers – The numbers evolved from the need of counting something. i.e. 1, 2, 3, Generally denoted by **N** or **Z₊**.

At some point, the idea of zero came to be considered as a number – If you don’t have anything.

(Say, a farmer at present has no cows – very commonly say he owns no cows or zero cows). So the whole numbers evolved.

Whole Numbers – Natural numbers together with zero. i.e. 0,1,2,3, Generally denoted by **W**.

Let us look the very fundamental important use of the numeral ‘0’. Invention of zero make possible the place-value number system that we use today. When we write a number we use only (0, 1, 2, ..., 9) – ten numerals. These numerals can stand for ones, tens, hundreds,... depending on their position in the number. In order for this to work, we have to have a way to make an empty place in a number otherwise the place value will be wrong. This is what the main role the numeral zero plays in the number system. The number zero obeys most of the rules of arithmetic that ordinary numbers do, so we call it a number. It is

rather a special number, because it does not quite obey all the same laws as other numbers – like you can't divide by zero.

Now come to the **negative numbers**. The idea of negative numbers are more abstract than zero. These numbers need to express when somebody takes something from others. For example, in your account you don't have any money (i.e. $\text{Acc} = 0$). Now you take some loan from somebody so your account becomes less than zero and again to come to zero you have to save that much amount. To represent this number the idea of negative numbers evolved. So extended set of whole numbers generates.

Integers - Whole numbers together with negative of natural numbers, i.e., ..., -3, -2, -1, 0, 1, 2, 3, Generally denoted by **Z**.

Natural numbers sometimes also called **+ive integers**, denoted by **Z₊**; its negatives are called **negative integers** and denoted by **Z₋**. Note that zero is considered to be neither negative nor positive.

The next extended set of numbers has been evolved from the idea of fractions. Many things in real life are measured in fractions, like $\frac{1}{2}$ spoon of sugar, To meet these requirements the idea of fractional numbers evolved which together with integers gives the set of rational numbers.

Rational Numbers – All numbers of the form a/b , where a and b are integers (but b can't be zero), i.e., rational numbers include what we usually call fractions. Fractions can be < 1 , i.e., $\frac{1}{2}$, $\frac{3}{4}$, ... or they can be > 1 , i.e., $2\frac{1}{2} = 5/2$, ... Note that integers can be expressed as a fraction where denominator is 1 i.e. say $3 = 3/1$.

Rational numbers are also not able to meet the need of numbers which generates from the geometrical development. These are numbers which can't be expressed as a fraction.

Irrational Numbers – The numbers which can't be expressed as a ratio of integers. i.e. as a decimal they never repeat or terminate as rational number do one or the other.

Examples – (Rational / Irrational)

$\frac{3}{4} = 0.75$ – rational (*terminates*)

$\frac{2}{3} = 0.66666\dots$ - rational (*repeats*)

$\frac{5}{7} = 0.714285714285\dots$ – rational (*repeats*)

square root of 2 = 1.41421356... irrational (*never repeats or terminates*)

$\pi = 3.14159265...$ irrational (*never repeats or terminates*)

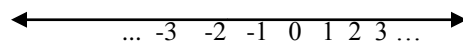
$e = 2.718281828...$ irrational (*never repeats or terminates*)

These all numbers together consist the set of Real numbers denoted by \mathbf{R} .

The real numbers have the **property that they are ordered**, which means that given any two different numbers we can definitely say that one is greater or less than the other, i.e., for any two numbers a, b belongs to \mathbf{R} one of the following three statements is true:

1) $a < b$ 2) $a > b$ 3) $a = b$

The ordered nature of the real numbers helps to arrange them along a line (imagine that the line is made up of infinite numbers of points all packed so closely together that they form a solid line).The points are ordered so that the points to the right side of the zero are greater than the points to the left. The Real number line is presented as follows:



i.e. every real number correspondence to a distance on the number line starting from the center zero ;+ , - shows the direction right, left correspondingly.

Another important notion is the absolute value of the number. When we talk about how “large” a number is without mentioning its direction we use the absolute value function. The absolute value of a number is the distance from that number to the origin on the number line. It is always a non-negative number, i.e., $|4| = 4$; $|-5| = 5$. Be careful to do the arithmetic inside the absolute value sign. First perform the arithmetic then take the absolute value.

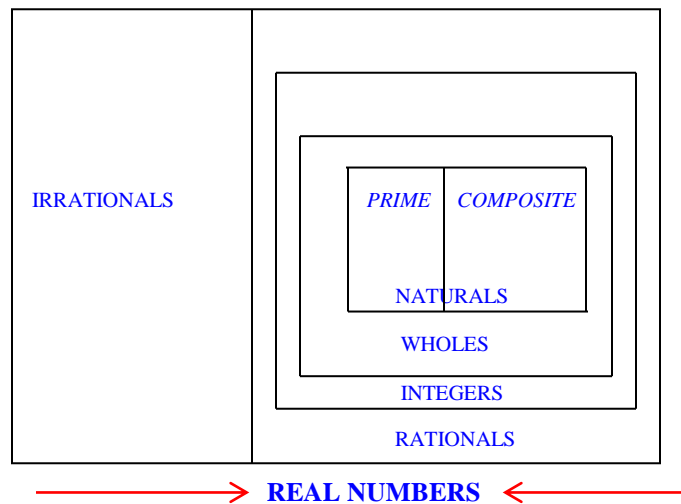
i.e., $|5 + (-2)| = 3$ but very common **mistake** is $|5 + (-2)| = 7$.

The set of natural numbers can be divided into two important subsets viz. the set of prime numbers and the set of composite numbers.

Prime Numbers – All natural numbers greater than one which have no divisors except themselves and one. i.e. 2,3,5,7,

Composite Numbers – All natural numbers greater than one which is not prime numbers.
i.e. 4,6,8,... .

So let us represented the nested diagram of the real number system as follows



Now development of different areas of mathematics demands **more numbers than real numbers**. The problem starts when the need arise to solve the equation $x^2 + 1 = 0$. This equation has no solution in the set of real numbers since the product of a real number multiplied by itself is zero or +ive. So to fulfill this need the new numbers have been invented and a new dimension has been added (viz. imaginary axis) to the real number system gives **imaginary number system**.

To solve the above mentioned equation introduce a number represented by i s.t. $i^2 = -1$.

Imaginary Number – An imaginary number is a multiple of a number by i s.t. $i^2 = -1$, i.e., it is a number system that contains square root of negative numbers.

This extended set of numbers is called the set of complex numbers and denoted by **C**.

Complex Number – The numbers of the form $a + bi$, in which a, b are real numbers are called complex numbers.

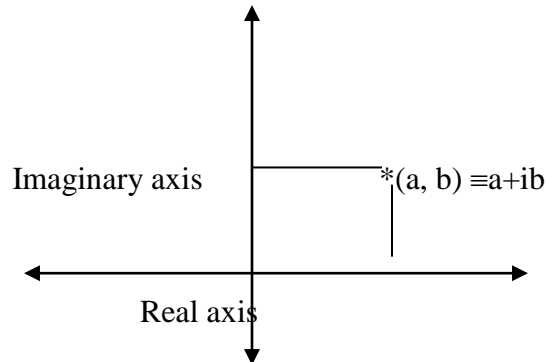
If $b \neq 0$ then the number $a + bi$ is called (general) imaginary number.

If $b \neq 0$ but $a = 0$ then it is called pure imaginary.

If $b = 0$ then it is real.

So complex number system is the extension of the real number system. In other words the real number represents point on a line, where as complex numbers can be placed in correspondence with the points on a plane and we can perform the arithmetic.

Geometrical representation of complex number $a + bi$ is as follows:



The complex number can also be divided into two subsets, viz. the **Algebraic numbers** and the **Transcendental numbers**.

Algebraic Numbers – These are complex numbers that can be obtained as the root of a polynomial with integer coefficients. For example, rational numbers, roots of integers.

Transcendental Numbers – Transcendental numbers are real numbers that are not algebraic viz. e , π .

So now after defining different types of numbers we will formally define what is meant by number system as follows:

Number System: It is just a collection of objects (set of numbers) for which

- There is a definition of what it means for two objects be equal.
- There is a rule for how to add two objects rule for multiplication

(Subtraction & division can be deduced from these provided that all objects have corresponding negative & some objects have corresponding reciprocals.)

- These rules for + & • satisfy the familiar properties of arithmetic, such as commutative, associative and distributive.

Examples: Complex numbers, Real numbers, Rational numbers, Natural numbers etc.

2.1.2. Some Basic Concept of Unique Factorization, Euclidean Algorithm, Gcd, Prime & Perfect Numbers; Fermat & Wilson's Theorem & Euler's Phi - Function

Factorization is a process of decomposing a number into its prime factors. This number n is decomposed into its prime factors $p_1, p_2^{a_2}, \dots, p_k^{a_k}$ as

$$n = p_1^{a_1} \cdot p_2^{a_2} \cdot \dots \cdot p_k^{a_k}.$$

where a_1, a_2, \dots, a_k are the exponents over those prime numbers.

Theorem 1.(Euclidean's first theorem) If p is prime, and $p \mid ab$, then $p \mid a$ or $p \mid b$.

For example, $3 \mid 12 = 4 \cdot 3$; $3 \mid 3$.

Three integers (a, b, c) which satisfy $a^2 + b^2 = c^2$ are called **pythagorean triples**.

If those triples are positive integers, and have no common factors, we call them **fundamental triple**.

For example, $(3, 4, 5)$ and $(5, 12, 13)$ are both pythagorean and fundamental triples.

Theorem 2. Given any pair of relatively prime integers, say (a, b) such that one of them is odd and the other even and $a > b > 0$, then $(a^2 - b^2, 2ab, a^2 + b^2)$ is a Fundamental Triple. Furthermore, every fundamental Triple is of this form.

Theorem 3 – (Fundamental Theorem of Arithmetic)

Factorization into primes is unique up to the order

Ex. $30 = 2 \cdot 3 \cdot 5 = 3 \cdot 5 \cdot 2$.

Definition 1. Given integers a & b , the gcd of a & b is the largest positive integer which divides both a & b and denoted by $\gcd(a, b)$ or (a, b) .

Two positive integers a & b are said to be **co-prime** or **relatively prime** to each other if they have no common factor (except 1) and denoted by $(a,b)=1$.

For Example, 2 and 3 & 32 and 75 are co-primes since $(2, 3)=1$ and $(32, 75)=1$

Theorem 4. Let a and b be integers and let $g = \gcd(a, b)$. Then there exist integers m & n such that $g = am + bn$.

The constructive prove of this theorem (i.e. the algorithm) is called The Euclidean Algorithm.

Theorem 5. (Euclidean Algorithm) Given integers a & b , $b \neq 0$, \exists unique integers m & r such that $a = m \cdot b + r$, r is here a natural number and is smaller than b .

From this it also follows that if $a = m \cdot b + r$ then $\gcd(a, b) = \gcd(b, r)$

Example: calculate $\gcd(208, 1800)$.

$$1800 = 208 \cdot 8 + 136$$

$$208 = 136 \cdot 1 + 72$$

$$136 = 72 \cdot 1 + 64$$

$$72 = 64 \cdot 1 + 8$$

$$64 = 8 \cdot 8 + 0$$

We stop when we get $r = 0$, i.e., remainder is zero. So the last non-zero remainder (by the above result) i.e. 8 is the $\gcd(208, 1800)$.

Using these equations we can calculate m & n such that $8 = 208 \cdot m + 1800 \cdot n$.

Start from the next to last of the equations and successively eliminate the remainders 64, 72, 136:

$$\begin{aligned} 8 &= 72 - 64 \\ &= 72 - (136 - 72) = 2 \cdot 72 - 136 \\ &= 2 \cdot (208 - 136) - 136 = 2 \cdot 208 - 3 \cdot 136 \\ &= 2 \cdot 208 - 3 \cdot (1800 - 208 \cdot 8) \\ &= 26 \cdot 208 - 3 \cdot 1800 \end{aligned}$$

So, the value of two integers m and n 26 and -2 respectively.

Definition 2. A +ive integer is said to be **Perfect** if it is the sum of its proper divisors (i.e. +ive divisors less than itself)

Ex. (i) $6 = 1 + 2 + 3$

(ii) $28 = 1 + 2 + 4 + 7 + 14$

Definition 3. Let $M(n) = 2^n - 1$. A **Mersenne prime** is a prime of the form $M(n)$ for some integer n .

Ex. 3, 7, 31, ...

Properties 1. If $M(n)$ is a Mersenne prime then $m = 2^{n-1} \cdot M(n)$ is a perfect number.

Proposition 2. If n is a composite then $M(n)$ is composite.

It follows that if $M(n)$ is prime then n has to be prime.

It is to be noted that the largest known Mersenne prime is $2^{2976221} - 1$ – discovered in 1997.

Proposition 3. If p is a prime then $a^p \equiv a \pmod{p}$ for any integer a

Theorem 6.(Fermat's Little Theorem) If p is a prime and p does not divide a then $a^{p-1} \equiv 1 \pmod{p}$

Theorem 7. (Wilson's theorem) Any integer $n > 1$ is prime iff $(n-1)! + 1$ is divisible by n .

2.1.3. Euler's Phi-Function

It is one of the most important function in the number Theory . Euler introduced this function to generalize a congruence result of Fermat's little theorem which concerns congruences with prime moduli to arbitrary moduli and the function is denoted by ϕ .

Definition 1. The Euler function ϕ is given by $\phi(n)$ = the number of integers 'a' satisfying $1 \leq a \leq n$ & $(a, n) = 1$, i.e. $\phi(n)$ is the number of +ive integers less than n which are co-prime to n .

Some Properties of ϕ Function:

Property 1. (Gauss) For each +ive integer $n \geq 1$, $n = \sum_{d|n} \phi(d)$;
 $d | n$

i.e.the sum being extended over all +ive divisors of n .

Example. $n = 10$; $d = 1, 2, 5, 10$

$$\sum \phi(d) = \phi(1) + \phi(2) + \phi(5) + \phi(10) = 1 + 1 + 4 + 4 = 10 = n$$

Note - $\phi(1) = 1$ since $(1,1) = 1$

Property 2. $\phi(n) = n-1$ if and only if n is prime.

Property 3. If p is a prime & $k > 0$ then $\phi(p^k) = p^k - p^{k-1} = p^k(1 - 1/p)$

Example. $\phi(9) = \phi(3^2) = 3^2 - 3 = 9 - 3 = 6$ & we can verify by definition

$$\phi(9) = |\{1, 2, 4, 5, 7, 8\}| = 6$$

Theorem 1. The function ϕ is multiplicative i.e. $\phi(mn) = \phi(m)\phi(n)$ where $(m,n) = 1$

And if $n = p_1^{\alpha_1} \dots p_k^{\alpha_k}$ then $\phi(n) = \prod p_i^{\alpha_i} - p_i^{\alpha_i-1} = n \prod (1 - 1/p_i)$

$$p_i | n$$

Theorem 2. (Euler's Theorem) --- If $(a,n) = 1$ then $a\phi(n) \equiv 1 \pmod{n}$

Let $n = p$ and p does not divide a i.e $(a,p) = 1$, then $\phi(n) = p - 1$

So, $a^{\phi(p)} \equiv 1 \pmod{p}$ by the above theorem. i.e. $a^{p-1} \equiv 1 \pmod{p}$ –Fermat's little theorem.

2.1.4. Modular Number System

This is a new number system [8] which plays an important role in many areas. It plays a major role in cryptography from classical to modern era.

Definition 1. Two integers a & b are congruent modulo n , written $a \equiv b \pmod{n}$ or $a \equiv b \pmod{n}$ if $b \equiv a + kn$ where $k \in \mathbf{Z}$.

This definition is equivalent to each of the following:

- (i) $n \mid b - a$
- (ii) The set $\{ nk + a \}_{k \in \mathbb{Z}}$ is the same set as the set $\{ nk + b \}_{k \in \mathbb{Z}}$
- (iii) If a & b are natural numbers, it is also equivalent to a & b leave the same remainder when divided by n .

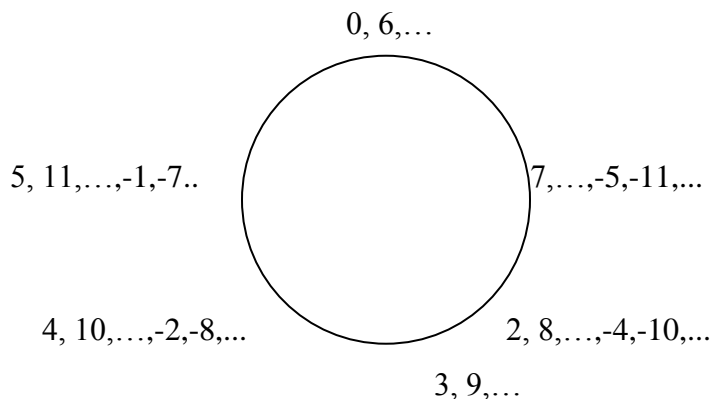
Example.

1325	\equiv	2 (9)
182	\equiv	119 (9)
3	\equiv	-1(4)
13	\equiv	0 (13)

Basic Properties

The congruence symbol looks like an equality, and this is no accident. In fact, one can view congruence geometrically as a kind of equality

Ex. Say $n = 6$



Integers which lie over the same point on the circle are congruent modulo n ; integers lying over different points are not.

Congruence satisfies the following fundamental properties:

- (i) If $a \equiv b \pmod{n}$ then $ka \equiv kb \pmod{n}$; $k \in \mathbb{Z}$
- (ii) If $a \equiv b \pmod{n}$ & $b \equiv c \pmod{n}$ then $a \equiv c \pmod{n}$

(iii) If $a \equiv b \pmod{n}$ & $a_1 \equiv b_1 \pmod{n}$ then $a + a_1 \equiv b + b_1 \pmod{n}$
& $aa_1 \equiv bb_1 \pmod{n}$

(iv) If $a \equiv b \pmod{n}$ and $d \mid n$ then $a \equiv b \pmod{d}$

(v) If $a \equiv b \pmod{r}$ and $a \equiv b \pmod{s}$ then $a \equiv b \pmod{[r,s]}$

Viz. $79 \equiv 2 \pmod{7}$ & $79 \equiv 2 \pmod{11}$ then $79 \equiv 2 \pmod{77}$

(vi) If $ra \equiv rb \pmod{n}$ then $a \equiv b \pmod{n / (r,n)}$

Say, $4.3 \equiv 4.8 \pmod{10}$

So, $3 \equiv 8 \pmod{10 / (10,4)}$ i.e. $3 \equiv 8 \pmod{5}$

Two special cases when (i) $r \mid n$ & (ii) $(r,n) = 1$

(vii) If $ra \equiv rb \pmod{rn}$ then $a \equiv b \pmod{n}$

(viii) If $ra \equiv rb \pmod{n}$ and $(r,n) = 1$ then $a \equiv b \pmod{n}$

Divisibility & congruences leads to the invention of new number system with respect to modular arithmetic. In this number system objects are congruence classes contain infinite integers and the rule of add & multiplication are as follows:

For some given n , $[a] = \{x \in \mathbf{Z} \text{ s.t. } a \equiv x \pmod{n}\}$ & we say that $[a] \in \mathbf{Z}_n$.

Now let $[a], [b] \in \mathbf{Z}_n$ then $[a] + [b] = [c]$ where $c = \text{the remainder of } (a + b) / n$

& $[a] \cdot [b] = [c]$ where $c = \text{the remainder of } (a \cdot b) / n$

Example. Let us consider \mathbf{Z}_2

It contains of two elements i.e. two integers classes viz. $[0]$ & $[1]$, i.e $[0] = \{ \dots, -4, -2, 0, 2, 4, \dots \}$ - all even integers & $[1] = \{ \dots, -3, -1, 1, 3, 5, \dots \}$ - all odd integers.

We can present the $+$ & \bullet in the following tables & we can check the basic arithmetic properties:

$+$	$[0]$	$[1]$
$[0]$	$[0]$	$[1]$
$[1]$	$[1]$	$[0]$

\bullet	$[0]$	$[1]$
$[0]$	$[0]$	$[0]$
$[1]$	$[0]$	$[1]$

We can easily verify that $[a] + [b] = [b] + [a]$,...so on. It satisfy all properties of arithmetics under addition and multiplication.

Linear Congruence

An equation of the form $ax \equiv b \pmod{n}$ is called a Linear congruence. By a solution of such an equation we mean an integer x_0 for which $ax_0 \equiv b \pmod{n}$. By definition $ax_0 \equiv b \pmod{n} \Leftrightarrow n \mid ax_0 - b \Rightarrow ax_0 - b = ny_0$ for some integer y_0 . So , problem of finding all integers satisfying $ax \equiv b \pmod{n}$ is identical with that of obtaining all solutions of linear equations $ax - ny = b$.

Theorem 1. : The linear congruence $ax \equiv b \pmod{n}$ has a solution iff $d = (a,n) \mid b$.

If $d \mid b$ then it has d mutually incongruent (i.e. d distinct) solutions modulo n and if x_0 is particular solution then d - solutions given by $\{x_0 + tn/d\}$ where

$$t = 0, 1, \dots, d-1$$

Example. Let us consider the equation $18x \equiv 30 \pmod{42}$. Since $\gcd(18, 42) = 6$ & $6 \mid 30 \Rightarrow \exists$ exactly 6 distinct solutions in Z_{42}

Now the particular solution $x = 4$ so all distinct solutions are $x = 4 + (42/6)t = 4 + 7t$ where $t = 0, 1, \dots, 5$. Hence the six distinct solutions in Z_{42} are 4, 11, 18, 25, 32, 39.

Property1. If $(a, n) = 1$ then the linear congruence $ax \equiv b \pmod{n}$ has exactly one solution.

Note that it will give the necessary & sufficient condition for the existence of the inverse (unit) in Z_n .

Generally an element of Z_n does not have multiplicative inverse.

Example. The equation $3x \equiv 1 \pmod{6}$ in Z_6 has no solution i.e. 3 has no multiplicative inverse in Z_6 .

Now for example $3x \equiv 1 \pmod{7}$, by using the property 1 it has a unique solution since $(3, 7) = 1$, so 3 has a multiplicative inv in Z_7 .

Theorem 2. (Chinese Remainder theorem) If m_1, m_2, \dots, m_n are pair wise relatively prime natural numbers > 1 (≥ 2), and a_1, a_2, \dots, a_n are any integers (not necessarily distinct) then the system of simultaneous linear congruences

$$x \equiv a_1 \pmod{m_1}$$

$$x \equiv a_2 \pmod{m_2}$$

.

.

.

$$x \equiv a_n \pmod{m_n}$$

has a unique solution modulo $M = m_1 \cdot \dots \cdot m_n$ which is given by

$$x = \sum a_i M_i y_i \pmod{M} \text{ where } M_i = M/m_i \text{ \& } y_i = M_i^{-1} \pmod{m_i}$$

Example.

$$x \equiv 2 \pmod{3}$$

$$x \equiv 3 \pmod{5}$$

$$x \equiv 2 \pmod{7}$$

So, $M = 3 \cdot 5 \cdot 7 = 105$ & $M_1 = M/m_1 = 35$, $M_2 = 21$, $M_3 = 15$

$$y_i = M_i^{-1} \pmod{m_i}$$

So, $y_1 = 35^{-1} \pmod{3}$ i.e. $35y_1 \equiv 1 \pmod{3} \Rightarrow y_1 = 2$

Similarly $21y_2 \equiv 1 \pmod{5} \Rightarrow y_2 = 1$ & $15y_3 \equiv 1 \pmod{7} \Rightarrow y_3 = 1$

So, $x = \sum a_i M_i y_i \pmod{M}$ i.e. $x = 2 \cdot 35 \cdot 2 + 3 \cdot 21 \cdot 1 + 2 \cdot 15 \cdot 1 \pmod{105}$

$$= 233 \pmod{105}$$

$\equiv 23 \pmod{105}$ is the unique solution.

Now we can use the CRT for solving the linear equation in large modulo n by decomposing n into relatively prime factors m_i and solve the system of linear congruences in corresponding m_i 's

2.2. Algebraic Structures

This section deals with some commonly used algebraic structures [9] and its important properties which are used as the building blocks of cryptography.

2.2.1. Groups

These systems are composed of nonempty set together with binary operations defined on this set so that certain properties hold.

