5-2014

# The Quasigroup Block Cipher and its Analysis

Matthew J. Battey
*University of Nebraska at Omaha*

# The Quasigroup Block Cipher and its Analysis

A Thesis

Presented to the

Department of Computer Sience

and the

Faculty of the Graduate College

University of Nebraska

In partial satisfaction of the requirements for the degree of

Masters of Science

by

Matthew J. Battey

May, 2014

Supervisory Committee:
Abhishek Parakh, Co-Chair
Haifeng Guo, Co-Chair
Kenneth Dick
Qiuming Zhu




# Abstract

**The Quasigroup Block Cipher and its Analysis**

Matthew J. Battey, MS

University of Nebraska, 2014

Advisor: Abhishek Parakh

This thesis discusses the Quasigroup Block Cipher (QGBC) and its analysis. We first present the basic form of the QGBC and then follow with improvements in memory consumption and security. As a means of analyzing the system, we utilize tools such as the NIST Statistical Test Suite, auto and crosscorrelation, then linear and algebraic cryptanalysis. Finally, as we review the results of these analyses, we propose improvements and suggest an algorithm suitable for low-cost FPGA implementation.

# Contents

# List of Figures

# List of Tables

# Chapter 1

# Introduction

Continuous research into novel cryptosystems is necessary, as past systems become vulnerable due to increased computational power and cryptanalysis that have identified weakness in existing systems[2][3]. For this reason, we propose and demonstrate a novel cryptosystem based on quasigroup polyalphabetic substitution.

In this thesis, we will first introduce the reader to quasigroups. Then in chapter 2, we define our Quasigroup Block Cipher (QGBC), which was specifically designed to overcome limitations from prior work[4]. Next in chapter 3, we discuss means for limiting the resources consumed by a quasigroup. From here, we expand the QGBC and utilize it as a pseudo random number generator, in chapter 4. Then in chapter 5 , we explore the cryptographic analysis of the QGBC. Which finally leads us to chapter 6, where we identify improvements based on the results of cryptanalysis, and further refine the memory requirements needed by the system.

In this chapter we will formulate baseline knowledge of quasigroups, first gaining background in § 1.2. Then we explore prior work where quasigroups are used in cryptography in § 1.3 and finally in § 1.4 we discuss the theoretical security of the quasigroup.

## 1.1 Research Motivation and Goals

Discovery, research, and enhancement of cryptographic systems has proven to not only be a curiosity but a necessity. History has shown that for every cryptographic measure a counter measure has been found to disable it. In some cases, human error and social engineering are the downfall, but in other cases cryptanalysis has shown weaknesses in systems.

Many cryptosystems were designed prior to the advent of low-cost, low-power computing equipment which are spawning the "Internet of Things." These devices not only need to be protected from each other, but from main-line computing system with much more processing capability and resources. While these systems often have more processing power than large computer system from just a decade ago, they often run on battery power, so the need for highly effective cryptographic measures that require little power to accomplish is a future need.

Starting in 1994, the National Institute for Standards and Technology (NIST) first proposed Secure Hash Algorithm #1 (SHA-1) as a means of random number generation [5]. In 2004, NIST released a brief on SHA-1, noting that a weakened variant of the system had been broken, and suggested that software vendors move to newer alorithms[2]. In another case, RC4 one of the most commonly used ciphers for secure internet traffic has shown similar issues[3]. Further, the triple DES cryptosystem has shown weaknesses to linear analysis[6].

For these reasons we set out with the following goals:

**Develop a novel block cipher** Block ciphers tend to be more reliable in distributed communication environments as well as provide additional security

**Utilize standard key sizes** Using standard keys allows us to re-use key exchange mechanisms and utilize current conventions.

**Pass the NIST-STS suite** NIST developed the Statistical Test Suite[7], which combines a number of pre-defined statistical test to evaluate the entropy generated by a crypto system.

Based on these goals, we chose to enhance the Quasigroup Stream Cipher[8] and use it as the basis for a new block cipher.