Definition 1. A group is an ordered pair (G, \bullet) , where G is a non-empty set and \bullet is a binary operation on G such that following properties hold:

- (i) For all $x, y, z \in G$, $x \bullet (y \bullet z) = (x \bullet y) \bullet z$ (associative law);
- (ii) There exist $e \in G$ such that for all $x \in G$, $e \bullet x = x = x \bullet e$ for all $x \in G$ (existence of an identity);
- (iii) For all $x \in G$, there exists $y \in G$ such that $x \bullet y = e = y \bullet x$ (existence of an inverse)

So, a group is a mathematical system (G, \bullet) satisfying above three axioms.

Theorem 1. Let (G, \bullet) be a group $e \in G$

- (i) There exists a unique element $e \in G$ such that for any $x \in G$, $e \bullet x = x = x \bullet e$
- (ii) For each $x \in G$, there exists a unique $y, v \in G$ such that $x \bullet y = e = y \bullet x$

Note that the unique element $e \in G$ satisfying (ii) of group is called the **identity** element of the group and the unique element y of an element $x \in G$ satisfying (iii) of the group G is called the **inverse** of x and is denoted by x^{-1} .

A group (G, \bullet) is called commutative or Abelian group if $x \bullet y = y \bullet x$ for all $x, y \in G$.

If (G, \bullet) is a group such that the number of elements of G is finite, then the group is said to be a **finite group** and the number of elements of G is called the **order** of the group G .

Examples

1. Set of integers \mathbf{Z} is a group under the binary operation $+$ (addition) . Here $0 \in \mathbf{Z}$ is the identity and for all $x \in \mathbf{Z}$, $-x \in \mathbf{Z}$. This is also a commutative group since for all $x, y \in \mathbf{Z}$, $x+y=y+x$.

Note that \mathbf{Z} is not a group under the binary operation \bullet (multiplication).

2. Similarly $(\mathbf{R}, +)$, $(\mathbf{Q}, +)$, $(\mathbf{C}, +)$ and also $\mathbf{R}^* = (\mathbf{R} \setminus \mathbf{0}, \bullet)$, $\mathbf{Q}^* = (\mathbf{Q} \setminus \mathbf{0}, \bullet)$, $\mathbf{C}^* = (\mathbf{C} \setminus \mathbf{0}, \bullet)$ are commutative groups

3. $\mathbf{GL}_2(\mathbf{R})$ -The general linear group of degree 2 is the group of all 2×2 matrices of real numbers with nonzero determinants under matrix multiplication.

Since matrix multiplication satisfy the following three group properties:

(i) Associativity: $(A \bullet B) \bullet C = A \bullet (B \bullet C)$ for all $A, B, C \in \mathbf{GL}_2(\mathbf{R})$

Since Matrix multiplication in general is associative.

(ii) Exist identity: $I_2 = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$ such that $A \bullet I = I \bullet A = A$ for all $A \in \mathbf{GL}_2(\mathbf{R})$

(iii) Exist Inverse for each $A \in \mathbf{GL}_2(\mathbf{R})$

Since for any element $A \in \mathbf{GL}_2(\mathbf{R})$ is of the form $\begin{bmatrix} a & b \\ c & d \end{bmatrix}$ whose determinat is non zero which guarantees the existence of inverse of the form

$$\begin{bmatrix} a & b \\ c & d \end{bmatrix}^{-1} = \frac{1}{ad-bc} \begin{bmatrix} d & -b \\ -c & a \end{bmatrix}$$

Note that this group is **Non commutative**, since matrix multiplication is not commutative.

For Example:

Suppose in $\mathbf{GL}_2(\mathbf{R})$, we have

$$A = \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix} \quad B = \begin{bmatrix} 5 & 6 \\ 7 & 8 \end{bmatrix}$$

$$A \bullet B = \begin{bmatrix} 19 & 22 \\ 43 & 50 \end{bmatrix}$$

$$B \bullet A = \begin{bmatrix} 23 & 34 \\ 31 & 46 \end{bmatrix}$$

So, $A \bullet B \neq B \bullet A$.

Some examples of finite groups are given below.

Examples

1. $(\mathbb{Z}_n, +)$ for any positive integer n

In particular say $n = 5$, The addition table of \mathbb{Z}_5 is given below:

+	0	1	2	3	4
0	0	1	2	3	4
1	1	2	3	4	0
2	2	3	4	0	1
3	3	4	0	1	2
4	4	0	1	2	3

All elements in the table are of \mathbb{Z}_5 . So addition is well defined on \mathbb{Z}_5 . From the table we can easily verify that it satisfies all the three properties of group. Here '0' plays the role of the identity. It also satisfies the commutative property. Therefore, $(\mathbb{Z}_5, +)$ is an Abelian group.

This type of table is known as Cayley's table. It can be also easily proved from the Cayley's table under multiplication (\bullet) that (\mathbb{Z}_5, \bullet) is also an Abelian group under multiplication where '1' plays the role of the identity. The order of these groups are 5.

But note that (\mathbb{Z}_4, \bullet) is not a group though $(\mathbb{Z}_4, +)$ is an Abelian group and order of this group is 4.

Here first we will define the integral power a^n of a as follows

$$a^0 = e$$

$$a^n = a * a^{n-1} \text{ if } n > 0$$

$$a^n = (a^{-1})^{-n} \text{ if } n < 0$$

In other words by a^n , we mean $a * a * \dots * a$ n -times in a group $(G, *)$. In this notation a^0 represents the identity of the group i.e. $a^0 = e$ and $a^{-n} = a^{-1} * a^{-1} * \dots * a^{-1}$ i.e. the binary operation performs n times a^{-1} .

Note that in case of additive group a^n is denoted by na , i.e. $na = a + a + \dots + a$, n times.

Theorem 2. Let $(G, *)$ be a group, $a, b \in G$ and $m, n \in \mathbb{Z}$. Then

- (i) $a^n * a^m = a^{n+m} = a^m * a^n$
- (ii) $(a^n)^m = a^{nm}$
- (iii) $a^{-n} = (a^n)^{-1}$
- (iv) $(a*b)^n = a^n * b^n$, if $(G, *)$ is commutative

Definition 2. Let $(G, *)$ be a group and $a \in G$. If there exists a positive integer n such that $a^n = e$, then the smallest positive integer is called the **order** of a .

If no such n exists then we say that a is of **infinite order**.

We denote the order of an element a of a group by $o(a)$. This one of the most important concept in group theory.

Theorem 3. The order of an element of a group divides the order of the group.

Example . Let $(\mathbb{Z}_6, +)$ is a finite group of order 6. The order of the elements 0,1,2,3,4,5 are 1, 6, 3, 2, 3, 6 respectively .

2.2.1.1. Subgroups

Some natural queries about the subsets of a group arises in the process of development of group theory. Main important question was whether subset of a group inherit the group structure or not. This section will discuss some of its basic concepts and examples.

Definition 1. Let $(G, *)$ and let H be a non-empty subset of G . Then H is called a subgroup of $(G, *)$, If H is closed under the binary operation $*$ and $(H, *)$ is a group.

Note that every group G has at least two subgroups , viz. $\{e\}$ and G itself. These two are called **trivial subgroups**, otherwise it is called H is a **nontrivial subgroup** of G .

Examples :

1. Let $(\mathbb{Z}, +)$ is a group & $H = \{2\mathbb{Z}\}$ set of all even numbers. H is a subset of \mathbb{Z} and it is closed under addition , since addition two even numbers is again an even number. It can be easily shown that $(H, +)$ is again a group . So H is a non trivial subgroup.

2. Let $(\mathbf{Z}_6, +)$ is a group & $H = \{ 0, 2, 4 \}$ is a proper subset of \mathbf{Z}_6 . H is a subgroup of \mathbf{Z}_6 , since H is closed under $+$, associative & commutative properties are hereditary, the identity element of H is '0' and inverse of 2 & 4 are 4 and 2 also belong to H . Hence, H is a nontrivial subgroup.

Note that where as groups $(\mathbf{Z}_5, +)$ & (\mathbf{Z}_5, \bullet) have no nontrivial subgroups.

Theorem 1. All subgroups of a group $(G, *)$ have the same identity element.

Theorem 2. Let G be a group and H be a non-empty subset of G . Then H is a subgroup of G if and only if $xy^{-1} \in H$ for all $x, y \in H$.

Note that If H be a non-empty finite subset of a group G , then H is a subgroup if and only if $xy \in H$ for all $x, y \in H$

Example: Let \mathbf{C}^* be a group and $H = \{1, i, -1, -i\}$. By the above result it can be easily shown that H is a subgroup.

2.2.1.2. Cyclic Groups

In this section we will briefly discuss about one of the important class of groups known as cyclic groups.

Let S be a subset of the group G . By $\langle S \rangle$, we mean the subgroup of G generated by S . So if $\langle S \rangle = G$ then S is called the set of generators for G . In particular, if the generator set is finite then G is called finitely generated. So for $a \in G$ we use the notation $\langle a \rangle$ to denote the subgroup of G generated by $\{a\}$.

Theorem 1. Let G be a group and $a \in G$. Then $\langle a \rangle = \{a^n \mid n \in \mathbf{Z}\}$

Note that in the group $(\mathbf{Z}, +)$, for $a \in \mathbf{Z}$, $\langle a \rangle = \{na \mid n \in \mathbf{Z}\}$

Definition 1. A group G is said to be **cyclic group** if there exists an element $a \in G$ such that $G = \langle a \rangle = \{a^n \mid n \in \mathbb{Z}\}$. Such an element $a \in G$ is called a **generator** of the cyclic group G .

Examples:

1. Group $(\mathbb{Z}, +)$ is a cyclic group since $\mathbb{Z} = \langle 1 \rangle$.
2. Group $(2\mathbb{Z}, +)$ is a cyclic group since it is generated by 2.
3. $(\mathbb{Z}_n, +)$ is a cyclic group since $\mathbb{Z}_n = \langle [1] \rangle$ (i.e. 1 in \mathbb{Z}_n).
4. Let $G = \{1, -1, i, -i\}$ be a group under multiplication of order 4. Here we can easily show that i & $-i$ both are generators of G .

Theorem 2. Every cyclic group is commutative.

Theorem 3. A finite group G is a cyclic group if and only if there exists an element $a \in G$ such that $o(a) = |G|$, i.e. $o(a)$ is equal to the order of the group G .

Theorem 4. Let $\langle a \rangle$ be a finite cyclic group. Then $o(a) = |\langle a \rangle|$.

For example $\langle 2 \rangle$ be a finite cyclic group under addition modulo 6. $\langle 2 \rangle = \{0, 2, 4\}$, so the order of the group is 3 and the order of 2 is 3.

Theorem 5. Let $G = \langle a \rangle$ be a cyclic group of order n . Then for any integer k where $1 \leq k \leq n$, a^k is a generator of G if and only if $(n, k) = 1$, i.e. k & n are co-prime.

It can be verified from example 4 given above. Here $G = \langle i \rangle$ and $(3, 4) = 1$ then $i^3 = -i$ is again a generator of G .

Theorem 6. Every subgroup of a cyclic group is cyclic.

Here again we can take for example $(\mathbb{Z}_6, +)$ and the subgroup $H = \{0, 2, 4\}$. Since we know $H = \langle 2 \rangle$ so again it is cyclic.

2.2.1.3. Permutation groups

A permutation of n different elements is nothing but an arrangements of those elements in any order. It plays a very important role in many areas. Hence study of its underlying algebraic structure is an important aspects of group theory.

Definition 1. Let A be a nonempty set. A Permutation of A is a bijective mapping of A onto itself.

Definition 2. A group $(G, *)$ is called a **permutation group** on a nonempty set A if the elements of G are some permutations of A and the operation $*$ is the composition of two mappings.

For example, let X be a nonempty set and let S_X be the set of all bijective functions of X onto itself. Then (S_X, \circ) is a group, where \circ is the composition two functions. Hence (S_X, \circ) is a **permutation group**.

Here our main interest lies in permutations of finite set. Let for any positive integer n , I_n denotes the finite set $\{1, 2, \dots, n\}$. The set of all permutations on I_n forms a group under composition of two functions denoted by ' \circ '. This group is called **symmetric group** of n elements and is denoted by S_n . It is also a very important class of groups.

It is easy to note that order of S_n is $n!$, i.e. $|S_n| = n!$. Generally we denote $\alpha \in S_n$ and represent either in two rows or cycle form. Two rows representation is quite convenient notation for carry out the composition. In this case second row represent the elements of 1st row under α .

For example, say $n = 3$ then $I_3 = \{1, 2, 3\}$ and say $\alpha(1) = 2, \alpha(2) = 3$ and $\alpha(3) = 1$ then we represent it as $\alpha = \begin{pmatrix} 1 & 2 & 3 \\ 2 & 3 & 1 \end{pmatrix}$.

As commonly follows $\alpha \circ \beta(x) = \alpha(\beta(x))$. Say for example $\alpha = \begin{pmatrix} 1 & 2 & 3 \\ 2 & 3 & 1 \end{pmatrix}$ & $\beta = \begin{pmatrix} 1 & 2 & 3 \\ 3 & 2 & 1 \end{pmatrix}$

So $\alpha \circ \beta = \begin{pmatrix} 1 & 2 & 3 \\ 1 & 3 & 2 \end{pmatrix}$ Sometimes we represent it by simply $\alpha\beta$

Example 1 . Here we consider the set $I_3=\{1,2,3\}$. There are total $6=3!$ Permutations on I_3 as there are 6 bijective functions of I_3 onto itself. They are namely

$e = \begin{pmatrix} 1 & 2 & 3 \\ 1 & 2 & 3 \end{pmatrix}$, identity permutation, $\tau_1 = \begin{pmatrix} 1 & 2 & 3 \\ 1 & 3 & 2 \end{pmatrix}$, $\tau_2 = \begin{pmatrix} 1 & 2 & 3 \\ 2 & 1 & 3 \end{pmatrix}$, $\tau_3 = \begin{pmatrix} 1 & 2 & 3 \\ 3 & 2 & 1 \end{pmatrix}$, $\sigma_1 = \begin{pmatrix} 1 & 2 & 3 \\ 2 & 3 & 1 \end{pmatrix}$ & $\sigma_2 = \begin{pmatrix} 1 & 2 & 3 \\ 3 & 1 & 2 \end{pmatrix}$. It forms a group under the binary operation i.e. composition of functions given in the Cayle's table below:

\circ	E	σ_1	σ_2	τ_1	τ_2	τ_3
e	E	σ_1	σ_2	τ_1	τ_2	τ_3
σ_1	σ_1	σ_2	E	τ_2	τ_3	τ_1
σ_2	σ_2	E	σ_1	τ_3	τ_1	τ_2
τ_1	τ_1	τ_3	τ_2	E	σ_2	σ_1
τ_2	τ_2	τ_1	τ_3	σ_1	E	σ_2
τ_3	τ_3	τ_2	τ_1	σ_2	σ_1	E

This is denoted by S_3 , symmetric group of order 6 and from the table one can easily verified that it is non commutative group and hence non cyclic = 3.

Definition 3. A permutation α on $I_n=\{1,2,...,n\}$ is called **k-cycle** or cycle of length k if there exist distinct elements i_1, \dots, i_k in I_n such that $\alpha(i_1) = i_2, \alpha(i_2) = i_3, \dots, \alpha(i_{k-1}) = i_k, \alpha(i_k) = i_1$ and $\alpha(x) = x$ for all $x \in I_n \setminus \{i_1, \dots, i_k\}$

In particular , a k-cycle with **k=2** is called **transposition**

Definition 4. Two cycles of S_n are said to be disjoint if there does not exist any common elements between two cycles

Theorem 1. Let α and β be any two permutations of S_n , then $\alpha\beta = \beta\alpha$

Theorem 2. Any nonidentity permutation $\alpha \in S_n (n \geq 2)$ can be expressed as a product of disjoint cycles of length ≥ 2 .

Theorem 3. Any cycle of length ≥ 2 is either a transposition or can be expressed as a product of transpositions.

Note that any cycle $(i_1, i_2, \dots, i_k) = (i_1 i_k)(i_1 i_{k-1}) \dots (i_1 i_2)$.

Say for example $(2\ 4\ 7) = (2\ 7)(2\ 4)$

Theorem 4. Any nonidentity permutation of $S_n (n \geq 2)$ is either a transposition or can be expressed as a product of transpositions.

So note that every permutation can be represented as a cycle or product of cycles

Say for example $\alpha = \begin{pmatrix} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 \\ 8 & 5 & 6 & 3 & 7 & 4 & 2 & 1 \end{pmatrix} \in S_8$

So the permutation α can be represented in cycle form which is as follows

$\alpha = (1\ 8)(2\ 5\ 7)(3\ 6\ 4)$, it verifies the theorem 2

Now by applying Theorem 3 we can show that α is a product of transpositions i.e.

$\alpha = (1\ 8)(2\ 7)(2\ 5)(3\ 4)(3\ 6)$. So Theorem 4 is also true.

Definition 4. A permutation $\alpha \in S_n$ is called an **even permutation** if α can be expressed as a product of an even number of 2-cycles and it is called an **odd permutation** if it is a 2-cycle or a product of odd numbers of 2-cycles.

Note that the set of all even permutations of S_n forms a group. This is called **alternating group** and denoted by A_n and $|A_n| = \frac{1}{2}|S_n|$

Also note that the **identity permutation** is an **even permutation**.

Theorem 3. Any permutation in S_n is either an odd or an even permutation but never both.

We will also note the very important property of a permutation.

Theorem 4. Let $n \geq 2$ and $\alpha \in S_n$ be a cycle. Then α is a k -cycle if and only if order of α is k .

Say for example in S_n order of $(1\ 2\ 3) = 3$ i.e $(1\ 2\ 3) (1\ 2\ 3) (1\ 2\ 3) = e$ the identity permutation. Where $(1\ 2\ 3) (1\ 2\ 3) \neq e$

Similarly $(1\ 2) (1\ 2) = e$

Theorem 5. Let $n \geq 2$ and $\alpha \in S_n$ and $\alpha = \alpha_1 \dots \alpha_k$, in other words α is a product of $\alpha_1 \dots \alpha_k$, disjoint cycles. If $o(\alpha_i) = n_i$ then $o(\alpha) = \text{lcm}(n_1, n_2, \dots, n_k)$.

The MatLab function to compute permutations of a given string is given in F4 and to compute combinations of a given string is in F5 in Appendix B.

2.2.2. Rings

Here we discuss briefly few concepts of algebraic structure [6,9] with two binary operations.

Definition 1. A ring is an ordered triple $(R, +, \cdot)$ such that R is a non empty set where $+$ (addition) and \cdot (multiplication) are two binary operations on R , satisfying the following axioms for all $a, b, c \in R$:

- (i) $a + b = b + a$, commutative under addition
- (ii) $a + (b + c) = (a + b) + c$, associative under addition
- (iii) There exists an element $0 \in R$ such that $a + 0 = 0 + a = a$, existence of additive identity
- (iv) For each $a \in R$, there exists $-a \in R$ such that $a + (-a) = 0$, existence of additive inverse
- (v) $a \cdot (b \cdot c) = (a \cdot b) \cdot c$, associative under multiplication
- (vi) $a \cdot (b + c) = a \cdot b + a \cdot c$ and $(b + c) \cdot a = b \cdot a + c \cdot a$ distributive laws

In other words a ring $(R, +, \cdot)$ is an Abelian group under addition and under multiplication it is associative and multiplication is distributive over addition.

Note that a ring R is said to be **commutative ring** if $a \cdot b = b \cdot a$ for all $a, b \in R$, i.e. commutative under multiplication.

R is said to be **ring with identity** if there exists an element $1 \in R$ such that $a \cdot 1 = 1 \cdot a = a$ for all $a \in R$, i.e. existence of multiplicative identity.

Note that the following identities are true in all rings:

- $0 \cdot a = a \cdot 0 = 0$
- $a \cdot (-b) = -(a \cdot b) = (-a) \cdot b$
- $(-a) \cdot (-b) = a \cdot b$

Examples :

1. Set of integers, rational numbers real numbers and complex numbers i.e. $\mathbf{Z, Q, R}$ & \mathbf{C} all are rings under usual addition and multiplication.
2. Let \mathbf{Z}_n be the set of integers modulo $n > 0$. It is a ring under addition & multiplication modulo n , It is a finite ring.

Note that \mathbf{Z}_n is a commutative ring and it is also a ring with identity.

3. Let $\mathbf{M}_2(\mathbf{R})$ – the set of 2×2 real matrices. $(\mathbf{M}_2(\mathbf{R}), +, \cdot)$, where $+$, \cdot are matrix addition and multiplication. Then it is easy to verify that it is a ring which has identity but non commutative since matrix multiplication is not commutative.
4. Let $\mathbf{2Z}$ (set of even integers), then $(\mathbf{2Z}, +, \cdot)$ is a ring which is not a ring with identity (unit ring) but it is commutative.

Definition 2. Let R be a ring with identity 1 and $1 \neq 0$. Then an element $a \in R$ is called **unit** or **invertible** if there exists an element $b \in R$ such that $ab = ba = 1$. b is called the inverse of a and it is denoted by a^{-1} . It is unique.

Theorem 1. Let R be a ring with identity 1 . Then the set of units of R forms a group under multiplication.

Definition 3. Let R be a ring. An element $a \in R$ is called a **Zero divisor** in R if $a \neq 0$ and there a non zero element $b \in R$ such that $ab = 0$ or $ba = 0$

It means this ring has some non-zero elements whose product is zero , the additive identity.

Say for example in the ring $M_2(\mathbb{R})$ $\begin{bmatrix} 1 & 0 \\ 0 & 0 \end{bmatrix}$, $\begin{bmatrix} 0 & 0 \\ 0 & 1 \end{bmatrix}$ are not equal to 0 but $\begin{bmatrix} 1 & 0 \\ 0 & 0 \end{bmatrix} \begin{bmatrix} 0 & 0 \\ 0 & 1 \end{bmatrix} = \begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix}$. So this ring has a zero divisor.

Also we can consider ring \mathbb{Z}_6 It can be seen very easily $3 \cdot 2 = 0$. $2 \cdot 3 = 0$. $4 \cdot 3 = 0$. Where 3, 2, 4 all are non zero elements of \mathbb{Z}_6 . So 2, 3, 4 are zero divisors in \mathbb{Z}_6 .

Definition 4. A commutative ring R with identity $1 \neq 0$ is called an **integral domain** if R has no zero divisors.

Examples.

1. \mathbb{Z} , \mathbb{R} , \mathbb{Q} , \mathbb{C} are Integral domain.
2. From above we can see \mathbb{Z}_6 , $M_2(\mathbb{R})$ are not integral domain.

Theorem 2. For any positive integer n , the ring \mathbb{Z}_n of all integers modulo n is an integral domain if and only if it is a prime integer.

Definition 5: - A ring R with identity $1 \neq 0$ is called a **division ring** if every non zero element of R has an inverse i.e. in other words they are unit.

Note that if R is a division ring , the set of all nonzero elements of R forms a multiplicative group.

Say for example \mathbb{Z}_p where p is a prime. Then it is a division ring but $M_n(\mathbb{R})$ is not a division ring because all non-zero matrices have no inverse.

2.2.3. Fields

This is a very important algebraic structure [8] used in cryptography.

Definition 1. A field is an ordered triple $(F, +, \cdot)$ where F is a non empty set and $+$, \cdot are two binary operations defined on F such that it is a commutative division ring.

So it is an additive abelian group and its non zero elements form a multiplicative commutative group and satisfies the distributive laws.

Examples.

1. **Q, R and C** are fields
2. **Z** is not a field.
3. **Z_p** is a field where p is prime integer.
4. **Z₆** is not a field.

Note that **Z_p** is a first example of finite field consists of p elements . It is also denoted by **F_p** and known as Galois field of order p.

The smallest finite field is **F₂ = Z₂** consists of two elements only under addition and multiplication modulo 2. It is a very important field for all digital communication and cryptography.

Definition 2. Let R be a ring . If there exists a positive integer n such that $na = 0$ for all $a \in R$ then the least such positive integer is called the **characteristic** of the ring R. if there exists no such positive integer n with the above property then R is said to be of **characteristic 0**

Theorem 1. A finite field has prime characteristic.

Theorem 2. The order of finite field is p^n for some prime p and positive integer n.

Theorem 3. For every finite field F_q the multiplicative group F_q^* ; of non-zero elements of F_q is cyclic.

Say for example **Z₅**, here **Z₅^{*} = {1, 2, 3, 4} = <2>**

Chapter 3.

Overview of Cryptography

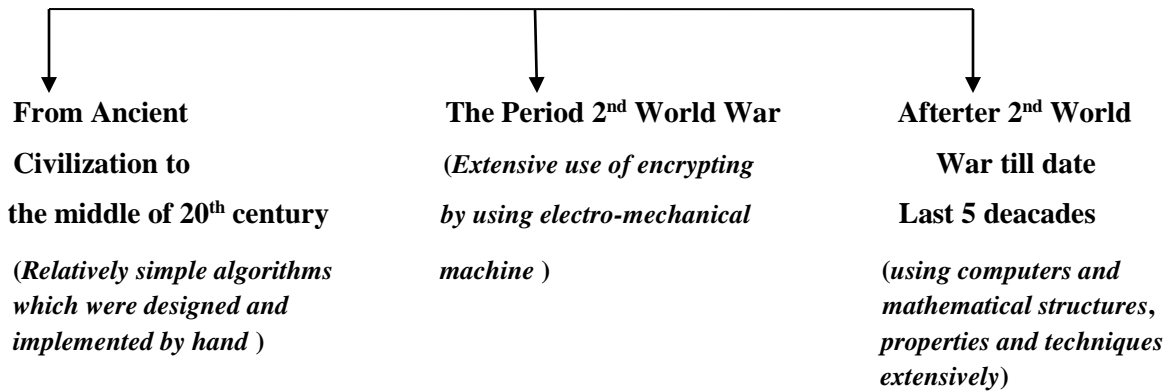
3.1. Introduction

This chapter deals with mainly the need, development and type of cryptography used for secure communication and information. This is basic survey work required for the project. At present the whole world is connected online through open networks. We can communicate with all across the globe using the open computer networks connected with personal computers. So the personal computers are also connected to open networks and there are threat of data privacy of one's own computer and communications through open channel. Today online communication is a part and parcel of the whole society. So the need for secure communication and its continuous development of strengthening its different aspects to handle the different threats and risks in the presence of adversaries both active and passive.. It helps to communicate confidentially through insecure channel by concealing its meaning to the unauthorized parties.

The fundamental building block of security is cryptography. From e-mail to mobile communications, from secure web access to digital cash, cryptography [6, 14, 16] is an essential part of today's information systems. As commerce and communications to move to computer networks, cryptography will play more and more important role.

3.2. Historical Background

Cryptography has a long and fascinating history of development. The history of the development of cryptography can be broadly divided into three phases as given below :



In earlier days it is mainly used in the private communication, art and religion and in Military communication. Cryptography was solely concerned with converting messages to unreadable groups of figure to protect the message's content during the time the message was being carried from one place to another. The earliest forms of cryptography were found in the cradle of civilization, which come from Egypt, Greece, and Rome.

As early as 1900 B.C., Egyptians hieroglyphs in a non-standard fashion, presumably to hide the meaning from those who were not authorized. The Greek's idea was to wrap a tape around a stick and write the message on the wound tape. When the tape would be unwounded, the writing would be meaningless. The roman method of cryptography was Caser Shift Cipher. It utilized the idea of shifting letters by an agreed upon numbers and thus writing the method using the letter shift. The famous Caesar cipher is based on 3-positions shift, that is, mathematically (considering the English 26 letters alphabet)

$$y = (x + 3) \bmod 26$$

The art and science of cryptography shoed no major changes or advancements until the Middle Ages. By the time, Western Europe started cryptography in one form or the other. Battisa Alberti is known as the "father of western cryptography" because of his polyalphabetic substitution.

Polyalphabetic substitution went through a variety of changes and is most notably attributed to Vigenere in 1586. It was considered to be practically impossible to be broken for almost 4 centuries. The Vigenere cipher consists of various Caesar ciphers in sequence with different shift values. To encipher, a table of alphabets was used, termed as "Vigenere

Table”. In 1918 Gilbert Vernam improved Vigenere ciphers. His work led to the “one time pad” which uses a keyword only once, and it proved to be nearly unbreakable.

On the due course of development of cryptography people began to understand that

- Complex ciphering could be obtained by concatenating a certain number of simple ciphering phases, and
- Ciphering operations could be no more by hand but with the help of machines

The wheel cipher of the wheels was named after the third U. S. President Jefferson. In the 2nd world war Enigma machine played an important role. It was used by Germans to communicate important military messages in a secure way. Enigma was a family of electro-chemical rotational machines.

The era of modern cryptography really begins with Claude Shannon, known as father of modern cryptography, for his work during 2nd world war on secure communications. He published two famous papers in 1949. He established theoretical basis for cryptography.

Now-a-days cryptography is based on Kerckhoff’s principle. Adversary knows the design completely except the secret key. The mid 1970’s saw two major public (non-secret) advances. First was the publication of the Data Encryption Standard (DES) in the US Federal Register on March 1975. The proposed DES was submitted by IBM at the invitation of National Bureau of Standards (now NIST) in an effort to develop secure electronic communication facilities for business such as banks and other large financial organizations. The DES officially replaced the Advanced Encryption Standard.

The second development in 1976 was perhaps even more important, a new paradigm in the history of cryptography. This was the publication of the new paper New Directions in Cryptography by Diffie and Hellman. It introduced a new method of distributing cryptographic keys which went far towards solving era of the fundamental problems of cryptography. Key distribution has become known as Diffie-Hellman Key Exchange.

This gave birth to a new class of cryptographic algorithms, asymmetric (public) algorithms. Before that, only symmetric (private) key algorithms were in use, in which only the sender and receiver know the key and they must keep it secret. In contrast, asymmetric key encryption uses a pair of mathematically related keys, each of which decrypts the encryption performed using the other. Generally these algorithms have additional property

that one of the paired keys cannot be deduced from the other by any known method other than trial and error. An algorithm of this kind is known as public key or asymmetric key system. In this type of algorithm only one key is needed per user.

3.3. Modern Cryptography

The common citizens of the world now understand the importance of cryptography and they need to study more about it.

Cryptography is the study of mathematical techniques [6] related to the aspects of information security such as hiding the information content, data integrity, authentication and data integration.

The main goal of modern cryptography is to provide:

- Privacy / confidentiality
- Authentication
- Integrity
- Non-repudiation

The cryptography can be divided broadly as follows :

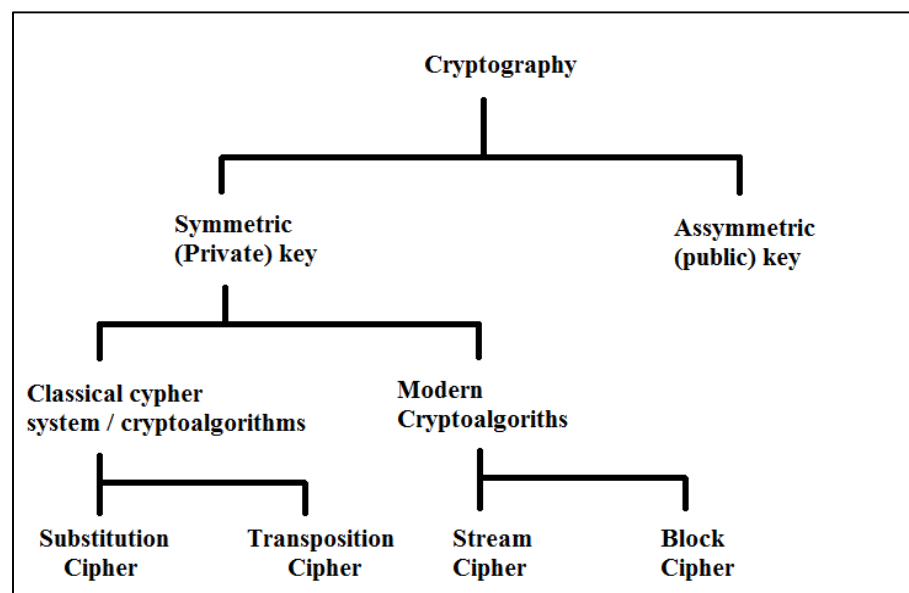


Figure. 1.Branch of Cryptography

Some of the basic terms / components used in cryptography are described below:

Plaintext: The original message which has to be secure.

Ciphertext: The transformed unintelligible message.

Key: Some critical information used by cryptography algorithm for transforming and intelligible into intelligible form.

Encryption / Decryption : The method of producing cipher text from plaintext (plaintext from cipher text) using the key.

Sender: The person who encrypts the plaintext and sends the cipher.

Receiver: The person who receives the cipher and decrypts it into plaintext .

Eavesdroppers / Adversary: The people who are not authenticated to get the information from the communication.

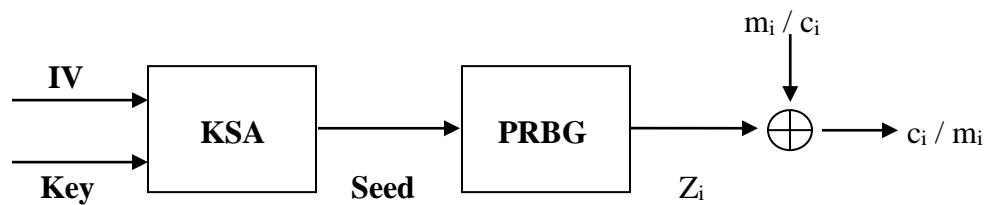
In **symmetric key cryptosystems**, a key is used for both encryption and decryption, though a message or group of messages can have a different key than others. A key management system is needed in symmetric key cryptosystems to use them securely. Each distinct pair of communicating parties must share a different key. This key is kept secret, though the encryption algorithm may not be kept secret.

Symmetric key cryptosystems are of two types, as defined below:

1. **Stream cipher:** - In this cipher, plaintext digits are combined with a pseudo-random cipher digit stream (keystream). In a stream cipher each plaintext binary digit is encrypted one at a time with the corresponding digit of the keystream, to give a digit of the ciphertext stream. One time pad is a perfect secret stream cipher. There are some well known stream ciphers viz. CBF, RC4, Edon 80 are stream ciphers.
2. **Block Cipher:** - In this cipher, a cryptographic key and algorithm are applied to an entire block of data (for example, 64 contiguous bits) at once as a group rather than to one bit at a time. Block cipher design principle is mainly based on two type of structure viz. Feistel structure & SP Network. DES, AES, IDEA are some examples of block cipher.

In this project, we work on stream cipher which is a part of symmetric (or private) key cryptography. Stream cipher design principle is mainly based on Pseudo Random Bit Generator (**PRBG**), which generates required length keystream sequence from the smaller

size of seed(key) and then mainly XOR function is applied to each keystream bit and message bit. Now a days the seed for PRBG is generated using Key Scheduling algorithm (KSA) by using initial vector (IV) and a key. The schematic diagram of stream cipher is given below:



In **public key crytosystems**, there are two parts of the key, one is for encryption (public part) and other is for decryption (private part). The public key are distributed freely, but its paired private key must remain secret. In a public key encryption system, the public key is used for encryption, while the private key is used for decryption. Public key cryptography can also be used for implementing digital signatures. Public key cryptography is manly based on the complexity of two computationally hard problems: (i) Euler factorization, for example RSA, and (ii) Discrete log problem, for example Diffie-Hellman and DSA.

Chapter 4.

Quasigroup Based Cryptography

4.1. Introduction

Till date standard cryptographic primitives and schemes are commonly based on associative and commutative algebraic structures. In eighties part of European crypto community visualized the importance of non-associative and non-commutative algebraic structures in cryptology. In 2001 first Denes & Keedwell [3] announced it as the new era in cryptology. In this direction some non-associative and non-commutative algebraic structures evolved theoretically which are having good potention to use in cryptology. They are viz. quasigroups, n-ary quasigroups [1, 2], loops etc.

In this chapter we will briefly discuss the basic definitions, concepts, properties of quasigroups and different string transformations and cryptographic primitives [19] and schemes based on these transformations. Here in this project we mainly deals with quasigroups of order 4. There are lots of advantages to design the cryptographic primitives and schemes by considering one or more than one quasigroups of small order. From this point of view lots of schemes are designed based on combinations of quasigroups of order 4.

4.2. Quasigroups

Quasigroups may be defined both from combinatorial and algebraic point of view, known as combinatorial quasigroups and equational (algebraic) quasigroups respectively.

Definition 1: A **(combinatorial) quasigroup** (Q, \cdot) is a groupoid consisting of elements of Q with repect to a binary operation \cdot such that for all $a, b \in Q$ there exists unique

$x, y \in Q$ for which it satisfies the identities $a \cdot x = b$ and $y \cdot a = b$. In other words, the equations $a \cdot x = b$ and $y \cdot a = b$, for any given $a, b \in Q$ have unique solutions x, y .

Any finite quasigroup of order m can be represented by Cayley's table. The main body of the table can be associated with Latin square of order m , which is an arrangement of m elements in $m \times m$ square matrix whose each row and column contains all m elements only once. The existence and uniqueness of the solution of the identities of quasigroup give the guarantee that all m elements will appear in the each row & column and it will appear only once respectively. Conversely each Latin square may be ordered to yield the Cayley's table of the finite quasigroup of same order.

Example. Let (Q, \cdot) be a quasigroup of order $m = 4$ represented by Cayley's table given below :

\cdot	1	2	3	4
1	1	3	2	4
2	3	4	1	2
3	2	1	4	3
4	4	2	3	1

The body of the Cayley's table

1	3	2	4
3	4	1	2
2	1	4	3
4	2	3	1

is a Latin square of order 4.

Conversely given the Latin square of order 4 as above can be bordered to yield the binary operation ' \cdot ' table of a quasigroup (Q, \cdot) given in the example of same order 4.

Definition 2 : An (equational / algebraic) quasigroup $(Q, \bullet, /, \backslash)$ is defined as a set Q closed under three binary operations \bullet (multiplication), $/$ (rightdivision) and \backslash (leftdivision), satisfying the following identities

1. $(y/x) \cdot x = y$
2. $(y \cdot x)/x = y$
3. $x \cdot (x \backslash y) = y$
4. $x \backslash (x \cdot y) = y$

From these four identities, following two more identities can also be derived

5. $x/(y \backslash x) = y$
6. $(x / y) \backslash x = y$

It is easy to prove that if $(Q, \bullet, /, \backslash)$ is an equational quasigroup then (Q, \bullet) is a combinatorial quasigroup. This definition of quasigroups is given from universal (generalized) algebraic point of view.

Conversely, suppose that (Q, \bullet) is a combinatorial quasigroup. For given elements $x, y \in Q$, define $x \backslash y$ as the unique solution of (3), and y/x as the unique solution of (1) in the Definition 2. It defines the binary operations $/$ and \backslash on Q that make $(Q, \bullet, /, \backslash)$ an equational quasigroup. It is to be noted that $(Q, /)$ and (Q, \backslash) are also quasigroups and hence Latin squares. Thus, it is usually not necessary to distinguish between the concepts of combinatorial and equational quasigroup of finite order. They are generally referred as simply quasigroups.

The algebraic definition of quasigroup means that they form a variety and thus we can study them by the methods and concepts of generalized (universal) algebras. So we can define say **Subquasigroup** as a subset P of a quasigroup Q which is closed under these three operations. We can generalize almost all concepts of algebraic structures.

The general algebraic representation of equational quasigroups is as follows:
 $(Q, \alpha, \alpha_1, \alpha_2)$ is an algebra where $\alpha, \alpha_1, \alpha_2$ are three binary operations satisfying the following identities

$$\left. \begin{aligned} \alpha(\alpha_1(x_1, x_2), x_2) &= x_1 = \alpha_1(\alpha(x_1, x_2), x_2) \\ \text{and } \alpha(x_1, \alpha_2(x_1, x_2)) &= x_2 = \alpha_2(x_1, \alpha(x_1, x_2)) \end{aligned} \right\} (i)$$

So we can now even in a position to give generalized definitions for n -ary quasigroups ($n \geq 2$) and other algebraic concepts of quasigroups.

Enumeration of quasigroups

Total number quasigroups of finite order plays an important role to make ciphers based on it secure [15, 18]. Here, we briefly discuss how to calculate the total number of quasigroups of finite order m . First we define reduced n -dimensional Latin square of order m .

Definition 3: Introduce an ordering in $Q = \{x_1, x_2, \dots, x_m\}$. An **n -dimensional Latin square** of order m is said to be **reduced** if it has identity permutations in 1st row & 1st column.

We denote the set of all reduced *Latin square* of order m by R_m and the set of all quasigroups of order m by Q_m . The total number of quasigroups of order m is given by $|Q_m| = m!(m-1)!|R_m|$. It increases asymptotically as m increases. In the following table for example we have shown some cases of m to give an idea.

M	$ R_m $	$ Q_m $
4	4	576
5	56	161280
6	9408	812851200

Table 1. *Number of reduced Latin squares and quasigroups with various m*

It gives an idea how it can increase the complexity in cryptographic schemes.

4.3. Lexicographical Arrangement of quasigroups of order 4

Definition 1. Suppose $\{A_1, A_2, \dots, A_n\}$ is an n-tuple of sets, with respective total orderings $\{<_1, <_2, \dots, <_n\}$ the lexicographical (dictionary) ordering $\{<_d\}$ of $\{A_1 \times A_2 \times \dots \times A_n\}$ is then

$$(a_1, a_2, \dots, a_n) <_d (b_1, b_2, \dots, b_n) \equiv (\exists m > 0)(\forall i < m)(a_i = b_i) \wedge (a_m <_m b_m)$$

That is, if one of the terms $a_m <_m b_m$ and all the preceding terms are equal.

The lexicographical order is very useful for searching and also it has advantage to identified with no so that both the party have the lexicographical arrangements of quasigroups order 4 , then by communicating the number only both will pick the same quasigroup. Basically it has the advantage in the implementation.

Here we have written an algorithm which will arrange set of all quasigroups of order 4 i.e. 576 quasigroups in a lexicographical order. The algorithm is implemented in both Matlab and C++.

Algorithm (MATLAB) – To arrange given list of all quasigroups of order 4 in lexicographical order :

1. Start.
2. Input arr=input array.
3. Store r=number of rows (number of strings); limit=number of letters in each string.
4. Apply selection sort on the array.
i=1 (i will run up to r-1)
5. j = i+1 (j will run up to r)
6. k=limit (k will run from limit to 1).
7. if int64(arr(i,k)) > int64(arr(j,k))
 - 7.1.Exchange whole strings (step 9.1 to step 9.3).
t = arr(i,:);
 - 7.2.arr(i,:)=arr(j,:);

```

7.3.arr(j,:)=t;
8. k=k-1;
9. if k!=1; go to step 7.
10. j=j+1;
11. if j!=r; go to step 6.
12. i=i+1;
13. if i!=r-1; go to step 5.
14. The array arr contains final output – The sorted strings in each row;
lexicographically.
15. Stop.

```

We also implemented the program to arrange quasigroups in lexicographical order in C++ also.

Algorithm (C++) – To arrange given list of quasigroups of order 4 in lexicographical order

```

1. Start
2. limit = 4 (the characters used are total 4 – 1,2,3,4).
3. Make queues (heads and tails for queues), total number equal to limit.
4. data[] = string type array of all strings.
5. r = number of rows (total number of strings to be sorted).
6. c = number of columns (Length of the biggest string).
7. j = c-1
8. i = 0
9. st = data[i]
10. if length of st is less than j+1, then
    enqueue st to queue number (limit-1)
    else, then
    enqueue st to queue number (st[j]-49)
11. i++
12. if i is less than r, then
    go to step 9.

```

13. dequeue the strings of queue number 1, 2, 3, and 4 serially to data[], overwriting its contents from the starting.
14. j--
15. if j is greater than, or equal to zero 0, then
go to step 8.
16. The contents of data[] are the required sorted strings (quasigroups of order 4).
17. Stop.

The software arranges the set of all quasigroups of order 4 in lexicographical order and given the number the it also show the quasigroup against that number out of 576 quasigroups. The code of the MatLab program for lexicographical order denoted by P8 is given in the Appendix-B and the C++ code is in P9 in Appendix-B.

4.4. Quasigroup Based String Transformations

The quasigroup based string transformations [10, 11, 12, 13, 15] play a very important role in the quasigroup based cryptography. First we will discuss here some basic definitions and generalization of this concepts. Mostly the quasigroup string transformations transform a given string in other string with equal length say n . Let (Q, \bullet) be a finite quasigroup. Consider Q as an alphabet set and $Q^+ = \{x_1 \dots x_n \mid x_i \in Q, n \leq 1\}$

For a fixed letter $l \in Q$ (called a leader) **elementary left quasigroup (string) transformations** $e_l, d_l: Q^+ \rightarrow Q^+$ are defined [] as follows:

$$e_l(x_1 x_2 \dots x_n) = (z_1 z_2 \dots z_n) \Leftrightarrow z_j = z_{j-1} \bullet x_j, 1 \leq j \leq n, \text{ where } z_0 = l$$

$$d_l(z_1 z_2 \dots z_n) = x_1 x_2 \dots x_n \Leftrightarrow x_j = z_{j-1} \backslash z_j, 1 \leq j \leq n$$

Here e_l and d_l are permutations and $e_l \circ d_l = d_l \circ e_l = I$ identity permutation on Q

Similarly we can define right transformations (e_l' and d_l') based on Q .

Let $\bullet_1, \bullet_2, \dots, \bullet_s$ be quasigroup operations defined on the set Q and l_1, l_2, \dots, l_s where $1 \leq i \leq s$.

The **composite quasigroup left transformation** is defined by $E = E_{l_s, l_{s-1}, \dots, l_1}^s = e_{l_s} \circ e_{l_{s-1}} \circ \dots \circ e_{l_1}$ & $D = D_{l_1, l_2, \dots, l_s}^s = d_{l_1} \circ e_{l_2} \circ \dots \circ e_{l_s}$ so that $E \circ D = D \circ E = I$ identity

permutation on Q . Here we note that all the operations may not be distinct and all leaders may not be different. All these cases based on fundamental cryptographic schemes are given which we have implemented and some experiments are carried out in our project.

4.5. Cryptographic Schemes Based on Quasigroup String Transformation

Quasigroups can be very useful for cryptographic purposes, mainly because there is a huge number of quasigroup operations on a given finite set, and it is easy to define the encoding and decoding functions by using the quasigroup operations as well.

4.5.1. Cryptographic Schemes based on Left Quasigroup String Transformation

Here we have given two schemes which are discussed below:

4.5.1.1. Cryptographic scheme based on left quasigroup transformation where leader is fixed

A quasigroup (Q, \bullet) and a fixed leader $l \in Q$ is given. Let the plaintext (message) $M = m_1 m_2 m_3 \dots m_n$ of length n . The encryption function denoted by e_l is given below to generate [7]

cipher $C = c_1 c_2 c_3 \dots c_n$ of M as follows:

$$e_l(M) = \begin{cases} c_1 = l \bullet m_1 \\ c_i = c_{i-1} \bullet m_i \text{ for } 2 \leq i \leq n \end{cases}$$

Where, \bullet is the effective quasigroup operation.

Algorithm:

1. Start.
2. Input x =message and arr =quasigroup matrix.
3. Input a =leader.

First derive corresponding left Inverse of Quasigroup (Q, \cdot) which is denoted by (Q, \backslash) :

\	1	2	3	4
1	4	1	2	3
2	1	2	3	4
3	2	3	4	1
4	3	4	1	2

Input (ciphertext): 4
4

[illegible]

4.5.1.2. Cryptographic scheme based on left quasigroup transformation where leader is based on key

A quasigroup (Q, \bullet) and a key stream $K = k_1 k_2 k_3 \dots k_n$ is given of required length of message string. Let $M = m_1 m_2 m_3 \dots m_n$ be the message string. The cipher $C = c_1 c_2 c_3 \dots c_n$ will be generated using the following method.

$$e_{l(k)} = \begin{cases} c_1 = l_{k_1} \cdot m_1 \\ c_i = c_{i-1} \cdot (l_{k_i} \cdot m_i) \text{ for } 2 \leq i \leq n \end{cases}$$

Where, \bullet is the quasigroup operation and leader is k_i denoted by l_{k_i} for $1 \leq i \leq n$.