## 1.2 Background on Quasigroups

A Quasigroup is an algebraic structure similar to a group. However, a quasigroup does not necessarily possess the axioms of identity and associativity; rather one only requires closure and inversion. Quasigroups have been part of popular culture, making their way into games like Sudoku, where they are represented as Latin squares. This thesis focuses on the quasigroup for two reasons, which are important to cryptography. First, one closure and inversion make handy tools for making cipher/decipher pairs, and secondly, the lack of associativity is the beginnings of a one-way function. The following list will be used as a guide describing expressions throughout this thesis.

$N$ – A quasigroup's order, or number of distinct elements
$M$ – Plain text, information which is directly readable
$M_i$ – A single word from the Plain text (sometimes called Message Text)
$m_i$ – A single bit form the Plain text
$C$ – Cipher text, information which has been encrypted so that is is not plainly readable
$C_x$ – A single word from the Cipher text
$c_i$ – A single bit from the cipher text
$K$ – Key text, a sequence of bits
$K_x$ – A single word from the Key text
$k_i$ – A single bit from the key text
$w$ – The size of a word in bits
$|M|, |C|, |K|$ – The number of words in $M$, $C$, or $K$

Also for this thesis, we will denote $\cdot$ and $\circ$ as the primary quasigroup operation and its inverse. Table 1.1 is an example of a quasigroup order $N = 4$ rendered as a Latin square. For a given row or column, each element appears only once; which alternately allows the quasigroup to be represented as a set of ordered triples; such as the following, which describes the first row in tbl. 1.1:

$\{(0,0,2),(0,1,0),(0,2,3),(0,3,1),...\}$. Membership in a quasigroup can also be defined by a known mathematical operation, such as addition modulo $N$ or bit-wise exclusive-or.

| | 0 | 1 | 2 | 3 |
|---|---|---|---|---|
| 0 | 2 | 0 | 3 | 1 |
| 1 | 3 | 2 | 1 | 0 |
| 2 | 0 | 1 | 2 | 3 |
| 3 | 1 | 3 | 0 | 2 |

Table 1.1: Sample quasigroup order $N = 4$

### 1.2.1 Quasigroup Computation

Computation of the quasigroup operation may be performed by a look-up from the ordered triples set or Latin square. Say we wish to compute $2 \cdot 3$ using the Latin square approach; we first look in row 2 then column 3 and find our answer is $2 \cdot 3 = 3$ (the first row and column are the 0'th). To perform $1 \circ 1$, we look in row 1 and find the column containing 1. We see that 1 is found in column 2, thus $1 \circ 1 = 2$. To reduce computation, it is possible to produce an equivalent inverse quasigroup, which lets one use the row and column indices as in the forward quasigroup.

### 1.2.2 Equivalent Classes of Latin Squares

Latin square equivalence classes [9] are those squares that are related by some simple transformation. One example is to add 1 modulo $N$ to every element. The equivalence class we are interested in is one where we rearrange the members of the ordered triple found in the orthogonal array representation, to from an inverse quasigroup. Here, we transpose $(r_i, c_j, v_{ij})$ with $(r_i, v_{ij}, c_j)$ of our Quasigroup. This is a valid transposition producing an equivalent Quasigroup. We know this to be true by the very nature of the Quasigroup's Latin square definition. Tables 1.2 and 1.3 are transpositions of each other.

Later in this thesis we make reference to the number of quasigroups, by count-

ing the number of Latin squares (see § 1.4). These counts are given based on the number of Reduced Latin squares possible for a given $N$. Here a special equivalence class is used required, where whole columns and rows are swapped; such that the first row and column are sorted in ascending order. The remaining elements can be present in any order as long as they conform to the properties of a Latin square.

## 1.3 Quasigroups in Cryptography

The quasigroup operation allows us to make polyalphabetic substitutions, and this property has been used in ciphers for more than four-hundred years. In 1585, Blaise de Vigenere constructed a Latin square of the same order as his target language (i.e. $N = 026$ for English) [10]. He then proposed a key word that would select the cipher text (or pad-text in today's language). This cipher was considered unbreakable until 1863, when it was discovered that for a large enough plain text the cipher text demonstrated repetitions. Although this cipher received Vigenere namesake, Giovan Battista Bellaso had actually published the cipher 1553.