Algorithm:

1. Start.
2. Input m=message, arr=quasigroup matrix and k=key.
3. i=1. (i will run upto then length of message)
4. If i == 1, then c(i) = arr (k(i),m(i));
- Otherwise

```
Val = arr(k(i),m(i));
c(i) = arr(c(i-1),val);
```

5. $i++$;
6. If $i \leq \text{length of message } m$, then go to step 4.
7. c is the cipher text required.
8. Stop.

The MatLab code for the above given algorithm is given in P7 in Appendix-B.

The corresponding decryption method $d_{l(k)}$ is defined as follows:

$$d_{l(k)} = \begin{cases} m_1 = l_{k_1} \setminus c_1 \\ m_i = l_{k_i} \setminus (c_{i-1} \setminus c_i) \text{ for } 2 \leq i \leq r \end{cases}$$

Where ‘\’ is the left inverse operation of the given quasigroup (Q, \bullet) . It can be easily verified that $e_{l(k)} \circ d_{l(k)} = d_{l(k)} \circ e_{l(k)} = I$.

Note that Similarly we can define the scheme based on right quasigroup transformation.

Example:

Quasigroup (Q, \bullet) :

Encryption:

•	1	2	3	4
1	1	2	3	4
2	2	3	4	1
3	3	4	1	2
4	4	1	2	3

Plaintext: 1

Key: 2 4 3 4 2 1 4 3 3 1 2 2 2 2 4 1 2 2 4 1 2 2 4 1 2 2 1 1 4 3 4 2 1 1 1 1 4 1 2 1

Generated Ciphertext: 2 1 3 2 3 3 2 4 2 2 3 4 1 2 1 1 2 3 2 2 3 4 3 3 4 1 1 1 4 2 1 2 2 2 2 2 1
[2 2

Decryption:

Left Inverse of Quasigroup (Q, \setminus) :

\	1	2	3	4
1	1	4	3	2
2	2	1	4	3
3	3	2	1	4
4	4	3	2	1

Input ciphertext: 2 1 3 2 3 3 2 4 2 2 3 4 1 2 1 1 2 3 2 2 3 4 3 3 4 1 1 1 4 2 1 2 2 2 2 2 1 1 2 2

Key: 2 4 3 4 2 1 4 3 3 1 2 2 2 2 4 1 2 2 4 1 2 2 4 1 2 2 1 1 4 3 4 2 1 1 1 1 4 1 2 1

Generated plaintext: 1
1 1

4.5.2. Iterated cryptographic schemes based on composite left quasigroup transformation

4.5.2.1. Iterated cryptographic scheme based on composite left quasigroup transformation where leader is fixed

A quasigroup (Q, \bullet) and a fixed leader $l \in Q$ is given. Say a message $M = m_1 \ m_2 \ m_3 \ \dots \ m_n$ is to be encrypted.

Let the generated cipher $C = c_1 c_2 c_3 \dots c_n$. We follow the following method.

$$E_{l...l}^s(M) = e_l^s \circ e_l^{s-1} \circ \dots e_l^1(M) = e_l^s(e_l^{s-1}(\dots(e_l^1(M))\dots))$$

This the s- times iterative encryption scheme based on composite left quasigroup transformation. Corresponding decryption function will be as follows :

$$D_{l \dots l}^s(C) = d_l^1 \circ d_l^2 \circ \dots d_l^s(C) = d_l^1(d_l^2(\dots(d_l^s(C)) \dots))$$

Here we have implemented this scheme for $s=4$ and l be any arbitrary but fixed element of the quasigroup.

Alorithm:

2. s=2-13311331133113311331133113311331133113311331
3. s=3-14121412141214121412141214121412141214121412
4. s=4-24312431243124312431243124312431243124312431

Decryption:

Left Inverse of Quasigroup (Q, \setminus) :

•	1	2	3	4
1	4	1	2	3
2	1	2	3	4
3	2	3	4	1
4	3	4	1	2

Leader: 2

Ciphertext: 2 4 3 1 2 4 3 1 2 4 3 1 2 4 3 1 2 4 3 1 2 4 3 1 2 4 3 1 2 4 3 1

Plaintext:

- [illegible]

4.5.2.2. Iterated cryptographic scheme by composite left transformation where leader is variable and is based on iteration.

A quasigroup (Q, \bullet) and a fixed leader $l \in Q$ is given. Say a message $M = m_1 m_2 m_3 \dots m_n$ is to be encrypted.

Let the generated cipher $C = c_1 c_2 c_3 \dots c_n$. We follow the following method.

$$E_{l(s)l(s-1) \dots l(1)}(M) = \begin{cases} c_1 = e_{l(1)}^1; s = 1 \\ c_i = e_{l(s)}^s \cdot e_{l(s-1)}^{s-1} \cdot \dots \cdot e_{l(1)}^1; 2 \leq s \leq n \end{cases}$$

Where, $l(s) = ((s * s) \% 4) + 1$, and \bullet is the effective quasigroup operation.

We will repeat this encryption iteratively, on the cipher generated at each step.

The decryption scheme $D_{l(1) \dots l(s)}$ will be defined as above.

Algorithm:

1. Start.
2. Input x =message and arr =quasigroup matrix.
3. $s=1$. (s will decide the number of iterations to be performed)
4. $a = \text{mod}((s * s), 4) + 1$; (this formula can be changed, but it should be reversible)
5. $i=1$. (i will run upto the length of the message)
6. if $i == 1$ then $c(i) = arr(x(i), a)$;
else then $c(i) = arr(x(i), c(i-1))$;
7. $i++$;
8. if $i \leq \text{length of message } x$, then go to step 6.
9. $s++$;
10. $x=c$;
11. if $s \leq 4$, then go to step 5. (the number **10** can be changed to change the required number of iterations)
12. c is the cipher required.
13. Stop.

The MatLab code for this algorithm is given in P3 in Appendix-B.

Note that similarly we can define cryptographic scheme based on right transformation.

Example:

Encryption:

Quasigroup (Q, \bullet) :

\bullet	1	2	3	4
1	2	3	4	1
2	1	2	3	4
3	4	1	2	3
4	3	4	1	2

Leader formula: $\text{mod}((s * s), 4) + 1$ (s is the iteration index of loop).

Plaintext: 4

Ciphertext:

1. 4 2 4 2 4 2 4 2 4 2 4 2 4 2 4 2 4 2 4 2 4 2 4 2 4 2 4 2 4 2 4 2
2. 1 3 3 1 1 3 3 1 1 3 3 1 1 3 3 1 1 3 3 1 1 3 3 1 1 3 3 1 1 3 3 1
3. 1 4 1 2 1 4 1 2 1 4 1 2 1 4 1 2 1 4 1 2 1 4 1 2 1 4 1 2 1 4 1 2
4. 2 4 3 1 2 4 3 1 2 4 3 1 2 4 3 1 2 4 3 1 2 4 3 1 2 4 3 1 2 4 3 1

Decryption:

Left Inverse of Quasigroup (Q, \):

\	1	2	3	4
1	4	1	2	3
2	1	2	3	4
3	2	3	4	1
4	3	4	1	2

Leader formula: $\text{mod}((s * s), 4) + 1$ (s is the iteration index of loop).

Ciphertext: 2 4 3 1 2 4 3 1 2 4 3 1 2 4 3 1 2 4 3 1 2 4 3 1 2 4 3 1 2 4 3 1

Plain Text:

1. 1 4 1 2 1 4 1 2 1 4 1 2 1 4 1 2 1 4 1 2 1 4 1 2 1 4 1 2 1 4 1 2
2. 1 3 3 1 1 3 3 1 1 3 3 1 1 3 3 1 1 3 3 1 1 3 3 1 1 3 3 1 1 3 3 1
3. 4 2 4 2 4 2 4 2 4 2 4 2 4 2 4 2 4 2 4 2 4 2 4 2 4 2 4 2 4 2 4 2
4. 4

4.5.3. Generalized iterated cryptographic scheme based on composite left quasigroup transformation

In this generalized cryptographic scheme, any four quasigroups of order 4 are chosen to apply the string transformation where quasigroup operation and leader will be based on

key. If the message is of length n , the key has to be of length $2n$. Here, let message $M = m_1 m_2 \dots m_n$, and the key is $K = k_1 k_2 \dots k_n k_{n+1} \dots k_{2n}$.

For each n th message element, 2 elements of the key will be used, viz. k_{2n-1} and k_{2n} . The first element, i.e. k_{2n-1} , will be used to choose the working quasigroup and the second element, i.e. k_n , will be used to choose the leader. Both these elements are of 2 bits. This is because we use four quasigroups, so we require 2 bits to choose one of them, and our quasigroups are of order 4, so we require 2 bits to choose leader.

The bit size of these elements can be varied if the total number of quasigroups is changed, or the order of those quasigroups is changed.

The cipher $C = c_1 c_2 c_3 \dots c_n$ is generated using the following method.

$$c = E_{k1 k2}(M)$$

Where E stands for encryption.

Its MatLab code is given in P10 in Appendix-B.

Example:

Encryption:

The four quasigroups:

1.

•	1	2	3	4
1	1	2	3	4
2	2	1	4	3
3	3	4	1	2
4	4	3	2	1

2.

•	1	2	3	4
1	1	2	3	4
2	2	1	4	3
3	3	4	2	1
4	4	3	1	2

3.

•	1	2	3	4
1	1	2	3	4
2	2	1	4	3
3	4	3	1	2
4	3	4	2	1

4.

•	1	2	3	4
1	1	2	3	4
2	2	1	4	3
3	4	3	2	1
4	3	4	1	2

Message: 4 2 1 3 4 2 2 3 3 3 4 3 4 2 1 3 4 2 2 2 2 2 4 1 3 3 2 1 2 1 4 4 2 1 3 3 2 2 4 3

Key: 3 3 2 2 4 3 4 3 1 4 3 1 3 2 2 1 3 1 3 1 2 2 3 4 2 2 1 3 3 1 3 2 1 4 1 1 2 4 2 1 1 1 1 4 3
1 4 3 2 3 1 2 2 2 1 2 3 3 4 3 1 1 4 1 1 1 4 3 2 4 3 3 1 3 2 4 3 4 3

Generated Cipher: 3 2 2 4 4 1 3 4 1 4 2 3 3 3 2 4 2 3 4 3 2 1 1 3 1 3 4 4 1 4 2 1 1 1 4 4 2 2 2
1

Decryption:

The four quasigroups:

1.

\	1	2	3	4
1	1	2	3	4
2	2	1	4	3
3	3	4	1	2
4	4	3	2	1

2.

\	1	2	3	4
1	1	2	4	3
2	2	1	3	4
3	3	4	1	2
4	4	3	2	1

3.

\	1	2	3	4
1	1	2	3	4
2	2	1	4	3
3	3	4	2	1
4	4	3	1	2

4.

\	1	2	3	4
1	1	2	4	3
2	2	1	3	4
3	3	4	2	1
4	4	3	1	2

Input Cipher: 3 2 2 4 4 1 3 4 1 4 2 3 3 3 2 4 2 3 4 3 2 1 1 3 1 3 4 4 1 4 2 1 1 1 4 4 2 2 2 1

Key: 3 3 2 2 4 3 4 3 1 4 3 1 3 2 2 1 3 1 3 1 2 2 3 4 2 2 1 3 3 1 3 2 1 4 1 1 2 4 2 1 1 1 1 4 3
1 4 3 2 3 1 2 2 2 1 2 3 3 4 3 1 1 4 1 1 1 4 3 2 4 3 3 1 3 2 4 3 4 3

Outputplaintext: 4 2 1 3 4 2 2 3 3 3 4 3 4 2 1 3 4 2 2 2 2 2 4 1 3 3 2 1 2 1 4 4 2 1 3 3 2 2 4 3

Chapter 5.

Experiments and Observations

Experiment 1.

We have taken ciphers of 40 bits and subjected them to encryption algorithm

$$e_l(M) = \begin{cases} c_1 = l \bullet m_1 \\ c_i = c_{i-1} \bullet m_i; \text{ for } 2 \leq i \leq n \end{cases}$$

where,

$$m = \{m_1 \ m_2 \ m_3 \ \dots\} \quad \text{(message, information string)}$$
$$c = \{c_1 \ c_2 \ c_3 \\} \quad \text{(cipher string)}$$

1 (leader key)

- (operation)

The algorithm of this method is discussed in section 4.5.1.1. The MatLab code for this method is given in P1 in Appendix-B.

We found that a pattern repeats itself. The pattern depends on quasigroup used, leader and information string. If $l^2 = l$, & $l \bullet m_1 = l$, then the message will all be equal to l if all $m_i = m_1$.

Example 1:

Quasigroup (Q, \bullet) : -

•	1	2	3	4
1	1	2	3	4
2	2	3	4	1
3	3	4	1	2
4	4	1	2	3

Leader: 2

Plain text: 4.

Example:

Quasigroup (Q, •): -

•	1	2	3	4
1	1	3	2	4
2	3	2	4	1
3	2	4	1	3
4	4	1	3	2

Leader: 1

Plain text: 2
2 2 2 2 2 2

Cipher:

1. 3 1 2 4 3 1 2 4 3 1 2 4 3 1 2 4 3 1 2 4 3 1 2 4 3 1 2 4 3 1 2 4 3 1
2 4 3 1 2 4
2. 2 3 1 1 3 2 4 4 4 4 3 3 1 1 2 2 2 3 1 1 3 2 4 4 4 4 3 3 1 1 2 2 2 3 1 1 3 2 4 4 4 4
3 3 1 1 2 2
3. 3 1 1 1 3 1 1 1 1 1 3 1 1 1 2 4 3 1 1 1 3 1 1 1 1 1 3 1 1 1 2 4 3 1 1 1 3 1 1 1 1 1
3 1 1 1 2 4
4. 2 3 2 3 1 1 1 1 1 1 3 2 3 2 4 4 4 4 4 4 4 4 4 4 4 4 4 4 3 3 1 1 1 1 3 2 3 2 3 2
2 3 2 3 1 1

Experiment 3.

We have taken ciphers of 40 and 80 bits and subjected them to the following encryption algorithm.

$$E_{l(s)l(s-1) \dots l(1)}(M) = \begin{cases} c_1 = e_{l(1)}^1; s = 1 \\ c_i = e_{l(s)}^s \cdot e_{l(s-1)}^{s-1} \cdot \dots \cdot e_{l(1)}^1; 2 \leq s \leq n \end{cases}$$

The algorithm of this method is discussed in section 4.5.2.2. The MatLab code for this algorithm is given in P3 in Appendix-B.

Our observations are as follows.

$$e_{l(k)} = \begin{cases} c_1 = l_{k_1} \cdot m_1 \\ c_i = c_{i-1} \cdot (l_{k_i} \cdot m_i) \text{ for } 2 \leq i \leq n \end{cases}$$

where,

$m = \{m_1, m_2, m_3, \dots\}$ (message, bit string of 80 bits)

$k = \{l_{k_1}, l_{k_2}, \dots, l_{k_n}\}$ (key, bit string of 80 bits)

$c = \{c_1, c_2, c_3, \dots\}$ (cipher string)

• (operation)

The algorithm of this method is discussed in the section 4.5.1.2. The MatLab code for the above given algorithm is given in P7 in Appendix-B.

Example:

The message value string:

3 1 0 2 3 1 1 2 2 2 3 2 3 1 0 2 3 1 1 1 1 1 3 0 2 2 1 0 1 0 3 3 1 0 2 2
1 1 3 2

The key value string:

1 3 2 3 1 0 3 2 2 0 1 1 1 1 3 0 1 1 3 0 1 1 3 0 1 1 0 0 3 2 3 1 0 0 0 0
3 0 1 0

The cipher value string

0 0 2 3 3 0 0 0 0 2 2 1 1 3 2 0 0 2 2 3 1 3 1 1 0 3 0 0 0 2 0 0 1 1 3 1
1 2 2 0

Experiment 5.

We have taken ciphers of 80 bits and subjected them to the following encryption algorithm. We took 4 different quasigroups to do this.

$$c = E_{k1, k2}(m)$$

where,

$m = \{m_1 m_2 m_3 \dots\}$ (message, bit string of 80 bits)
 $k = \{k_1 k_2 k_3 \dots\}$ (key, bit string of 160 bits)
 $c = \{c_1 c_2 c_3 \dots\}$ (cipher string)
 E (encryption process)
 k1 (quasigroup selector bits – 2 bit)
 k2 (leader bits – 2 bit)

The algorithm of this method is discussed in section 4.5.3. The MatLab code for the above given algorithm is given in P10 in Appendix-B.

Example:

Given four quasi groups:

1.

•	1	2	3	4
1	1	3	2	4
2	3	2	4	1
3	2	4	1	3
4	4	1	3	2

2.

•	1	2	3	4
1	2	4	1	3
2	1	2	3	4
3	3	1	4	2
4	4	3	2	1

3.

•	1	2	3	4
1	3	2	1	4
2	2	3	4	1

3	4	1	3	2
4	1	4	2	3

4.

•	1	2	3	4
1	4	3	2	1
2	2	1	4	3
3	1	4	3	2
4	3	2	1	4

The message value string:

3 1 0 2 3 1 1 2 2 2 3 2 3 1 0 2 3 1 1 1 1 1 3 0 2 2 1 0 1 0 3 3 1 0 2 2
1 1 3 2

The key value string:

0 0 0 0 1 1 2 2 0 3 0 3 0 2 0 1 2 3 2 3 3 3 0 0 2 0 1 1 3 0 0 2 2 3 3 0
1 2 0 1 0 1 2 2 3 2 2 0 2 0 0 2 0 0 2 0 0 3 2 1 3 1 0 2 1 3 3 2 3 0 2 3
0 2 3 2 0 3 2 2

The encrypted String:

4 3 4 3 2 1 4 4 2 2 4 2 1 2 4 1 3 2 3 2 2 4 1 3 4 1 3 3 1 2 2 3 4 2 1 2
4 4 2 3

Experiment 6.

We did observations by generating cipher texts to show:

- 1) The change due to change of leader used. (The MatLab code for this program is given in P4 in Appendix-B)
- 2) The change due to change of quasigroup used. (The MatLab code for this program is given in P5 in Appendix-B)

Example:

[illegible]

Quasigroup (Q, \bullet) :

•	1	2	3	4
1	4	3	2	1
2	2	1	4	3
3	1	4	3	2
4	3	2	4	1

Leader: 3

[illegible]

```
1:0    2:0    3:48    4:0    var:432    sd:2.078461e+01
```

- Some leaders form cyclic ciphers, i.e., values form the same pattern to repeat after a regular interval.

Example:

[illegible]

Quasigroup(Q, •):

•	1	2	3	4
1	1	2	3	4
2	2	3	4	1
3	3	4	1	2
4	4	1	2	3

Leader: 1

Message:

- Some messages return a uniform pattern, i.e., all values in cipher are same.
- o Some messages return a pattern which consists of leader only.

Example:

Quasigroup(Q, •):

•	1	2	3	4
1	1	3	2	4
2	3	2	1	4
3	2	4	1	3
4	4	1	3	2

Leader: 3

[illegible][illegible]

```
1:0    2:0    3:48    4:0    var:432    sd:2.078461e+01
```

- o Some messages return themselves as cipher.

Example:

Quasigroup(Q, •):

•	1	2	3	4
1	1	3	2	4
2	3	2	1	4
3	2	4	1	3
4	4	1	3	2

[illegible][illegible]

- Some messages return cyclic cipher, i.e., values form the same pattern to repeat after a regular interval.

Quasigroup(Q, •):

•	1	2	3	4
1	1	3	2	4
2	3	2	1	4
3	2	4	1	3
4	4	1	3	2

Message: 2 4 4 1 2 1 4 1 2 4 4 1 2 1 4 1 2 4 4 1 2 1 4 1 2 4 4 1 2 1 4 1 2 4 4 1 2 1 4 1 2 4
4 1 2 1 4 1

Cipher: 3 3 3 2 2 3 3 2 2 1 4 4 4 4 2 3 1 4 2 3 1 1 4 4 4 2 1 1 3 2 1 1 3 3 3 2 2 3 3 2 2 1 4
4 4 4 2 3

sd:1.870829e+00

Chapter 6.

Quasigroup based Stream Cipher Edon 80

Edon 80 is a binary additive stream cipher [4]. It uses a key K which is of length 80 bits and an initial vector IV , which is of length 64 bits. This IV will be padded with 16 bits fixed value, that is 1 1 1 0 0 1 0 0 0 0 1 1 0 1 1.

Using the key and IV generates a keystream which is of a variable length. That keystream generates cipher from the message. The following is schematic representation of Edon80 as a binary additive stream cipher.

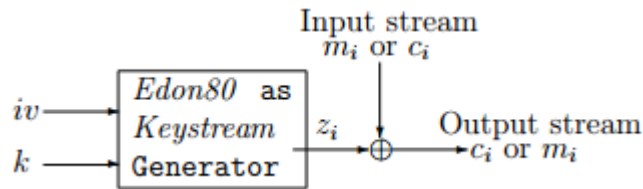


Figure 2: Schematic representation of Edon80 as a binary additive stream cipher.

The following is the schematic and behavioral description of Edon80.

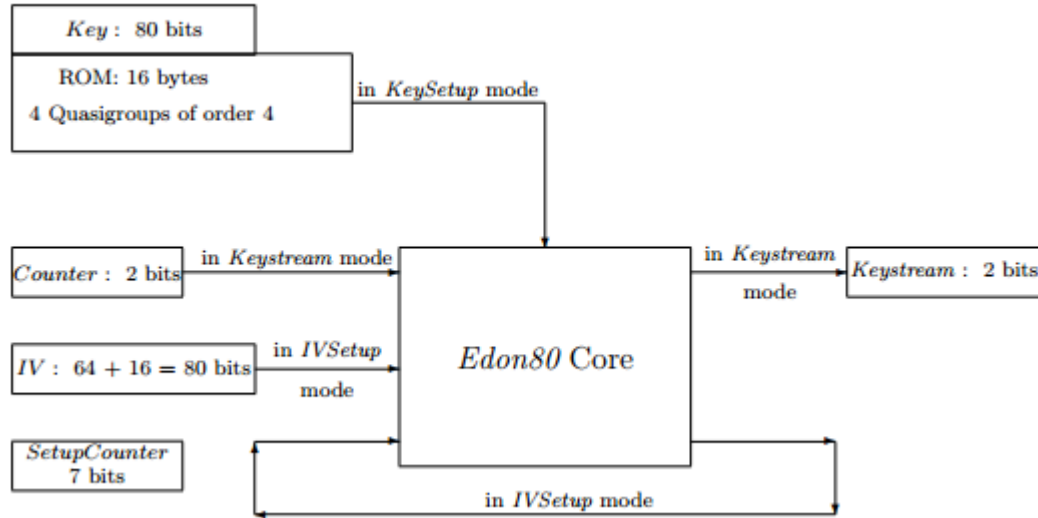


Figure 3: Edon 80 components and their relations

Edon80 works in three possible modes: 1) Key Setup, 2) IV Setup and 3) Keystream mode. For its proper work Edon80 beside the core (that will be described later) has the following additional resources:

1. One register Key of 80 bits to store the actual secret key,
2. One register IV of 80 bits to store padded initialization vector,
3. One internal 2-bit counter Counter as a feeder of Edon80 Core in Keystream mode,
4. One 7 bit SetupCounter that is used in IVSetup mode,
5. One $4 \times 4 = 16$ bytes ROM bank where 4 quasigroups (i.e. Latin squares) of order 4, indexed from $(Q, \bullet 0)$ to $(Q, \bullet 3)$, are stored.

The structure of the Edon80 Core is described in the next two figures. The internal structure of Edon80 can be seen as pipelined architecture of 80 simple 2-bit transformation called e-transformation. The schematic view of a single e-transformer is shown in the following figure.

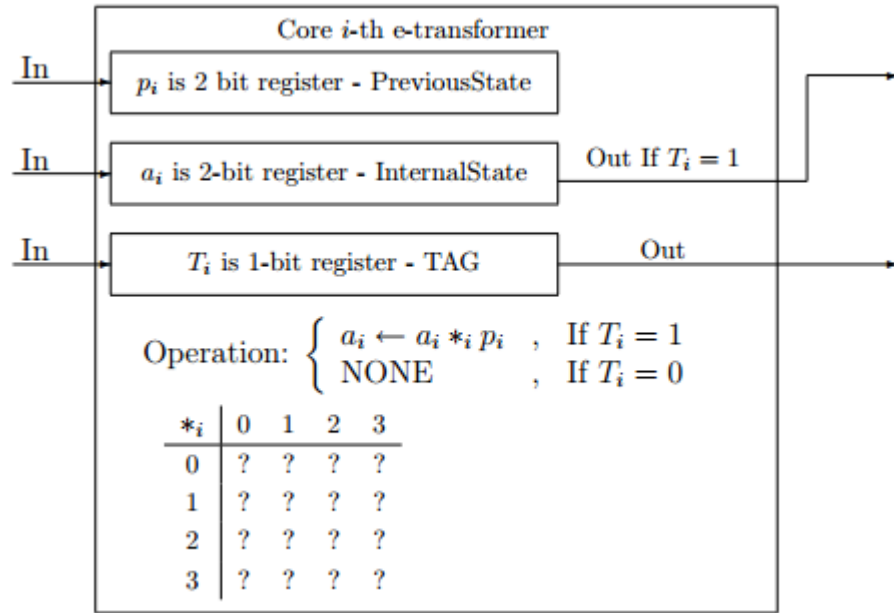


Figure 4: Schematic representation of a single e-transformation of Edon80.

The structure that performs the operation $*_i$ in e-transformers is a quasigroup operation of order 4. We refer an e-transformer by its quasigroup operation $*_i$. The definition of quasigroup string transformations [10] and some of their properties can be found elsewhere. So, in Edon80 we have 80 of this e-transformation (the index i varies from 0 to

79), cascaded in a pipeline, one feeding another. The two 2-bit registers inside every e-transformer (p_i and a_i) are used as two operands by which the new value of a_i is determined according to the defined quasigroup operation $*_i$ for that e-transformer. For different e-transformers there is possibility different quasigroup operations to be defined, out of a set of 4 predefined quasigroups of order 4.

Every e-transformer has one tag-bit T_i which controls whether the e-transformer will compute the next value of a_i or do nothing. All of this 80 e-transformers can work in parallel to calculate their new value of a_i (if the tag permits that) and then pass that new value a_i to the right neighboring register p_{i+1} . If the tag forbids the calculation of a_i , the only value that is transferred to the neighboring element is the value of the tag T_i . The following figure shows the pipelined core of Edon80.

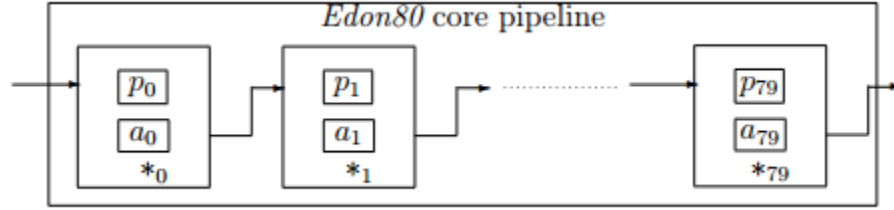


Figure 5: Edon80 core of 80 pipelined e-transformers.

6.1. Implementation of Edon 80

We use 4 quasigroups of order 4 here, which are total 576 in number. Out of those, the developer has declared 64 quasigroups to be good to use in the algorithm. Out of those 64, quasigroups, the developer has used 4 distinct fixed quasigroups in the algorithm. Here we discuss the implementation process of the algorithm in MatLab.

6.1.1. Edon 80 Algorithm

- We have to input IV (length = 64 bits) and Key (length = 80 bits)
- We have to modify IV to 80 bits by adding padding bits (1 1 1 0 0 1 0 0 0 0 0 1 1 0 1 1) to it.
- Store the 4 quasigroups in an array. Those four quasigroups are the following four:

\bullet_1	1	2	3	4
1	1	3	2	4
2	3	2	4	1
3	2	4	1	3
4	4	1	3	2

\bullet_2	1	2	3	4
1	2	4	1	3
2	1	2	3	4
3	3	1	4	2
4	4	3	2	1

\bullet_3	1	2	3	4
1	3	2	1	4
2	2	3	4	1
3	4	1	3	2
4	1	4	2	3

\bullet_4	1	2	3	4
1	4	3	2	1
2	2	1	4	3
3	1	4	3	2
4	3	2	1	4

- Then we generate table 1 by mixing key and IV based on 80 times iteration of e-transformation based on key based quasigroup operation (working quasigroup). The key is denoted by $K = k_1 k_2 k_3 \dots$, and the IV is denoted by $v_1 v_2 v_3 \dots$
- The last row of table 1 is then used to generate table 2 according to the size of input (message or cipher) using a pattern initial input string of quasigroup elements.

- Keystream is obtained from table 2 and the input then xor with this keystream to get cipher/message.

Algorithm:

1. Start.
2. Input k = key (80 bits) and v = initial vector (length = 64 bits) in binary digit form.
3. Add padding values, i.e., 1 1 1 0 0 1 0 0 0 0 0 1 1 0 1 1, at the end of v .
4. Convert the key k to quasigroup form $K = K_1 K_2 K_3 \dots K_{40}$.
5. Convert the initial vector v to quasigroup form $V = V_1 V_2 V_3 \dots V_{40}$.
6. Store $(Q, \bullet_1), (Q, \bullet_2), (Q, \bullet_3), (Q, \bullet_4)$ in array.
7. Calculate working quasigroup operation $\ast_i = \begin{cases} \bullet_{K_i} & 1 \leq i \leq 40 \\ \bullet_{K_{i-40}} & 41 \leq i \leq 80 \end{cases}$
8. Generate table 1 (table 2 of Edon80 keystream mode) using the following algorithm:
 - i. $t_{1,1} = v_{40} \ast_1 K_1$
 - ii. $t_{1,j} = t_{1,j-1} \ast_1 K_j \quad 2 \leq j \leq 40$
 - iii. $t_{1,j} = t_{1,j-1} \ast_1 V_{j-40} \quad 41 \leq j \leq 80$
 - iv. $t_{i,1} = V_{40-i} \ast_i t_{i-1,1} \quad 2 \leq i \leq 40$
 - v. $t_{i,1} = K_{80-i} \ast_i t_{i-1,1} \quad 40 \leq i \leq 80$
 - vi. $t_{i,j} = t_{i,j-1} \ast_i t_{i-1,j} \quad 2 \leq i \leq 80, 2 \leq j \leq 80$
9. Print table 1.
10. a = the last row of table 1.
11. Input bit size of keystream.
12. Generate table 2 (table 3 of Edon80 keystream mode) using the following algorithm (; the row length is the bit size of keystream):
 - i. $a_{1,1} = a_1 \ast_1 1$
 - ii. $a_{0,j} = a_{0,j-1} \ast_1 ((j-1) \bmod 4 + 1) \quad 2 \leq j$
 - iii. $a_{i,1} = a_i \ast_i a_{i-1,1} \quad 2 \leq i \leq 80$
 - iv. $a_{i,j} = a_{i,j-1} \ast_i a_{i-1,j} \quad 2 \leq i \leq 80, 2 \leq j$

13. Print table 2.
14. x = last row of table 2.
15. $finalKey$ = an array of each element in x at even positions.
16. f = convert $finalKey$, the keystream in bits form.
17. Choose type of input message – alphabetical, hexadecimal, quasigroup form, or binary form.
18. Input message.
19. m = Convert the message into bits form correspondingly.
20. Xor each element of m with each element of f at corresponding positions and store in cipher.
21. The array cipher is the cipher text.
22. Convert the ciphertext in required form – hexadecimal, quasigroup form, or unchanged to binary form.
23. Stop.

The MatLab code for the above algorithm of Edon-80 is given in P11 in Appendix-B.

Example: -

[illegible][illegible]

Generated Table 1:

$*_i = qK_i$ $\equiv \bullet_{K_i} \downarrow$	Leader \downarrow	Initial state ($V_1 \dots V_{40} K_1$ $\dots K_{40}$) \rightarrow	3	1	1	-	-	-	2	3	4
q3	4 (V_{40})		2	2	2	-	-	-	3	3	2
q1	3 (V_{39})		4	1	3	-	-	-	1	2	2

q1	2 (V_{38})		1	1	2	-	-	-	1	3	4
-	-	-									
-	-	-									
-	-	-									
q1	1 (K_3)		4	1	4	-	-	-	1	2	1
q1	1 (K_2)		4	4	2	-	-	-	4	1	1
q1	1 (K_1)		3	3	4	-	-	-	4	4	4

a = 3 3 4 3 2 2 2 2 3 4 4 3 2 3 3 3 2 3 4 1 2 1 2 2 1 4 1 2 3 3 2 4 1 2 1 4 3 4 2 4 1 2 1 1 1 4
1 4 1 2 1 3 4 3 2 3 1 2 3 1 3 2 4 2 4 1 2 3 2 2 4 3 4 2 2 1 4 4 4

Keystream size = 100

Generated Table 2:

$*_i = qK_i \equiv$ $\bullet_{K_i} \downarrow$	Leader (a_i) \downarrow	Initial state (1 2 3 4 1 2 3 4 ...) \rightarrow	1	2	3	4	1	2	3	4	-	-	-
q3	3		4	4	2	1	3	1	1	4	-	-	-
q1	3		3	3	4	4	3	2	3	3	-	-	-
q1	4		3	1	4	2	4	1	2	4	-	-	-
-	-	-											
-	-	-											
-	-	-											
q1	4		1	4	1	2	3	1	4	2	-	-	-
q1	4		4	2	3	4	3	2	1	3	-	-	-
q1	4		2	2	4	2	4	1	1	2	-	-	-

Message type – quasigroup form.

Message – (string of quasigroup elements) = 4

8. Generate table 1 (table 2 of Edon80 keystream mode) using the following algorithm:
 - i. $t_{1,1} = v_{40} * K_1$
 - ii. $t_{1,j} = t_{1,j-1} * K_j \quad 2 \leq j \leq 40$
 - iii. $t_{1,j} = t_{1,j-1} * V_{j-40} \quad 41 \leq j \leq 80$
 - iv. $t_{i,1} = V_{40-i} * t_{i-1,1} \quad 2 \leq i \leq 40$
 - v. $t_{i,1} = K_{80-i} * t_{i-1,1} \quad 40 \leq i \leq 80$
 - vi. $t_{i,j} = t_{i,j-1} * t_{i-1,j} \quad 2 \leq i \leq 80, 2 \leq j \leq 80$
9. Print table 1.
10. a = the last row of table1.
11. Input bit size of keystream (= total number of pixels in the image $\times 3 \times 8$).
3 is because there will be three values for each pixel – red, green and blue; 8 because each value is of 8 bits.
12. Generate table 2 (table 3 of Edon80 keystream mode) using the following algorithm (; the row length is the keystream bit size):
 - i. $a_{1,1} = a_1 * 1$
 - ii. $a_{0,j} = a_{0,j-1} * ((j-1) \bmod 4 + 1) \quad 2 \leq j$
 - iii. $a_{i,1} = a_i * a_{i-1,1} \quad 2 \leq i \leq 80$
 - iv. $a_{i,j} = a_{i,j-1} * a_{i-1,j} \quad 2 \leq i \leq 80, 2 \leq j$
13. Print table 2.
14. x = last row of table 2.
15. finalKey = an array of each element in x at even positions.
16. f = convert finalKey, the keastream in bits form.
17. Choose type of input message – alphabetical, hexadecimal, quasigroup form, or binary form.
18. Input image. Store in an array arr. It will be a three dimensional array – the three layers of two-dimensional matrices containing color information over one other. These layers contain information of red, blue and green respectively.
19. Xor each bit of the image with the keystream. Follow the following algorithm.
20. n=1
21. i = 0

22. $j = 0$
23. $k = 0$
24. $\text{byte} = \text{arr}(i, j, k)$
25. $\text{bits}[] = \text{bit form of byte}$
26. $h = 0$
27. $\text{converted}(n) = \text{bits}(h) \oplus f(n)$
28. $n = n + 1$
29. if h is less than or equal to 8, go to step 27
30. $\text{cipher}(i, j, k) = 8 \text{ bit integer form of data in the array converted}[]$
31. $k = k + 1$
32. if k is less than or equal to 3 (height of image), go to step 24
33. $j = j + 1$
34. if j is less than or equal to c , go to step 23
35. $i = i + 1$
36. if i is less than or equal to r , go to step 22
37. The array cipher is the cipher text.
38. Render the data inside cipher as an image with the input name by the user.
39. Stop.

The MatLab code for this algorithm is given in P13 in Appendix – B.

6.2. Application of Edon 80 stream cipher on strings and images

We have implemented Edon80 on all types of strings namely alphabet, hexadecimal, quasigroup elements and binary and generated ciphers. The samples of outputs of seven types of strings are as follows.

IV used in all seven strings (binary) - 00000000000000000000000000000000
00

Plaintext 5 - 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 (Quasigroup form)

Ciphertext 5 –

- 040D386C38 (Hexadecimal)
- 1 1 2 1 1 1 4 2 1 4 3 1 2 3 4 1 1 4 3 1 (Quasigroup form)
- 0 0 0 0 0 1 0 0 0 0 0 0 1 1 0 1 0 0 1 1 1 0 0 0 0 1 1 0 1 1 0 0 0 0 1 1 1 0 0 0 (binary)

Plaintext 6 - 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 (Quasigroup form)

Ciphertext 6 –



- FBF2C793C7 (Hexadecimal)
- 4 4 3 4 4 4 1 3 4 1 2 4 3 2 1 4 4 1 2 4 (Quasigroup form)
- 1 1 1 1 1 0 1 1 1 1 1 1 0 0 1 0 1 1 0 0 0 1 1 1 1 0 0 1 0 0 1 1 1 1 0 0 0 1 1 1 (binary)

Plaintext 7 - 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 (Quasigroup form)

Ciphertext 7 –

- AEA792C692 (Hexadecimal)
- 3 3 4 3 3 3 2 4 3 2 1 3 4 1 2 3 3 2 1 3 (Quasigroup form)
- 1 0 1 0 1 1 1 0 1 0 1 0 0 1 1 1 1 0 0 1 0 0 1 0 1 1 0 0 0 1 1 0 1 0 0 1 0 0 1 0 (binary)

We also applied Edon 80 on images. The demastration is in the following table.

S. No.	Input Image	Output Image
1.	<div data-bbox="669 1423 789 1549"></div> <p>(The input image does not include the black boundaries. They are marked just to show the size of the image.)</p>	<div data-bbox="1175 1419 1292 1545"></div>

Chapter 7.

Conclusion

Here we have discussed few schemes based on fundamental quasigroup based transformations. The importance of the choice of quasigroups are also exhibited in this work. The one of the E-stream finalist scheme based on quasigroup, viz. Edon-80 has been successfully implemented for data & image. The statistical strength of this scheme has been tested and its modification where the parameterization is introduced by giving the choice of quasigroups in user hand has also been implemented. Software has been tested successfully.

Chapter 8.

Future Work

There are lots of scope to improve all these scheme by introducing different domain transformations and using different types of quasigroup transformations. The schemes are to be improved by selecting the cryptographic good choice of quasigroup of different orders. We will carry our future work in this direction.

Appendices

Appendix -A

Some general Functions used in the programs

F1 – RightInverse: -

```
function final= RightInverse(arr)
    r=size(arr);
    c=r(1,2);
    r=r(1,1);

    for z = 1 : r
        for y = 1 : c
            x=arr(z,y);

            final(x,y)=z;
        end
    end
```

F2 – LeftInverse: -

```
function final= LeftInverse(arr)
    r=size(arr);
    c=r(1,2);
    r=r(1,1);

    for z = 1 : r
        for y = 1 : c
            x=arr(z,y);

            final(x,z)=y;
        end
    end
```

F3 – Figure (to figure out the repeating string in the cipher): -

```
function figure = Figure(st)
    len = length(st);

    for i = 1 : len
        if i == 1
            figure = st(i);
        else
            figure = [figure st(i)];
        end

        n = i;
        countF=1;
        found=1;

        while n < len
            for j = n+1 : n + length(figure)
                if figure(j-n) ~= st(j)
                    found=0;
                    break;
                end
            end
            if ~(j<len)
                break;
            end
        end
        if found == 0
            break;
        else
            countF = countF+1;
        end
        n = j;
    end

    if countF >= len/length(figure)
        break;
    end
end
```

```

end
end

if countF >= len/length(figure)
    figure;
else
    figure = st;
end

```

F4 – Permutations: -

```

% returns permutations of input string
function final = Permutations(arr,n)
%arr = input array / string; n = required length of each permutation
    s=size(arr);
    s=s(1,2);

    index=1;

    for i = 1 : n
        a(i)=1;
    end

    f=a;

    while sum(sum(a == s)) ~= n
        index = index + 1;

        a(n) = a(n) + 1;

        for i = n : -1 : 1
            if a(i) == s+1
                a(i) = 1;
                a(i-1) = a(i-1) + 1;
            end
        end
    end
end

```

```

        f = [f;a];
end

s = size(f);
r = s(1,1);
c = s(1,2);

i2=1;
for i = 1 : r

    t=1;
    for j = 1 : c-1
        for k = j+1 : c
            if f(i,j)==f(i,k)

                t=0;

            break;
        end
    end

    if t == 0
        break;
    end
end

if t == 0
    continue;
end;

for j = 1 : c
    final(i2,j) = arr(f(i,j));
end

    i2=i2+1;

end
end

```

F5 – Combinations: -

% returns all the combinations of input array / string and returns the
% output matrix

```
function final = Combinations(arr,n)
```

```
%arr=input array/string; n=required length of each output combination
```

```
    s=size(arr);
```

```
    s=s(1,2);
```

```
    index=1;
```

```
    for i = 1 : n
```

```
        a(i)=1;
```

```
    end
```

```
    f=a;
```

```
    while sum(sum(a == s)) ~= n
```

```
        index = index + 1;
```

```
        a(n) = a(n) + 1;
```

```
    for i = n : -1 : 1
```

```
    if a(i) == s+1
```

```
        a(i) = 1;
```

```
        a(i-1) = a(i-1) + 1;
```

```
    end
```

```
end
```

```
    f = [f;a];
```

```
end
```

```
    s = size(f);
```

```
    r = s(1,1);
```

```

        c = s(1,2);

for i = 1 : r
for j = 1 : c
        final(i,j) = arr(f(i,j));
end
end
end

```

F6 – GetValueFromBits: -

```

% returns integer value of input bit string n
function x=GetValueFromBits(n)
    s=size(n);
    x=0;
    s=s(1,2);
for i = 1 : s
        x=x+((n(i))*(2.^(s-i)));
end
end

```

F7 – GetBitsFromValue: -

```

% returns bit string of input integer value x
function v=GetBitsFromValue(x)
    index=1;
if x==0
        r=0;
end
while x>0
        a=mod(x,2);
        x=floor(x/2);
        r(index)=a;
        index=index+1;
end
    s=size(r);
    s=s(1,2);
if s==1

```

```

        r=[r,0];
        s=size(r);
        s=s(1,2);
end
for i = 1 : s
    v(i)=r(s-i+1);
end
end
end

```

F8 – Alphabet2Binary (to convert an alphabet and return the binary of its ascii): -

```

function f=Alphabet2Binary(m)
    m=int64(m);
    f=0;
    for i = 1 : length(m)
        x=dec2bin(m(i));
        x=int64(x);
        x=x-48;
        y=0;
    for j = 2 : 8-length(x)
        y = [y,0];
    end
    for j = 1 : length(x)
        y = [y,x(j)];
    end
    if f==0
        f=y;
    else
        f=[f,y];
    end
end
end
end

```

F9 – Hexadecimal2Binary (hexadecimal to binary): -

```

function f=Hexadecimal2Binary(m)
    %m=int64(m);
    hx=['0','1','2','3','4','5','6','7','8','9','A','B','C','D','E','F'];

```

```

hxd=[0 1 2 3 4 5 6 7 8 9];
bin=[
    [0 0 0 0];
    [0 0 0 1];
    [0 0 1 0];
    [0 0 1 1];
    [0 1 0 0];
    [0 1 0 1];
    [0 1 1 0];
    [0 1 1 1];
    [1 0 0 0];
    [1 0 0 1];
    [1 0 1 0];
    [1 0 1 1];
    [1 1 0 0];
    [1 1 0 1];
    [1 1 1 0];
    [1 1 1 1];
];
f(1)=0;
for i = 1 : length(m)
    m(i);
    x=find(hx==m(i));
    f(i*4-3)=bin(x,1);
    f(i*4-2)=bin(x,2);
    f(i*4-1)=bin(x,3);
    f(i*4)=bin(x,4);
end
end

```

F10 – QuasiForm2Binary (Quasigroup to binary): -

```

function f=QuasiForm2Binary(m)
%m=int64(m);
hx=[1 2 3 4];
bin=[
    [0 0];
    [0 1];

```



```

        [1 0];
        [1 1];
    ];
    f=0;
for i = 1 : length(m)
    x=find(hx==m(i));
    f(i*2-1)=bin(x,1);
    f(i*2)=bin(x,2);
end
end

```

F11 – Binary2Hexadecimal(Binary to hexadecimal): -

```

function f=Binary2Hexadecimal(x)
    hx='0123456789ABCDEF';
    f=' ';
for i = 1 : length(x)/4
    y=0;
for j = i*4-3 : i*4
    if y==0
        y=x(j);
    else
        y=[y,x(j)];
    end
end
    y=GetValueFromBits(y); % Refer F6 from Appendix-A for this
function
if f == ' '
    f = hx(y+1);
else
    f=strcat(f,hx(y+1));
end
end
end

```

F12 – Binary2QuasiForm (Binary to Quasigroup): -

```

function f=Binary2QuasiForm(x)

```

```

        f=0;
    for i = 1 : length(x)/2
        y=0;
        for j = i*2-1 : i*2
            if y == 0
                y = x(j);
            else
                y = [y,x(j)];
            end
        end
        f(i)=GetValueFromBits(y)+1; % Refer F6 from Appendix-A for this
    function
end
end

```

Appendix-B

List of programs

P1 -Cryptographic scheme based on left quasigroup transformation where leader is fixed: -

```
% works on string transformation based on the following algorithm
%  $c_1 = 1 * m_1$ 
%  $c_i = c_{i-1} * m_i$ 

function e=EncryptMain(x)
    prompt='Enter the name of file which contains Plain Text.\nEnter<0>
for default name (Plain Text.txt).\n';

    o=input(prompt);
    if o == 0
        stringSource='Plain Text.txt';
    else
        stringSource = o;
    end

    x=importdata(stringSource);

    prompt='Press <1> for right encipher\nPress <2> for left
encipher.\n';

    op=input(prompt);
    if op == 1
        e=RightEncipherRepeatRandom(x);
    else
        e=LeftEncipherRepeatRandom(x);
    end
end
```

```

function z = LeftEncipher(x,arr,a)

    li=RightInverse(arr); % Refer F1 from Appendix-A for this function

    for i = 1 : length(x)
        if i == 1
            y(i) = arr(a,x(i));
        else
            y(i) = arr(x(i),y(i-1));
        end
    end

    z=y;
end

function z = RightEncipher(x,arr,a)

    ri=RightInverse(arr); % Refer F1 from Appendix-A for this function

    for i = 1 : length(x)
        if i == 1
            y(i) = arr(x(i),a);
        else
            y(i) = arr(x(i),y(i-1));
        end
    end

    z=y;
end

```

P2 - Iterated cryptographic scheme by left transformation where leader is fixed

```
% works on string transformation based on the following algorithm
%  $c_l = l * m_l$ 
%  $c_i = c_{i-1} * m_i$ 

function e=EncryptMain(x)
    prompt='Enter the name of file which contains Plain Text.\nEnter<0>
for default name (Plain Text.txt).\n';

    o=input(prompt);
    if o == 0
        stringSource='Plain Text.txt';
    else
        stringSource = o;
    end

    x=importdata(stringSource);

    prompt='Press <1> for right encipher\nPress <2> for left
encipher.\n';

    op=input(prompt);
    if op == 1
        e=RightEncipherRepeat(x);
    else
        e=LeftEncipherRepeat(x);
    end
end

function z = LeftEncipherRepeat(x)
    a=2;
    % + operation

    matxixSource='QuasigroupMatrix';
    arr=importdata(matxixSource);
```

```

        li=LeftInverse(arr); % Refer F2 from Appendix-A for this function

        y=x;
    for h = 1 : 4
    for i = 1 : length(x)
    if i == 1
            y(i) = arr(y(i),a);
    else
            y(i) = arr(y(i),y(i-1));
    end
    end
        disp(y);
    end

        z=y;
    for h = 1 : 4
    for i = length(x) : -1 : 1
    if i == 1
            z(i) = li(a,z(i));
    else
            z(i) = li(z(i-1),z(i));
    end
    end
    end
    end

function z = RightEncipherRepeat(x)
    a=2;
    % + operation

    matxixSource='QuasigroupMatrix';
    arr=importdata(matxixSource);

    ri=RightInverse(arr); % Refer F1 from Appendix-A for this function

```

```

        y=x;
    for h = 1 : 10
        for i = 1 : length(x)
            if i == 1
                y(i) = arr(a,y(i));
            else
                y(i) = arr(y(i-1),y(i));
            end
        end
        disp(y);
    end

        z=y;
    for h = 1 : 10
        for i = length(x) : -1 : 1
            if i == 1
                z(i) = ri(a,z(i));
            else
                z(i) = ri(z(i-1),z(i));
            end
        end
    end
end
end
end

```

P3 - Iterated cryptographic scheme by composite left transformation where leader is variable and is based on iteration

```
% works on string transformation based on the following algorithm
%  $c_l = l * m_l$ 
%  $c_i = c_{i-1} * m_i$ 

function e=EncryptMain(x)
    prompt='Enter the name of file which contains Plain Text.\nEnter<0>
for default name (Plain Text.txt).\n';

    o=input(prompt);
    if o == 0
        stringSource='Plain Text.txt';
    else
        stringSource = o;
    end

    x=importdata(stringSource);

    prompt='Press <1> for right encipher\nPress <2> for left
encipher.\n';

    op=input(prompt);
    if op == 1
        e=RightEncipherRepeatRandom(x);
    else
        e=LeftEncipherRepeatRandom(x);
    end
end

function z = LeftEncipherRepeatRandom(x)
% + operation
```



```

    prompt='Enter the name of file which contains Quasigroup.\nEnter<0>
for default name (Quasigroup Matrix.txt).\n';
    o=input(prompt);
if o == 0
    matrixSource='Quasigroup Matrix.txt';
else
    matrixSource = o;
end

arr=importdata(matrixSource);

li=LeftInverse(arr); % Refer F2 from Appendix-A for this function

    prompt='Enter the name of file for output.\nEnter<0> for default name
(Output Left.txt).\n';
    o=input(prompt);
if o == 0
    output='Output Left.txt';
else
    output = o;
end

fileID=fopen(output,'w');

y=x;
fprintf(fileID,'Plain Text\n');
fprintf(fileID,'%d',y);
fprintf(fileID,'\n\n');
for h = 1 : 4
    a=mod( ( h * h ) , 4 ) + 1;
for i = 1 : length(x)
if i == 1
        y(i) = arr(a,y(i));
else
        y(i) = arr(y(i-1),y(i));
end
end

```

```

end

    fprintf(fileID, '%d', y);
    fprintf(fileID, '\n');
end

    z=y;
for h = 4 : -1 : 1
    a=mod( ( h * h ) , 4 ) + 1;
for i = length(x) : -1 : 1
if i == 1

        z(i) = li(a,z(i));

else

        z(i) = li(z(i-1),z(i));

end
end
end

    fprintf(fileID, '\n\nRetrieved Plain Text\n');
    fprintf(fileID, '%d', z);

    fclose('all');

    z=0;
end

function z = RightEncipherRepeatRandom(x)
% + operation

    prompt='Enter the name of file which contains Quasigroup.\nEnter<0>
for default name (Quasigroup Matrix.txt).\n';
    o=input(prompt);
if o == 0
        matrixSource='Quasigroup Matrix.txt';
else
        matrixSource = o;
end
end

```

```

arr=importdata(matrixSource);

ri=RightInverse(arr); % Refer F1 from Appendix-A for this function

prompt='Enter the name of file for output.\nEnter<0> for default name
(Output Right.txt).\n';
o=input(prompt);
if o == 0
    output='Output Right.txt';
else
    output = o;
end

fileID=fopen(output,'w');

y=x;
fprintf(fileID,'Plain Text\n');
fprintf(fileID,'%d',y);
fprintf(fileID,'\n\n');
for h = 1 : 10
    a=mod(mod( ( h * h ) , 10 ) , 4 )+ 1;
for i = 1 : length(x)
    if i == 1
        y(i) = arr(y(i),a);
    else
        y(i) = arr(y(i),y(i-1));
    end
end
    fprintf(fileID,'%d',y);
    fprintf(fileID,'\n');
end

z=y;
for h = 10 : -1 : 1
    a=mod(mod( ( h * h ) , 10 ) , 4 )+ 1;

```

```

for i = length(x) : -1 : 1
if i == 1
        z(i) = ri(z(i),a);
else
        z(i) = ri(z(i),z(i-1));
end
end
end

fprintf(fileID, '\n\nRetrieved Plain Text\n');
fprintf(fileID, '%d', z);

fclose('all');
z=0;
end

```

P4 – Automation algorithm to test P1 on different leaders ($\in Q$) from an in built array: -

```
% uses different leaders to encipher to test
function f = MainForLeader()

    prompt='Press <1> for Right transformation\nPress <2> for left
transformation\n';
    op=input(prompt);

    l=[1 2 3 4];

    prompt='Enter the name of file which contains Quasigroup.\nEnter<0>
for default name (Quasigroup Matrix.txt).\n';
    o=input(prompt);
    if o == 0
        matrixSource='Quasigroup Matrix.txt';
    else
        matrixSource = o;
    end

    q = importdata(matrixSource);

    prompt='Enter the name of file which contains Plain Text.\nEnter<0>
for default name (Plain Text.txt).\n';
    o=input(prompt);
    if o == 0
        fileName='Plain Text.txt';
    else
        fileName = o;
    end

    pt=importdata(fileName);

    pts=size(pt);
    ptr=pts(1,1);
```

```

ptc=pts(1,2);

qs=size(q);
qr=qs(1,1);
qc=qs(1,2);

ls=size(l);
lc=ls(1,2);

mkdir('leader');
for pta = 1 : ptr

    ptw=pt(pta,:);
    file=int2str(pta);
    file=strcat('leader\ ',file);
    fileID=fopen(file,'w');

    fprintf(fileID,'Message');
    fprintf(fileID,'%d',pta);
    fprintf(fileID,':\t');

    fprintf(fileID,'%d',ptw);
    fprintf(fileID,'\n\n\n');

for qa = 1 : qr/4

    qw=q(qa*lc-lc+1:qa*lc,:);

    fprintf(fileID,'Quasigroup');
    fprintf(fileID,'%d',qa);
    fprintf(fileID,':\n');

for k = 1 : lc

    fprintf(fileID,'%d',qw(k,:));
    fprintf(fileID,'\n');

```

```

end

for lb = 1 : lc

    fprintf(fileID, '\t');

    fprintf(fileID, 'Leader');
    fprintf(fileID, '%d', lb);
    fprintf(fileID, ':\t');

    lstr=int2str(l(lb));
    fprintf(fileID, lstr);
    fprintf(fileID, '\n');

    c_=ptw;

for k = 1 : 4

    if op == 1

        c_=RightEncipher(c_,qw,l(lb));

    else

        c_=LeftEncipher(c_,qw,l(lb));

    end

    fprintf(fileID, '\t\t');

    fprintf(fileID, 'c');
    fprintf(fileID, '%d', k);
    fprintf(fileID, ':\t');

    fprintf(fileID, '%d', c_);
    fprintf(fileID, '\n');

    mean=ptc/lc;
    var=0;

```

```

fprintf(fileID, '\t\t\t');

fprintf(fileID, 'Frequency:\t');

for k = 1 : lc

    Sum=sum(sum(c_==l(k)));

    fprintf(fileID, '\t');
    fprintf(fileID, '%d', l(k));
    fprintf(fileID, ':');
    fprintf(fileID, '%d', Sum);

    var=var+(mean-Sum)^2;

end

var=var/lc;
sd=var^0.5;

fprintf(fileID, '\t');
fprintf(fileID, 'var:');
fprintf(fileID, '%d', var);

fprintf(fileID, '\t');
fprintf(fileID, 'sd:');
fprintf(fileID, '%d', sd);
fprintf(fileID, '\n');

rs=Figure(c_); % Refer F3 from Appendix-A for this
function
if(rs(length(rs))~=lb)
    rs=0;
    rsl=length(rs);
    fprintf(fileID, '\t\t\tRepeating string:');
    fprintf(fileID, '%d', rs);
    fprintf(fileID, '\tPeriod:');
    fprintf(fileID, '%d', rsl);

```



```
                                fprintf(fileID, '\n');  
end  
end  
end  
                                fclose('all');  
end  
end
```

P5 – Automation algorithm to test P1 on different quasigroups (given in file) using all leaders ($\in Q$) sequentially: -

```
% uses different quasigroups to encipher to test
function f = MainForQuasigroup()

    prompt='Press <1> for Right transformation\nPress <2> for left
transformation\n';
    op=input(prompt);

    l=[1 2 3 4];

    prompt='Enter the name of file which contains Quasigroup.\nEnter<0>
for default name (Quasigroup Matrix.txt).\n';
    o=input(prompt);
    if o == 0
        matrixSource='Quasigroup Matrix.txt';
    else
        matrixSource = o;
    end

    q = importdata(matrixSource);

    prompt='Enter the name of file which contains Plain Text.\nEnter<0>
for default name (Plain Text.txt).\n';
    o=input(prompt);
    if o == 0
        fileName='Plain Text.txt';
    else
        fileName = o;
    end
    pt=importdata(fileName);

    pts=size(pt);
    ptr=pts(1,1);
```

```

ptc=pts(1,2);

qs=size(q);
qr=qs(1,1);
qc=qs(1,2);

ls=size(l);
lc=ls(1,2);

mkdir('quasigroup');
for pta = 1 : ptr

    ptw=pt(pta,:);

    file=int2str(pta);
    file=strcat('quasigroup\ ',file);
    fileID=fopen(file,'w');

    fprintf(fileID,'Message');
    fprintf(fileID,'%d',pta);
    fprintf(fileID,':\t');

    fprintf(fileID,'%d',ptw);
    fprintf(fileID,'\n\n\n');

for lb = 1 : lc

    fprintf(fileID,'Leader');
    fprintf(fileID,'%d',lb);
    fprintf(fileID,':\t');

    lstr=int2str(l(lb));
    fprintf(fileID,lstr);
    fprintf(fileID,'\n');

```

```

for qa = 1 : qr/lc

    qw=q(qa*lc-lc+1:qa*lc,:);

    fprintf(fileID, '\tQuasigroup');
    fprintf(fileID, '%d', qa);
    fprintf(fileID, ':\n');

for k = 1 : lc

    fprintf(fileID, '\t');
    fprintf(fileID, '%d', qw(k,:));
    fprintf(fileID, '\n');

end

c_=ptw;

for k = 1 : 4
if op == 1

    c_=RightEncipher(c_,qw,l(lb));

else

    c_=LeftEncipher(c_,qw,l(lb));

end

    fprintf(fileID, '\t\t');

    fprintf(fileID, 'c');
    fprintf(fileID, '%d', k);
    fprintf(fileID, ':\t');

    fprintf(fileID, '%d', c_);
    fprintf(fileID, '\n');

    mean=ptc/lc;
    var=0;

    fprintf(fileID, '\t\t\t');

```

```

fprintf(fileID, 'Frequency:\t');

for n = 1 : lc

    Sum=sum(sum(c_==1(n)));

    fprintf(fileID, '\t');
    fprintf(fileID, '%d', l(n));
    fprintf(fileID, ':');
    fprintf(fileID, '%d', Sum);

    var=var+(mean-Sum)^2;

end

var=var/lc;
sd=var^0.5;

fprintf(fileID, '\t');
fprintf(fileID, 'var:');
fprintf(fileID, '%d', var);

fprintf(fileID, '\t');
fprintf(fileID, 'sd:');
fprintf(fileID, '%d', sd);
fprintf(fileID, '\n');

rs=Figure(c_); % Refer F3 from Appendix-A for this
function
if(rs(length(rs))~=lb)
    rs=0;
    rsl=length(rs);
    fprintf(fileID, '\t\t\tRepeating string:');
    fprintf(fileID, '%d', rs);
    fprintf(fileID, '\tPeriod:');
    fprintf(fileID, '%d', rsl);
    fprintf(fileID, '\n');
end

```

```
end
end
end
    fclose('all');
end
```

P6 – Automation algorithm to test P1 on different messages (given in file) using all leaders ($\in Q$) sequentially: -

```
% uses different plaintexts to encipher to test
function f = MainForMessage()

    prompt='Press <1> for Right transformation\nPress <2> for left
transformation\n';
    op=input(prompt);

    l=[1 2 3 4];

    prompt='Enter the name of file which contains Quasigroup.\nEnter<0>
for default name (Quasigroup Matrix.txt).\n';
    o=input(prompt);
    if o == 0
        matrixSource='Quasigroup Matrix.txt';
    else
        matrixSource = o;
    end

    q = importdata(matrixSource);

    prompt='Enter the name of file which contains Plain Text.\nEnter<0>
for default name (Plain Text.txt).\n';

    o=input(prompt);
    if o == 0
        fileName='Plain Text.txt';
    else
        fileName = o;
    end

    pt=importdata(fileName);

    pts=size(pt);
```

```

ptr=pts(1,1);
ptc=pts(1,2);

qs=size(q);
qr=qs(1,1);
qc=qs(1,2);

ls=size(l);
lc=ls(1,2);

mkdir('message');

for qa = 1 : qr/lc

    file=int2str(qa);
    file=strcat('message\',file);
    fileID=fopen(file,'w');

    qw=q(qa*lc-lc+1:qa*lc,:);

    fprintf(fileID,'Quasigroup');
    fprintf(fileID,'%d',qa);
    fprintf(fileID,':\n');

for k = 1 : lc
    fprintf(fileID,'%d',qw(k,:));
    fprintf(fileID,'\n');
end
    fprintf(fileID,'\n\n\n');

for lb = 1 : lc

    fprintf(fileID,'Leader');
    fprintf(fileID,'%d',lb);
    fprintf(fileID,':\t');

```



```

lstr=int2str(l(lb));
fprintf(fileID,lstr);
fprintf(fileID,'\n');

for pta = 1 : ptr

    ptw=pt(pta,:);

    fprintf(fileID,'\t');

    fprintf(fileID,'Message');
    fprintf(fileID,'%d',pta);
    fprintf(fileID,':\t');

    fprintf(fileID,'%d',ptw);
    fprintf(fileID,'\n');

    c_=ptw;
for k = 1 : 4
if op == 1
        c_=RightEncipher(c_,qw,l(lb));
else
        c_=LeftEncipher(c_,qw,l(lb));
end

    fprintf(fileID,'\t\t');

    fprintf(fileID,'c');
    fprintf(fileID,'%d',k);
    fprintf(fileID,':\t');

    fprintf(fileID,'%d',c_);
    fprintf(fileID,'\n');

```

```

mean=ptc/lc;
var=0;

fprintf(fileID, '\t\t\t');

fprintf(fileID, 'Frequency:\t');

for n = 1 : lc

    Sum=sum(sum(c_==l(n)));

    fprintf(fileID, '\t');
    fprintf(fileID, '%d', l(n));
    fprintf(fileID, ':');
    fprintf(fileID, '%d', Sum);

    var=var+(mean-Sum)^2;

end

var=var/lc;
sd=var^0.5;

fprintf(fileID, '\t');
fprintf(fileID, 'var:');
fprintf(fileID, '%d', var);

fprintf(fileID, '\t');
fprintf(fileID, 'sd:');
fprintf(fileID, '%d', sd);
fprintf(fileID, '\n');

rs=Figure(c_); % Refer F3 from Appendix-A for this
function
if(rs(length(rs))~=lb)
    rs=0;
    rsl=length(rs);
    fprintf(fileID, '\t\t\tRepeating string:');

```

```
        fprintf(fileID, '%d', rs);  
        fprintf(fileID, '\tPeriod:');  
        fprintf(fileID, '%d', rsl);  
        fprintf(fileID, '\n');  
    end  
end  
end  
    fclose('all');  
end  
end
```

P7 – Cryptographic scheme by left transformation where leader is based on key

```
% works on string transformation based on the following algorithm
%  $c_0 = k_0 * m_0$ 
%  $c_i = c_{i-1} * (k_i * m_i)$ 

function x=Bits80Values40(m,k)
    prompt='Enter the name of file which contains Plain Text.\nEnter<0>
for default name (Plain Text.txt).\n';

    o=input(prompt);
    if o == 0
        fileName='Plain Text.txt';
    else
        fileName = o;
    end

    prompt='Enter the name of file which contains Quasigroup.\nEnter<0>
for default name (Quasigroup Matrix.txt).\n';
    o=input(prompt);
    if o == 0
        matrixSource='Quasigroup Matrix.txt';
    else
        matrixSource = o;
    end

    prompt='Enter the name of file for output.\nEnter<0> for default name
(Output.txt).\n';
    o=input(prompt);
    if o == 0
        output='Output.txt';
    else
        output = o;
    end
end
```

```

    prompt='Press <1> for right encipher\nPress <2> for left
encipher.\n';

    op=input(prompt);

    fileID=fopen(output,'w');

    m=importdata(fileName);
    k=round(rand(1,80));

    sm=size(m);
    sm=sm(1,2);

    for i = 1 : sm/2
        if i == 1
            mv = GetValueFromBits([m(i*2-1) m(i*2)]); % Refer F6 from
Appendix-A for this function
        else
            mv = [mv,GetValueFromBits([m(i*2-1) m(i*2)])]; % Refer F6
from Appendix-A for this function, GetValueFromBits()
        end
    end

    for i = 1 : sm/2
        if i == 1
            kv = GetValueFromBits([k(i*2-1) k(i*2)]); % Refer F6 from
Appendix-A for this function
        else
            kv = [kv,GetValueFromBits([k(i*2-1) k(i*2)])]; % Refer F6
from Appendix-A for this function, GetValueFromBits
        end
    end

    fprintf(fileID,'The message value string:\n');
    fprintf(fileID,'%d',mv);
    fprintf(fileID,'\n');

```

```

fprintf(fileID, 'The message bit string:\n');
fprintf(fileID, '%d', m);
fprintf(fileID, '\n');

fprintf(fileID, 'The key value string:\n');
fprintf(fileID, '%d', kv);
fprintf(fileID, '\n');
fprintf(fileID, 'The key bit string:\n');
fprintf(fileID, '%d', k);
fprintf(fileID, '\n');

arr=importdata(matrixSource);

ri=RightInverse(arr); % Refer F1 from Appendix-A for this function
li=LeftInverse(arr); % Refer F2 from Appendix-A for this function

for i = 1 : (sm/2)
    vm=GetValueFromBits([m(i*2-1) m(i*2)]) + 1; % Refer F6 from
Appendix-A for this function
    vk=GetValueFromBits([k(i*2-1) k(i*2)]) + 1; % Refer F6 from
Appendix-A for this function
    if i == 1
        if op == 1
            y(i)=arr(vk,vm);
        else
            y(i)=arr(vm,vk);
        end
    else
        if op == 1
            val=arr(vk,vm);
        else
            val=arr(vm,vk);
        end
        y(i)=arr(y(i-1),val);
    end
end
end

```

```

for i = 1: (sm/2)
if i == 1
e = GetBitsFromValue(y(i)-1); % Refer F7 from Appendix-A for
this function
else
e=[e,GetBitsFromValue(y(i)-1)]; % Refer F7 from Appendix-A
for this function, GetBitsFromValue()
end
end

fprintf(fileID,'The cipher value string\n');
fprintf(fileID,'%d',y-1);
fprintf(fileID,'\n');
fprintf(fileID,'The cipher bit string\n');
fprintf(fileID,'%d',e);
fprintf(fileID,'\n');

for i = sm/2 : -1 : 1
vk=GetValueFromBits([k(i*2-1) k(i*2)])+1; % Refer F6 from
Appendix-A for this function
if i == 1
if op == 1
z(i)=ri(y(i),vk);
else
z(i)=li(y(i),vk);
end
else
if op == 1
val=ri(y(i),y(i-1));
else
val=li(y(i),y(i-1));
end
z(i)=ri(val,vk);
end
end

z=char(z);

s=size(z);

```

```

        s=s(1,2);
    for i = 1 : s
        if i == 1
            x=GetBitsFromValue(z(i)-1); % Refer F7 from Appendix-A for
this function
        else
            x=[x,GetBitsFromValue((z(i)-1))]; % Refer F7 from Appendix-A
for this function, GetBitsFromValue()
        end
    end

    fprintf(fileID,'\n\nRetrieved Plain Text\n');
    fprintf(fileID,'%d',x);

    fclose('all');
    x=0;
end

```


P8 - Lexicographical Arrangement of quasigroups of order 4 (MatLab)

```
% Arranges arrays / strings in lexicographical order - independent
program
function f = OrderLexiV2(arr)
    prompt='Enter the name of file which for input list of
quasigroups.\nEnter<0> for default name (gen4all.txt).\n';
    o=input(prompt);
    if o == 0
        fileName='gen4all.txt';
    else
        fileName = o;
    end
    arr=importdata(fileName);

    start = cputime;

    s = size(arr);

    r = s(1,1);
    limit = s(1,2);

    for i = 1 : r-1
        for j = i+1 : r
            for k = limit : -1 : 1
                if int64(arr(i,k)) > int64(arr(j,k))
                    t = arr(i,:);
                    arr(i,:)=arr(j,:);
                    arr(j,:)=t;
                end
            end
        end
    end

    prompt='Enter the name for output.\nEnter<0> for default name
(regenerated.txt).\n';
```

```

        o=input(prompt);
if o == 0
        fileName='regenerated.txt';
else
        fileName = o;
end
        fileID=fopen(fileName,'w');

for i = 1 : r
for j = 1 : limit
        fprintf(fileID,'%d',arr(i,j));
end
        fprintf(fileID,'\n');
end
        fclose('all');

End = cputime;

disp('Time Taken');
disp(End-start);

n=1;
while n~=0
        prompt='Enter the index of arranged quasigroup to output.\nEnter
<0> to exit: ';
        n=input(prompt);
if n == 0
break;
end
        fprintf('\n');
for i = 1 : length(arr(n,:))
        fprintf('%d',arr(n,i));
if mod(i,4) == 0
        fprintf('\n');
end
end
        fprintf('\n');

```

```
end
```

```
    fclose('all');
```

```
end
```

P9 - Lexicographical Arrangement of quasigroups of order 4 (C++)

```
#include<iostream>
#include<fstream>
using namespace std;

const int r=5;
int c=6;
string data[r];
const int limit=4;

class ArrangeLexicographically{
private:

    ArrangeLexicographically *heads[limit];
    ArrangeLexicographically *tails[limit];
    ArrangeLexicographically *next,*prev;
    string quasi;

    void input(){
        ifstream ifile("gen4all.txt");
        for(int i=0;i<r;i++){
            getline(ifile,data[i]);
        }
    }

    void enqueue(int n,string st){
        ArrangeLexicographically *ob=new ArrangeLexicographically(st);

        if(heads[n]==NULL){
            heads[n]=tails[n]=ob;
            heads[n]->prev=NULL;

```

```

    }
    else{
        tails[n]->next=ob;
        ob->prev=tails[n];
        tails[n]=ob;
    }
    tails[n]->next=NULL;
}

```

```

void dequeueAllToData(){
    int index=0;
    for(int i=0;i<limit;i++){

        ArrangeLexicographically *ob=heads[i];
        while(ob){
            data[index]=ob->quasi;
            index++;
            ob=ob->next;
        }
        heads[i]=tails[i]=NULL;
    }
}

```

```

void sort(){
    for(int j=c-1;j>=0;j--){
        for(int i=0;i<r;i++){
            string st=data[i];
            if(st.length()<j+1)enqueue(limit-1,st);
            else enqueue(st[j]-49,st);
        }
    }
}

```

```

        dequeueAllToData();
    }
}

void print(){
    ofstream ofile("regenerated.txt");
    for(int i=0;i<r;i++){
        string st=data[i];
        for(int j=0;j<c;j++){
            ofile<<st[j]<<" ";
        }
        ofile<<"\n";
    }
    cout<<"Please find the output quasigroup in the file \"regenerated.txt\".";
}

void printTheChoosen(){
    while(true){
        cout<<"\n\nEnter the quasigroup index to output.\n";
        cout<<"Press <0> to exit:\n";
        int n;
        cin>>n;
        if(n==0)break;
        cout<<data[n-1]<<"\n";
    }
}

public:
    ArrangeLexicographically(string st){
        quasi=st;
    }
}

```

```

        ArrangeLexicographically(){
            for(int i=0;i<limit;i++){
                heads[i]=tails[i]=NULL;
            }

            input();
            sort();
            print();
            printTheChoosen();
        }
};

int main(){
    new ArrangeLexicographically();
}

```

P10 - Generalized iterated cryptographic scheme based on composite left quasigroup transformation

```
% works on string transformation based on the following algorithm
% given four quasigroups
%  $c = E \cdot k_1 \cdot k_2 \cdot (m)$ 
function x=Encrypt(y)
    prompt='Enter the name of file which contains Plain Text.\nEnter<0>
for default name (Plain Text.txt).\n';

    o=input(prompt);
    if o == 0
        fileName='Plain Text.txt';
    else
        fileName = o;
    end
    m=importdata(fileName);

    prompt='Enter the name of file which contains Quasigroup.\nEnter<0>
for default name (Quasigroup Matrix.txt).\n';
    o=input(prompt);
    if o == 0
        matrixSource='Quasigroup Matrix.txt';
    else
        matrixSource = o;
    end

    prompt='Enter the name of file for output.\nEnter<0> for default name
(Output.txt).\n';
    o=input(prompt);
    if o == 0
        output='Output.txt';
    else
        output = o;
    end
    fileID=fopen(output,'w');
```



```

prompt='Press <1> for Right Encipher\nPress <2> for Left Encipher\n';
op=input(prompt);

k=round(rand(1,160));

sm=size(m);
sm=sm(1,2);

for i = 1 : sm/2
    mv(i) = GetValueFromBits([m(i*2-1) m(i*2)]); % Refer F6 from
Appendix-A for this function
end

km=size(k);
km=km(1,2);

for i = 1 : km/2
    kv(i) = GetValueFromBits([k(i*2-1) k(i*2)]); % Refer F6 from
Appendix-A for this function
end

fprintf(fileID,'The message bit string:\n');
fprintf(fileID,'%d',m);
fprintf(fileID,'\n');
fprintf(fileID,'The message value string:\n');
fprintf(fileID,'%d',mv);
fprintf(fileID,'\n');

fprintf(fileID,'The key bit string:\n');
fprintf(fileID,'%d',k);
fprintf(fileID,'\n');
fprintf(fileID,'The key value string:\n');
fprintf(fileID,'%d',kv);
fprintf(fileID,'\n');

```

```

a=importdata(matrixSource);

for i = 1 : sm/2
    vm=GetValueFromBits([m(i*2-1) m(i*2)])+1; % Refer F6 from
Appendix-A for this function
    vk1=GetValueFromBits([k(i*4-3) k(i*4-2)])+1; % Refer F6 from
Appendix-A for this function
    vk2=GetValueFromBits([k(i*4-1) k(i*4)])+1; % Refer F6 from
Appendix-A for this function

    arr=a(vk1*4-3:vk1*4,:);

if op == 1
    c(i)=arr(vm,vk2);
else
    c(i)=arr(vk2,vm);
end
end

fprintf(fileID,'The encrypted String:\n');
fprintf(fileID,'%d',c);
fprintf(fileID,'\n');

for i = 1 : sm/2

    vk1=GetValueFromBits([k(i*4-3) k(i*4-2)])+1; % Refer F6 from
Appendix-A for this function
    vk2=GetValueFromBits([k(i*4-1) k(i*4)])+1; % Refer F6 from
Appendix-A for this function

    arr=a(vk1*4-3:vk1*4,:);

    ri=RightInverse(arr); % Refer F1 from Appendix-A for this
function
    li=LeftInverse(arr); % Refer F2 from Appendix-A for this function

```

```

if op == 1
    x(i)=ri(c(i),vk2);
else
    x(i)=li(c(i),vk2);
end
end
x=x-1;

fprintf(fileID,'\n\nRetrieved Plain Text\n');
fprintf(fileID,'%d',x);

fclose('all');
x=0;
end

```

P11 - Edon80

```
%implementation of Edon80 algorithm
function f=Edon80()

    prompt='Enter the name of file to input key.\nEnter <0> to input key
from default file (Key.txt):\n';
    o=input(prompt);
    if o == 0
        keySource='Key.txt';
    else
        keySource=o;
    end
    k=importdata(keySource);

    prompt='Enter the name of file to input the IV.\nEnter <0> to input
key from default file (IV.txt):\n';
    o=input(prompt);
    if o == 0
        ivSource='IV.txt';
    else
        ivSource=o;
    end
    v=importdata(ivSource);

    q=[
        [1 3 2 4 3 2 4 1 2 4 1 3 4 1 3 2];
        [2 4 1 3 1 2 3 4 3 1 4 2 4 3 2 1];
        [3 2 1 4 2 3 4 1 4 1 3 2 1 4 2 3];
        [4 3 2 1 2 1 4 3 1 4 3 2 3 2 1 4];
    ];

    a=GenerateKey(k,v,q,4);% 4 is the order of quasigroup
end
```

```

% generates table 3 of Edon 80 keystream mode- referred as table 2 in
this
% prog
% generates keystream; enc/dec the input

function f = Encipher(a,K,V,limit,Q)

% a=last row of table 1 of prog; K=key; V=IV; limit=order of
quasigroup;Q=4
% quasigroups matrix
    prompt='Enter the length of keystream in number of bits.\n';
    lm=input(prompt);
if mod(lm,2)==1
    lm=lm+1;
end

    s=size(a);
    r=s(1,1);
    A=a(r,:);

    prompt='Enter the name of file to write the second table
formed.\nEnter<0> for default name (Table2.txt).\n';
    o=input(prompt);
if o == 0
    outputTables='Table2.txt';
else
    outputTables = o;
end
    fileID=fopen(outputTables,'w');

    fprintf(fileID,'\tLeader\t');
for i = 1 : lm
    fprintf(fileID,'%d',mod(i-1,4)+1);
    fprintf(fileID,'\t');
end
    fprintf(fileID,'\n');

```

```

for i = 1 : length(a)

    fprintf(fileID, 'q');

    for j = 1 : limit
        for l = 1 : limit
            if i <= 40

                qw(j,l)=Q(j,l,K(i));

            else

                qw(j,l)=Q(j,l,K(i-40));

            end
        end
    end

    if i <= 40

        fprintf(fileID, '%d',K(i));

    else

        fprintf(fileID, '%d',K(i-40));

    end

    fprintf(fileID, '\t');
    fprintf(fileID, '%d',A(i));
    fprintf(fileID, '\t\t');

    for j = 1 : lm

        if i == 1

            y=mod(j-1,4)+1;

        else

            y=z(i-1,j);

        end

        if j == 1

            x=A(i);

        else

```

```

        x=z(i,j-1);
end

        z(i,j)=qw(x,y);

        fprintf(fileID,'%d',z(i,j));
        fprintf(fileID,'\t');
end

        fprintf(fileID,'\n');
end

s=size(z);
r=s(1,1);
Z=z(r,:);

finalKey=0;

for i = 1 : lm
    if mod(i,2) == 0
        x=Z(i)-1;

    if i == 2
        finalKey=x;
    else
        finalKey=[finalKey,x];
    end
end
end

    finalKey=QuasiForm2Binary(finalKey+1); % Refer F10 from Appendix-A
    for this function

    prompt='Enter the name of file to write the keystream.\nEnter <0> for
the default file (Keystream.txt).\n';
    o=input(prompt);
    if o == 0
        keystreamFile='Keystream.txt';

```

```

else
    keystreamFile=o;
end
fileID=fopen(keystreamFile,'w');
fprintf(fileID,'%d',finalKey);

prompt='Enter <1> to encipher using generated Keystream.\nEnter <2>
to decipher using generated Keystream.\nEnter <0> to exit.\n';
o=input(prompt);
if o == 1

    prompt='Enter the name of file where the message / plaintext is
present.\nEnter <0> for default file (Plain Text.txt)\n';
    o=input(prompt);
if o==0
        messageFile='Plain Text.txt';
else
        messageFile=o;
end

    prompt='Enter the form of input plaintext.\nEnter <1> for
alphabplet.\nEnter <2> for Hexadecimal.\nEnter <3> for quasigroup form (1-
4).\nEnter <4> for binary.\n';
    o=input(prompt);

if o == 1
        fileID=fopen(messageFile,'r');
        m=fscanf(fileID,'%s');
        m=Alphabet2Binary(m); % Refer F8 from Appendix-A for this
function
elseif o == 2
        fileID=fopen(messageFile,'r');
        m=fscanf(fileID,'%s');
        m=Hexadecimal2Binary(m); % Refer F9 from Appendix-A for
this function
elseif o == 3
        m=importdata(messageFile);

```



```

                                m=QuasiForm2Binary(m); % Refer F10 from Appendix-A
for this function
else
                                m=importdata(messageFile);

end
end
end

    lm=length(m);

    f(1)=0;
for i = 1 : lm
        x=finalKey(i);
        y=m(i);
        f(i)=xor(x,y);
end

    prompt='Enter the name of file to write the cipher.\nEnter <0>
for default file (Cipher.txt)\n';
    o=input(prompt);
if o==0
    cipherFile='Cipher.txt';
else
    cipherFile=o;
end

    prompt='Enter the form of output cipher.\nEnter <1> for
Hexadecimal.\nEnter <2> for quasigroup form (1-4).\nEnter <3> for
binary.\n';
    o=input(prompt);

if o == 1
        f=Binary2Hexadecimal(f); % Refer F11 from Appendix-A for this
function
        fileID=fopen(cipherFile,'w');
        fprintf(fileID,'%s',f);
elseif o == 2

```

```

        f=Binary2QuasiForm(f); % Refer F12 from Appendix-A for
this function

        fileID=fopen(cipherFile,'w');
        fprintf(fileID,'%d ',f);
else
        fileID=fopen(cipherFile,'w');
        fprintf(fileID,'%d ',f);
end
end

else
if o == 2

        prompt='Enter the name of file where the cipher is
present.\nEnter <0> for default file (Cipher.txt)\n';
        o=input(prompt);
if o==0

        messageFile='Cipher.txt';
else

        messageFile=o;
end

        prompt='Enter the form of input cipher.\nEnter <1> for
alphabplet.\nEnter <2> for Hexadecimal.\nEnter <3> for quasigroup form (1-
4).\nEnter <4> for binary.\n';
        o=input(prompt);

if o == 1

        fileID=fopen(messageFile,'r');
        m=fscanf(fileID,'%s');
        m=Alphabet2Binary(m); % Refer F8 from Appendix-A for this
function
elseif o == 2

        fileID=fopen(messageFile,'r');
        m=fscanf(fileID,'%s');

```

```

        m=Hexadecimal2Binary(m); % Refer F9 from Appendix-A
for this function
elseif o == 3
        m=importdata(messageFile);
        m=QuasiForm2Binary(m); % Refer F10 from Appendix-
A for this function
else
        m=importdata(messageFile);

end
end
end

        lm=length(m);

        f(1)=0;
for i = 1 : lm
        x=finalKey(i);
        y=m(i);
        f(i)=xor(x,y);
end

        prompt='Enter the name of file to write the message / plain
text.\nEnter <0> for default file (Plain Text.txt)\n';
        o=input(prompt);
if o==0
cipherFile='Plain Text.txt';
else
cipherFile=o;
end

        prompt='Enter the form of output message / plain text.\nEnter
<1> for Hexadecimal.\nEnter <2> for quasigroup form (1-4).\nEnter <3> for
binary.\n';
        o=input(prompt);

if o == 1
        f=Binary2Hexadecimal(f); % Refer F11 from Appendix-A for
this function

```

```

        fileID=fopen(cipherFile,'w');
        fprintf(fileID,'%s',f);
elseif o == 2
        f=Binary2QuasiForm(f); % Refer F12 from Appendix-A
for this function

        fileID=fopen(cipherFile,'w');
        fprintf(fileID,'%d ',f);
else
        fileID=fopen(cipherFile,'w');
        fprintf(fileID,'%d ',f);
end
end
end
        f=0;
end
end

% generates table 2 of Edon 80 Keystream mode- referred as table 1 in
this
% prog
function a=GenerateKey(k,v,q,limit)
% k=input key; v=input iv; q=input 4 quasigroups; limit=order of
quasigroups
        tt=0;
for i = 1 : length(k)/2
if i == 1
        K=GetValueFromBits([k(i*2-1),k(i*2)]); % Refer F6 from
Appendix-A for this function
else
        K=[K,GetValueFromBits([k(i*2-1),k(i*2)])]; % Refer F6 from
Appendix-A for this function
end
end
        K=K+1;

```

```

for i = 1 : length(v)/2
if i == 1
V=GetValueFromBits([v(i*2-1),v(i*2)]); % Refer F6 from
Appendix-A for this function
else
V=[V,GetValueFromBits([v(i*2-1),v(i*2)])]; % Refer F6 from
Appendix-A for this function, GetValueFromBits()
end
end

V=[V,3,2,1,0,0,1,2,3];
V=V+1;

s=size(q);
r=s(1,1);
c=s(1,2);
for i = 1 : r
for j = 1 : c
Q(ceil(j/limit),mod(j-1,limit)+1, i)=q(i,j);
end
end

prompt='Enter the name of file to write the first table
formed.\nEnter<0> for default name (Table1.txt).\n';
o=input(prompt);
if o == 0
outputTables='Table1.txt';
else
outputTables = o;
end

fileID=fopen(outputTables,'w');

fprintf(fileID,'\tLeader\t');
for i = 1 : length(K)
fprintf(fileID,'%d',K(i));
fprintf(fileID,'\t');
end
for i = 1 : length(V)

```

```

        fprintf(fileID, '%d', V(i));
        fprintf(fileID, '\t');
    end
    fprintf(fileID, '\n');

    for i = 1 : length(V)

        fprintf(fileID, 'q');
        fprintf(fileID, '%d', K(i));
        fprintf(fileID, '\t');
        fprintf(fileID, '%d', V(41-i));
        fprintf(fileID, '\t\t');

        for j = 1 : limit
            for l = 1 : limit
                qw(j,l)=Q(j,l,K(i));
            end
        end

        for j = 1 : length(K)

            if i == 1
                y=K(j);
            else
                y=a(i-1,j);
            end

            if j == 1
                x=V(41-i);
            else
                x=a(i,j-1);
            end

            a(i,j)=qw(x,y);
            fprintf(fileID, '%d', a(i,j));
            fprintf(fileID, '\t');
        end
    end

```

```

end

for j = 1 : length(V)

if i == 1
    y=V(j);
else
    y=a(i-1,j+40);
end

    x=a(i,(j+40)-1);

    a(i,j+40)=qw(x,y);
    fprintf(fileID,'%d',a(i,j+40));
    fprintf(fileID,'\t');

end

    fprintf(fileID,'\n');
end

for i = 1 : length(K)

    fprintf(fileID,'*');
    fprintf(fileID,'%d',K(i));
    fprintf(fileID,'\t');
    fprintf(fileID,'%d',K(i));
    fprintf(fileID,'\t\t');

    for j = 1 : limit
        for l = 1 : limit
            qw(j,l)=Q(j,l,(K(i)));
        end
    end

    for j = 1 : length(K)

```

```

        y=a(40+i-1,j);

if j == 1
        x=K(41-i);
else
        x=a(40+i,j-1);
end

        a(40+i,j)=qw(x,y);
        fprintf(fileID,'%d',a(40+i,j));
        fprintf(fileID,'\t');
end

for j = 1 : length(V)

        y=a(40+i-1,40+j);

        x=a(40+i,40+j-1);

        a(i+40,j+40)=qw(x,y);
        fprintf(fileID,'%d',a(i+40,j+40));
        fprintf(fileID,'\t');
end

        fprintf(fileID,'\n');
end

cipher=Encipher(a,K,V,limit,Q);

a=cipher+1;
fclose('all');
end

```


P12 – Modified Edon80, with variable quasigroups: -

```
%implementation of Edon80 algorithm - changeable quasigroups through
files
function f=Edon80()

    prompt='Enter the name of file to input key.\nEnter <0> to input key
from default file (Key.txt):\n';
    o=input(prompt);
    if o == 0
        keySource='Key.txt';
    else
        keySource=o;
    end
    k=importdata(keySource);

    prompt='Enter the name of file to input the IV.\nEnter <0> to input
key from default file (IV.txt):\n';
    o=input(prompt);
    if o == 0
        ivSource='IV.txt';
    else
        ivSource=o;
    end
    v=importdata(ivSource);

    prompt='Enter the name of file to input the four quasigroups.\nEnter
<0> to input key from default file (QuasiG..txt):\n';
    o=input(prompt);
    if o == 0
        quasiGSource='QuasiG..txt';
    else
        quasiGSource=o;
    end
    q=importdata(quasiGSource);
```

```
a=GenerateKey(k,v,q,4);% 4 is the order of quasigroup  
% refer in P11 for this function, GenerateKey().  
end
```

P13 – Edon80-image

```
%implementation of Edon80 algorithm - for images
function f=Edon80()

    prompt='Enter the name of file to input key.\nEnter <0> to input key
from default file (Key.txt):\n';
    o=input(prompt);
    if o == 0
        keySource='Key.txt';
    else
        keySource=o;
    end
    k=importdata(keySource);

    prompt='Enter the name of file to input the IV.\nEnter <0> to input
key from default file (IV.txt):\n';
    o=input(prompt);
    if o == 0
        ivSource='IV.txt';
    else
        ivSource=o;
    end
    v=importdata(ivSource);

    q=[
        [1 3 2 4 3 2 4 1 2 4 1 3 4 1 3 2];
        [2 4 1 3 1 2 3 4 3 1 4 2 4 3 2 1];
        [3 2 1 4 2 3 4 1 4 1 3 2 1 4 2 3];
        [4 3 2 1 2 1 4 3 1 4 3 2 3 2 1 4];
    ];

    a=GenerateKey(k,v,q,4);% 4 is the order of quasigroup
end
```

```

% generates table 3 of Edon 80 keystream mode- referred as table 2 in
this
% prog
% generates keystream; enc/dec the input

function f = Encipher(a,K,V,limit,Q)

% a=last row of table 1 of prog; K=key; V=IV; limit=order of
quasigroup;Q=4
% quasigroups matrix
    prompt='Enter the length of keystream in number of bits.\n';
    lm=input(prompt);
if mod(lm,2)==1
    lm=lm+1;
end

    s=size(a);
    r=s(1,1);
    A=a(r,:);

    prompt='Enter the name of file to write the second table
formed.\nEnter<0> for default name (Table2.txt).\n';
    o=input(prompt);
if o == 0
    outputTables='Table2.txt';
else
    outputTables = o;
end
    fileID=fopen(outputTables,'w');

    fprintf(fileID,'\tLeader\t');
for i = 1 : lm
    fprintf(fileID,'%d',mod(i-1,4)+1);
    fprintf(fileID,'\t');
end
    fprintf(fileID,'\n');

```

```

for i = 1 : length(a)

    fprintf(fileID, 'q');

    for j = 1 : limit
        for l = 1 : limit
            if i <= 40

                qw(j,l)=Q(j,l,K(i));

            else

                qw(j,l)=Q(j,l,K(i-40));

            end
        end
    end

    if i <= 40

        fprintf(fileID, '%d',K(i));

    else

        fprintf(fileID, '%d',K(i-40));

    end

    fprintf(fileID, '\t');
    fprintf(fileID, '%d',A(i));
    fprintf(fileID, '\t\t');

    for j = 1 : lm

        if i == 1

            y=mod(j-1,4)+1;

        else

            y=z(i-1,j);

        end

        if j == 1

            x=A(i);

        else

```

```

        x=z(i,j-1);
end

        z(i,j)=qw(x,y);

        fprintf(fileID,'%d',z(i,j));
        fprintf(fileID,'\t');
end

        fprintf(fileID,'\n');
end

s=size(z);
r=s(1,1);
Z=z(r,:);

finalKey=0;

for i = 1 : lm
    if mod(i,2) == 0
        x=Z(i)-1;

    if i == 2
        finalKey=x;
    else
        finalKey=[finalKey,x];
    end
end
end

finalKey=QuasiForm2Binary(finalKey+1); % Refer F10 from Appendix-A
for this function

prompt='Enter the name of file to write the keystream.\nEnter <0> for
the default file (Keystream.txt).\n';
o=input(prompt);
if o == 0
    keystreamFile='Keystream.txt';

```

```

else
    keystreamFile=o;
end
fileID=fopen(keystreamFile,'w');
fprintf(fileID,'%d',finalKey);

prompt='Enter <1> to encipher using generated Keystream.\nEnter <2>
to decipher using generated Keystream.\nEnter <0> to exit.\n';
o=input(prompt);
if o == 1

    prompt='Enter the name of input file.\nEnter <0> for default file
(Input.jpg):\n';
    o=input(prompt);
if o == 0
        imageSource='Input.jpg';
else
        imageSource=o;
end

    inim=imread(imageSource);
    s=size(inim);

    i=1;
for r = 1 : s(1)
for c = 1 : s(2)
for h = 1 : s(3)
        bin=dec2bin(inim(r,c,h));
        b=8-length(bin);
        bits=0;
for n = 1 : b
                bits(n)=0;
end
for n = 1 : length(bin)
                bits(n+b)=int64(bin(n))-48;
end

        g=0;

```

```

for n = 1 : length(bits)
    x=finalKey(i);
    i=i+1;
    y=bits(n);
    g(n)=xor(x,y);
end
    d='';
for n = 1 : length(g)
    d=strcat(d,int2str(g(n)));
end
    outpix=bin2dec(d);
    outim(r,c,h)=uint8(outpix);
end
end
end
    outim=uint8(outim);

    prompt='Enter the name of output file.\nEnter <0> for default
file (Output.jpg):\n';
    o=input(prompt);
if o == 0
    imageSource='Output.jpg';
else
    imageSource=o;
end

    imwrite(outim, imageSource, 'Mode', 'lossless');

else
if o == 2

    prompt='Enter the name of input file.\nEnter <0> for default
file (Input.jpg):\n';
    o=input(prompt);
if o == 0
    imageSource='Input.jpg';
else

```



```

        imageSource=o;

end

        inim=imread(imageSource);
        s=size(inim);

        i=1;
    for r = 1 : s(1)
    for c = 1 : s(2)
    for h = 1 : s(3)

            bin=dec2bin(inim(r,c,h));
            b=8-length(bin);
            bits=0;

    for n = 1 : b

            bits(n)=0;

    end
    for n = 1 : length(bin)

            bits(n+b)=int64(bin(n))-48;

    end

            g=0;
    for n = 1 : length(bits)

            x=finalKey(i);
            i=i+1;
            y=bits(n);
            g(n)=xor(x,y);

    end

            d='';

    for n = 1 : length(g)

            d=strcat(d,int2str(g(n)));

    end

            outpix=bin2dec(d);
            outim(r,c,h)=uint8(outpix);

    end
    end
    end

        outim=uint8(outim);

```

```

        prompt='Enter the name of output file.\nEnter <0> for default
file (Output.jpg):\n';
        o=input(prompt);
if o == 0
        imageSource='Output.jpg';
else
        imageSource=o;
end

        imwrite(outim, imageSource, 'Mode', 'lossless');
end
        f=0;
end
end

% generates table 2 of Edon 80 Keystream mode- referred as table 1 in
this
% prog
function a=GenerateKey(k,v,q,limit)
% k=input key; v=input iv; q=input 4 quasigroups; limit=order of
quasigroups
        tt=0;
for i = 1 : length(k)/2
if i == 1
        K=GetValueFromBits([k(i*2-1),k(i*2)]); % Refer F6 from
Appendix-A for this function
else
        K=[K,GetValueFromBits([k(i*2-1),k(i*2)])]; % Refer F6 from
Appendix-A for this function, GetValueFromBits()
end
end
        K=K+1;

for i = 1 : length(v)/2
if i == 1
        V=GetValueFromBits([v(i*2-1),v(i*2)]); % Refer F6 from
Appendix-A for this function

```

```

else
    V=[V,GetValueFromBits([v(i*2-1),v(i*2)])]; % Refer F6 from
Appendix-A for this function
end
end
V=[V,3,2,1,0,0,1,2,3];
V=V+1;

s=size(q);
r=s(1,1);
c=s(1,2);
for i = 1 : r
for j = 1 : c
    Q(ceil(j/limit),mod(j-1,limit)+1, i)=q(i,j);
end
end

prompt='Enter the name of file to write the first table
formed.\nEnter<0> for default name (Table1.txt).\n';
o=input(prompt);
if o == 0
    outputTables='Table1.txt';
else
    outputTables = o;
end
fileID=fopen(outputTables,'w');

fprintf(fileID,'\tLeader\t');
for i = 1 : length(K)
    fprintf(fileID,'%d',K(i));
    fprintf(fileID,'\t');
end
for i = 1 : length(V)
    fprintf(fileID,'%d',V(i));
    fprintf(fileID,'\t');
end
fprintf(fileID,'\n');

```

```

for i = 1 : length(V)

    fprintf(fileID, 'q');
    fprintf(fileID, '%d', K(i));
    fprintf(fileID, '\t');
    fprintf(fileID, '%d', V(41-i));
    fprintf(fileID, '\t\t');

    for j = 1 : limit
        for l = 1 : limit
            qw(j,l)=Q(j,l,K(i));
        end
    end

    for j = 1 : length(K)

        if i == 1
            y=K(j);
        else
            y=a(i-1,j);
        end

        if j == 1
            x=V(41-i);
        else
            x=a(i,j-1);
        end

        a(i,j)=qw(x,y);
        fprintf(fileID, '%d', a(i,j));
        fprintf(fileID, '\t');
    end

    for j = 1 : length(V)

```

```

if i == 1
    y=V(j);
else
    y=a(i-1,j+40);
end

x=a(i,(j+40)-1);

a(i,j+40)=qw(x,y);
fprintf(fileID,'%d',a(i,j+40));
fprintf(fileID,'\t');

end

fprintf(fileID,'\n');
end

for i = 1 : length(K)

    fprintf(fileID,'*');
    fprintf(fileID,'%d',K(i));
    fprintf(fileID,'\t');
    fprintf(fileID,'%d',K(i));
    fprintf(fileID,'\t\t');

for j = 1 : limit
for l = 1 : limit
    qw(j,l)=Q(j,l,(K(i)));

end
end

for j = 1 : length(K)

    y=a(40+i-1,j);

if j == 1
    x=K(41-i);

```

```

else
    x=a(40+i,j-1);
end

    a(40+i,j)=qw(x,y);
    fprintf(fileID,'%d',a(40+i,j));
    fprintf(fileID,'\t');
end

for j = 1 : length(V)

    y=a(40+i-1,40+j);

    x=a(40+i,40+j-1);

    a(i+40,j+40)=qw(x,y);
    fprintf(fileID,'%d',a(i+40,j+40));
    fprintf(fileID,'\t');
end

    fprintf(fileID,'\n');
end

cipher=Encipher(a,K,V,limit,Q);

a=cipher+1;
fclose('all');
end

```

Appendix-C

Statistical report of the NIST test applied on Edon 80.

```

-----
RESULTS FOR THE UNIFORMITY OF P-VALUES AND THE PROPORTION OF
PASSING SEQUENCES
-----
    generator is <joined.txt>
-----
    C1  C2  C3  C4  C5  C6  C7  C8  C9  C10  P-VALUE  PROPORTION
STATISTICAL TEST
-----
    0   1   2   0   0   0   1   1   0   0      ----      5/5
Frequency
    0   0   0   1   2   0   1   1   0   0      ----      5/5
BlockFrequency
    0   1   1   1   1   0   0   0   0   1      ----      5/5
CumulativeSums
    0   2   1   0   0   0   1   1   0   0      ----      5/5
CumulativeSums
    0   0   2   1   0   0   0   0   1   1      ----      5/5
Runs
    2   0   0   0   0   0   0   1   0   2      ----      5/5
LongestRun
    1   1   0   1   0   1   0   1   0   0      ----      5/5
Rank
    0   0   2   0   0   0   0   0   2   1      ----      5/5
FFT
    0   0   0   2   1   0   1   0   0   1      ----      5/5
NonOverlappingTemplate
    0   0   1   1   1   0   0   0   0   2      ----      5/5
NonOverlappingTemplate
    1   0   0   1   0   0   0   1   1   1      ----      5/5
NonOverlappingTemplate
    1   0   0   0   1   0   1   0   0   2      ----      5/5
NonOverlappingTemplate
    0   0   0   1   0   1   1   0   1   1      ----      5/5
NonOverlappingTemplate
    0   0   1   0   0   0   2   0   0   2      ----      5/5
NonOverlappingTemplate
    0   1   0   0   0   1   0   2   1   0      ----      5/5
NonOverlappingTemplate
    0   0   0   0   1   1   0   1   0   2      ----      5/5
NonOverlappingTemplate
    0   0   1   1   0   1   2   0   0   0      ----      5/5
NonOverlappingTemplate

```

0	1	0	0	0	0	2	0	1	1	----	5/5
NonOverlappingTemplate											
0	0	1	0	1	0	0	1	1	1	----	5/5
NonOverlappingTemplate											
0	0	0	1	0	0	2	1	0	1	----	5/5
NonOverlappingTemplate											
1	0	0	0	2	0	1	1	0	0	----	5/5
NonOverlappingTemplate											
1	1	0	0	0	0	0	1	1	1	----	5/5
NonOverlappingTemplate											
0	0	0	2	0	0	0	1	2	0	----	5/5
NonOverlappingTemplate											
1	0	1	1	0	1	1	0	0	0	----	5/5
NonOverlappingTemplate											
0	1	0	0	0	2	0	0	1	1	----	5/5
NonOverlappingTemplate											
0	0	2	1	0	0	0	1	0	1	----	5/5
NonOverlappingTemplate											
0	0	0	1	0	0	0	0	0	4	----	5/5
NonOverlappingTemplate											
1	1	1	1	0	0	0	0	0	1	----	5/5
NonOverlappingTemplate											
0	2	0	1	0	0	1	0	1	0	----	5/5
NonOverlappingTemplate											
1	0	2	0	1	0	0	0	0	1	----	5/5
NonOverlappingTemplate											
1	1	0	0	0	1	1	1	1	0	----	5/5
NonOverlappingTemplate											
2	0	1	0	0	1	0	0	1	0	----	5/5
NonOverlappingTemplate											
2	0	0	0	1	0	0	0	0	2	----	4/5
NonOverlappingTemplate											
0	1	1	0	1	0	1	0	1	0	----	5/5
NonOverlappingTemplate											
0	0	1	1	0	1	0	1	0	1	----	5/5
NonOverlappingTemplate											
0	1	1	1	1	0	0	0	1	0	----	5/5
NonOverlappingTemplate											
2	0	0	0	0	1	0	1	0	1	----	4/5
NonOverlappingTemplate											
0	0	0	1	0	1	0	1	2	0	----	5/5
NonOverlappingTemplate											
0	0	2	1	0	1	0	0	1	0	----	5/5
NonOverlappingTemplate											
0	1	0	0	0	1	0	0	2	1	----	5/5
NonOverlappingTemplate											
2	0	1	0	0	0	0	1	0	1	----	5/5
NonOverlappingTemplate											
0	0	0	1	0	1	0	0	1	2	----	5/5
NonOverlappingTemplate											

0	0	0	1	0	0	1	2	0	1	----	5/5
NonOverlappingTemplate											
0	0	1	0	0	2	0	1	1	0	----	5/5
NonOverlappingTemplate											
1	0	1	1	0	0	1	0	0	1	----	5/5
NonOverlappingTemplate											
1	1	0	0	0	1	1	0	0	1	----	5/5
NonOverlappingTemplate											
0	0	2	0	0	0	1	1	1	0	----	5/5
NonOverlappingTemplate											
1	2	0	0	0	0	1	0	0	1	----	5/5
NonOverlappingTemplate											
0	2	0	1	1	0	0	0	0	1	----	5/5
NonOverlappingTemplate											
0	0	1	1	1	0	0	1	0	1	----	5/5
NonOverlappingTemplate											
0	1	1	0	1	0	0	1	1	0	----	5/5
NonOverlappingTemplate											
0	1	1	2	1	0	0	0	0	0	----	5/5
NonOverlappingTemplate											
0	0	1	0	0	1	1	1	0	1	----	5/5
NonOverlappingTemplate											
0	1	0	2	0	1	0	0	0	1	----	5/5
NonOverlappingTemplate											
1	0	1	1	1	0	0	0	1	0	----	5/5
NonOverlappingTemplate											
1	0	0	1	1	0	0	0	1	1	----	5/5
NonOverlappingTemplate											
2	2	0	0	0	0	0	0	0	1	----	5/5
NonOverlappingTemplate											
0	1	0	0	0	1	0	2	0	1	----	5/5
NonOverlappingTemplate											
1	1	1	0	0	0	0	1	1	0	----	5/5
NonOverlappingTemplate											
1	0	0	1	0	0	2	1	0	0	----	5/5
NonOverlappingTemplate											
0	0	1	2	1	0	0	1	0	0	----	5/5
NonOverlappingTemplate											
0	0	0	2	1	0	0	0	2	0	----	5/5
NonOverlappingTemplate											
0	0	0	1	0	3	0	0	0	1	----	5/5
NonOverlappingTemplate											
1	0	3	0	0	1	0	0	0	0	----	5/5
NonOverlappingTemplate											
1	0	0	0	1	0	1	1	0	1	----	5/5
NonOverlappingTemplate											
0	0	1	1	1	0	0	0	2	0	----	5/5
NonOverlappingTemplate											
1	0	0	0	0	1	1	0	0	2	----	5/5
NonOverlappingTemplate											

0	1	0	0	2	1	1	0	0	0	----	5/5
NonOverlappingTemplate											
0	0	0	0	0	3	0	1	1	0	----	5/5
NonOverlappingTemplate											
1	0	0	1	0	0	0	2	1	0	----	5/5
NonOverlappingTemplate											
2	0	0	1	0	0	1	0	0	1	----	5/5
NonOverlappingTemplate											
0	1	1	0	1	0	1	0	0	1	----	5/5
NonOverlappingTemplate											
1	0	0	0	0	1	1	1	1	0	----	5/5
NonOverlappingTemplate											
0	1	0	0	1	1	0	0	0	2	----	5/5
NonOverlappingTemplate											
0	0	0	2	0	1	1	1	0	0	----	5/5
NonOverlappingTemplate											
0	0	0	0	1	2	1	0	1	0	----	5/5
NonOverlappingTemplate											
1	0	0	0	1	1	0	1	0	1	----	5/5
NonOverlappingTemplate											
0	0	2	2	0	0	0	0	1	0	----	5/5
NonOverlappingTemplate											
0	0	0	0	1	1	0	2	0	1	----	5/5
NonOverlappingTemplate											
0	1	0	0	1	0	2	0	0	1	----	5/5
NonOverlappingTemplate											
1	1	0	1	0	0	0	1	1	0	----	5/5
NonOverlappingTemplate											
2	0	0	0	0	0	1	1	1	0	----	5/5
NonOverlappingTemplate											
0	0	0	2	1	0	1	0	0	1	----	5/5
NonOverlappingTemplate											
0	0	0	0	0	0	3	1	1	0	----	5/5
NonOverlappingTemplate											
2	1	0	0	0	0	0	0	2	0	----	5/5
NonOverlappingTemplate											
0	0	0	0	0	1	1	1	0	2	----	5/5
NonOverlappingTemplate											
2	0	1	0	1	1	0	0	0	0	----	5/5
NonOverlappingTemplate											
0	1	0	1	0	1	1	0	1	0	----	5/5
NonOverlappingTemplate											
0	0	0	0	2	0	0	0	1	2	----	5/5
NonOverlappingTemplate											
0	0	0	0	2	1	1	1	0	0	----	5/5
NonOverlappingTemplate											
1	1	0	1	0	1	1	0	0	0	----	5/5
NonOverlappingTemplate											
0	0	0	1	1	1	1	0	1	0	----	5/5
NonOverlappingTemplate											

0	0	0	0	0	2	2	1	0	0	----	5/5
NonOverlappingTemplate											
1	0	0	1	0	1	0	1	0	1	----	5/5
NonOverlappingTemplate											
0	0	0	0	0	1	1	0	0	3	----	5/5
NonOverlappingTemplate											
0	2	0	0	1	1	1	0	0	0	----	5/5
NonOverlappingTemplate											
0	1	0	1	0	0	1	1	0	1	----	5/5
NonOverlappingTemplate											
0	0	0	1	1	0	0	2	1	0	----	5/5
NonOverlappingTemplate											
1	1	0	0	0	1	0	0	1	1	----	5/5
NonOverlappingTemplate											
1	0	1	2	0	0	0	0	1	0	----	5/5
NonOverlappingTemplate											
0	1	0	0	0	1	2	0	1	0	----	5/5
NonOverlappingTemplate											
0	0	2	1	0	0	0	1	1	0	----	5/5
NonOverlappingTemplate											
0	1	1	2	0	0	0	0	1	0	----	5/5
NonOverlappingTemplate											
0	0	1	1	2	0	0	0	0	1	----	5/5
NonOverlappingTemplate											
0	1	0	1	0	0	1	0	2	0	----	5/5
NonOverlappingTemplate											
1	1	1	0	0	0	1	1	0	0	----	5/5
NonOverlappingTemplate											
1	0	2	1	0	0	0	1	0	0	----	4/5
NonOverlappingTemplate											
1	0	2	1	0	0	0	0	0	1	----	5/5
NonOverlappingTemplate											
0	0	0	2	1	1	1	0	0	0	----	5/5
NonOverlappingTemplate											
0	0	0	0	1	3	0	0	1	0	----	5/5
NonOverlappingTemplate											
1	0	1	1	0	1	0	1	0	0	----	5/5
NonOverlappingTemplate											
0	0	0	0	0	0	0	3	2	0	----	5/5
NonOverlappingTemplate											
0	0	0	2	0	1	1	0	1	0	----	5/5
NonOverlappingTemplate											
1	0	1	1	0	0	0	1	1	0	----	5/5
NonOverlappingTemplate											
0	0	1	1	2	0	0	1	0	0	----	5/5
NonOverlappingTemplate											
1	0	1	1	0	1	0	0	1	0	----	5/5
NonOverlappingTemplate											
1	1	0	0	1	1	0	1	0	0	----	5/5
NonOverlappingTemplate											

0	1	0	1	0	0	1	1	0	1	----	5/5
NonOverlappingTemplate											
0	0	0	0	2	2	0	0	1	0	----	5/5
NonOverlappingTemplate											
0	2	0	1	0	1	0	1	0	0	----	5/5
NonOverlappingTemplate											
0	1	1	1	0	1	0	0	0	1	----	5/5
NonOverlappingTemplate											
0	1	1	0	2	0	0	0	1	0	----	5/5
NonOverlappingTemplate											
0	0	1	0	0	1	1	2	0	0	----	5/5
NonOverlappingTemplate											
0	0	2	0	0	0	0	0	2	1	----	5/5
NonOverlappingTemplate											
1	0	1	1	0	1	0	1	0	0	----	5/5
NonOverlappingTemplate											
0	1	0	0	0	1	0	0	2	1	----	5/5
NonOverlappingTemplate											
0	1	0	1	0	1	1	0	1	0	----	5/5
NonOverlappingTemplate											
1	1	0	2	0	0	0	0	1	0	----	5/5
NonOverlappingTemplate											
1	1	2	0	0	0	0	1	0	0	----	4/5
NonOverlappingTemplate											
1	0	1	0	1	1	1	0	0	0	----	5/5
NonOverlappingTemplate											
0	2	0	0	0	0	0	2	1	0	----	5/5
NonOverlappingTemplate											
0	0	0	1	1	0	1	0	2	0	----	5/5
NonOverlappingTemplate											
1	0	0	0	0	1	0	2	0	1	----	5/5
NonOverlappingTemplate											
0	0	2	1	0	1	0	0	0	1	----	5/5
NonOverlappingTemplate											
1	0	1	0	0	1	0	0	0	2	----	5/5
NonOverlappingTemplate											
0	0	0	0	2	1	1	0	0	1	----	5/5
NonOverlappingTemplate											
0	0	0	1	1	0	0	0	3	0	----	5/5
NonOverlappingTemplate											
0	1	0	0	1	1	0	2	0	0	----	5/5
NonOverlappingTemplate											
0	0	1	1	0	1	1	1	0	0	----	5/5
NonOverlappingTemplate											
0	0	5	0	0	0	0	0	0	0	----	5/5
NonOverlappingTemplate											
0	0	1	0	0	2	1	0	0	1	----	5/5
NonOverlappingTemplate											
2	0	1	1	0	0	1	0	0	0	----	5/5
NonOverlappingTemplate											

0 1 1 0 1	0	0	0	0	2	----	5/5
NonOverlappingTemplate							
2 0 0 0 2	1	0	0	0	0	----	5/5
NonOverlappingTemplate							
0 0 0 0 2	0	0	1	0	2	----	5/5
NonOverlappingTemplate							
0 0 0 0 1	0	0	1	2	1	----	5/5
NonOverlappingTemplate							
0 0 0 0 0	1	1	0	2	1	----	5/5
NonOverlappingTemplate							
0 1 0 2 0	2	0	0	0	0	----	5/5
NonOverlappingTemplate							
1 0 1 0 2	0	0	1	0	0	----	5/5
NonOverlappingTemplate							
1 0 1 1 0	0	1	0	0	1	----	5/5
NonOverlappingTemplate							
0 1 2 1 0	0	0	1	0	0	----	5/5
NonOverlappingTemplate							
0 0 0 0 0	1	0	1	2	1	----	5/5
NonOverlappingTemplate							
1 0 0 1 1	1	1	0	0	0	----	5/5
NonOverlappingTemplate							
0 3 0 1 0	1	0	0	0	0	----	5/5
NonOverlappingTemplate							
0 2 0 0 0	0	1	0	2	0	----	5/5
NonOverlappingTemplate							
2 0 0 0 0	0	1	1	1	0	----	5/5
NonOverlappingTemplate							
0 0 0 0 1	1	0	2	1	0	----	5/5
OverlappingTemplate							
0 1 0 1 0	0	1	0	1	1	----	5/5
Universal							
0 0 1 1 1	0	1	0	0	1	----	5/5
ApproximateEntropy							
0 0 0 0 0	1	0	0	2	0	----	3/3
RandomExcursions							
0 1 0 0 0	0	0	0	1	1	----	3/3
RandomExcursions							
0 0 0 1 0	0	1	0	0	1	----	3/3
RandomExcursions							
1 0 0 1 0	0	0	0	0	1	----	3/3
RandomExcursions							
0 0 1 1 0	1	0	0	0	0	----	3/3
RandomExcursions							
0 0 1 0 0	0	1	0	1	0	----	3/3
RandomExcursions							
1 0 0 0 0	1	0	0	0	1	----	3/3
RandomExcursions							
0 0 1 0 1	0	0	1	0	0	----	3/3
RandomExcursions							

1	0	0	0	0	0	0	1	1	0	----	3/3
RandomExcursionsVariant											
1	0	0	0	0	0	1	0	0	1	----	3/3
RandomExcursionsVariant											
1	0	0	0	0	0	0	0	1	1	----	3/3
RandomExcursionsVariant											
0	0	1	0	0	0	0	2	0	0	----	3/3
RandomExcursionsVariant											
0	0	0	0	2	0	0	0	1	0	----	3/3
RandomExcursionsVariant											
0	0	0	0	1	0	0	0	2	0	----	3/3
RandomExcursionsVariant											
0	0	0	0	0	1	0	2	0	0	----	3/3
RandomExcursionsVariant											
0	0	0	0	2	0	0	0	0	1	----	3/3
RandomExcursionsVariant											
0	1	1	0	0	0	1	0	0	0	----	3/3
RandomExcursionsVariant											
0	0	1	0	0	2	0	0	0	0	----	3/3
RandomExcursionsVariant											
0	0	1	1	1	0	0	0	0	0	----	3/3
RandomExcursionsVariant											
0	0	1	0	0	1	1	0	0	0	----	3/3
RandomExcursionsVariant											
0	0	0	0	1	1	1	0	0	0	----	3/3
RandomExcursionsVariant											
0	0	1	1	0	1	0	0	0	0	----	3/3
RandomExcursionsVariant											
0	2	0	0	0	0	1	0	0	0	----	3/3
RandomExcursionsVariant											
1	1	0	0	0	0	0	1	0	0	----	3/3
RandomExcursionsVariant											
2	0	0	0	0	0	0	0	0	1	----	3/3
RandomExcursionsVariant											
2	0	0	0	0	0	1	0	0	0	----	3/3
RandomExcursionsVariant											
1	0	1	1	0	0	0	1	1	0	----	5/5
Serial											
2	1	0	0	0	0	0	0	2	0	----	5/5
Serial											
0	0	0	1	0	1	1	2	0	0	----	5/5
LinearComplexity											

The minimum pass rate for each statistical test with the exception of the random excursion (variant) test is approximately = 4 for a sample size = 5 binary sequences.

The minimum pass rate for the random excursion (variant) test

is approximately = 2 for a sample size = 3 binary sequences.

For further guidelines construct a probability table using the
MAPLE program
provided in the addendum section of the documentation.

- - - - -

References

1. Chakrabarti S., Pal S. K., Gangopadhyay S.: An Improved 3-ary quasigroup Based Encryption Scheme, ICT Innovations Conference 2012 on Secure and Intelligent Systems, Macedonia, Web proceedings ISSN 1857-7288, 173-184, 2012.
2. Chakrabarti S., Pal S.K.: On Increasing Key Space of Quasigroup Based Ciphers, presented in National Workshop on Cryptology, India, 2013.
3. Denes J., Keedwell A.D. : LatinSquares. New Development in the Theory and Applications, Vol 46, Annals of Discrete Mathematics, North –Holland, 1991.
4. Gligorski D., Markovski S., and Knapskog S. J. : The Stream Cipher Edon 80, Stream Cipher Designs : The eSTREAM Finalists , LNCS, Vol. 4986, pp. 152-169, 2008.
5. Hardy, G. H., Wright, E. M., “An Introduction to theory of Numbers”, fourth edition. Published by Oxford University Press.
6. Kościelny C., Kurkowski M., Srebny M.: Modern Cryptography Premier: Theretical Foundations and Practical Applications, Springer Verlag Barlin, 2013.
7. Kościelny C.: A Method of Constructing Quasigroup – based Stream Ciphers, Int. Journal Applied Math and Computational Sciences, Vol. 6, No. 1, 1996, pp. 109-121, 1996.
8. Lidl R., Niederreiter H.: Finite Fields, Encyclopedia of mathematics and its Applications, Vol. 20 , Cambridge Univ. Press, 1997.
9. Malik D.S, Mordeson, Sen M.K: Fundamentals Of Abstract Algebra, McGraw Hill, 1997
10. Markovski S. and Kusakatav V. : Quasigroup String Processing-Part 2, Contributions, Sec.Math.Tech.Sci. MANU XXI,1-2, pp. 15-32, 2000
11. Markovski S., Gligoroski D. and Bakeva V. : Quasigroup String Processing-Part 1, Contributions, Sec.Math.Tech.Sci. MANU XX,1-2, pp. 13-28, 1999
12. Markovski S. and Kusakatav V. : Quasigroup String Processing-Part 3, Contributions, Sec.Math.Tech.Sci. MANU XXI,1-2, pp. 7-27, 2002-2003
13. Markovski S. and Bakeva V.: Quasigroup String Processing-Part 4, Contributions, Sec.Math.Tech.Sci. MANU XXVII-XXVIII,1-2, pp. 41-53, 2006-2007.

14. Menezes A., VanOorschot P. and Vanstone S. : Handbook of Applied Cryptography, CRC press, 1996.
15. Mileva A.: Cryptographic Primitives with Quasigroup Transformation, PhD Thesis, Faculty of natural science, Ss Cyril and Methodius University in Skopje, Republic of Macedonia, 2010.
16. NIST ,Technology administration, US Dept. : A Statistical test suite for random and pseudorandom number generator for cryptographic applications.
17. Paar C ,and Pelzl J. : Understanding of Cryptography, Springer, 2010.
18. Shcherbacov V.A. : Quasigroups in Cryptology, Computer science journal of Moldova, Vol.17, No2,pp.193-228, 2009.
19. Smith J.D.H., : An Introduction to quasigroups and their representations , Chapman & Hall
CRC, 2007.
20. Stanley B., Sankappanavar H.P.: A Course in Universal Algebra, Springer –Verlag, 1981.