The matrix used for one-time pad (OTP) is also a quasigroup, and is also known as a bitwise exclusive-or (XOR). Frank Miller first described this cryptographic system in 1882 [11], then again by Gilbert Vernal in 1917 [12] [13], where it was then patented. We know the OTP to have perfect security, as long as the pad is of the same length as the plain text, random, uniform, and independent of the input.

Within the last decade, further research has gone into the polyalphabetic substitution properties of quasigroups. Gligoroski and Markovski (G&M) report cryptographic potentials of matrix quasigroups and suggest a stream cipher [4]. With this system the quasigroup remains secret and is pre-shared between communication partners. A published seed word is combined with the first word in the plain text. Then subsequent plain text words are combined with the previous cipher

text word. This stream cipher takes the form $C_0 = s \cdot M_0, C_i = C_{i-1} \cdot M_i, i \geq 1$, which chains each output byte to the previous (we will see an expanded explanation below, in § 1.3.1). The strength of security is based on the order of the quasigroup selected; lending form the raw number of quasigroups to choose from.

Then with Kocarev, Gligoroski and Markovski explore the potentials of removing the bias from poor PRNG systems by utilizing a quasigroup stream cipher to further randomize the data [14]. Here a weak random sequence generator such as libC's `random()` passed through G&M's stream cipher, with the goal of improving the data distribution of the driver. This method suffers in statistical evaluation however [15].

Satti and Kak envision a quasigroup cryptosystem for both data and speech in their paper [8]. Their research applies G&M's stream cipher to a number of practical inputs such as English text, constant values, and PCM audio data. They demonstrate success through autocorrelation techniques as well as propose systems for distribution of the quasigroup and implementation in communication devices.

### 1.3.1 Quasigroup Stream Cipher Encryption

Consider the criticality of the equivalence of the base Quasigroup and its inverse. The equations found in 1.1 construct the protocol for a Quasigroup based stream cipher. Using the $\cdot$ operator, we can quickly and efficiently re-encode plain text to cipher text. Using the equivalent inverse Quasigroup, we can reverse that process. We have constructed a encryption cipher with its corresponding decryption function!

| · | 1 | 2 | 3 | 4 | 5 | 6 |
|---|---|---|---|---|---|---|
| 1 | 1 | 3 | 2 | 6 | 4 | 5 |
| 2 | 2 | 6 | 4 | 5 | 1 | 3 |
| 3 | 3 | 2 | 6 | 4 | 5 | 1 |
| 4 | 4 | 5 | 1 | 3 | 2 | 6 |
| 5 | 5 | 1 | 3 | 2 | 6 | 4 |
| 6 | 6 | 4 | 5 | 1 | 3 | 2 |

Table 1.2: A quasigroup of order 6.

**Example 1:** Table 1.2 presents a quasigroup of order 6. The left most column and the top most row are index numbers. An initial seed element is chosen, say $s = 3$, and let the input data stream be represented by $\{M_1, M_2, M_3, M_4, M_5, M_6, M_7, M_8\} = \{1, 5, 4, 2, 6, 4, 5, 3\}$. Then the encryption process produces an encrypted output stream $\{C_1, C_2, C_3, C_4, C_5, C_6, C_7, C_8\}$ as defined by Alg. 1.

**Data**: QG[][] – A two dimensional array containing the quasigorup
**Data**: S – The Seed
**input** : M – The Plain Text
**output**: C – The Cipher Text
$C_0 = QG[S][M_0]$;
**for** $i = 1to|M|$ **do**
  $C_i = QG[C_{i-1}][M_i]$;
**end**

**Algorithm 1:** Quasigroup Stream Cipher

$$
\begin{aligned}
C_1 &= S \cdot M_1 &&= 3 \cdot 1 = 3 \\
C_2 &= C_1 \cdot M_2 &&= 3 \cdot 5 = 5 \\
C_3 &= C_2 \cdot M_3 &&= 5 \cdot 4 = 2 \\
C_4 &= C_3 \cdot M_4 &&= 2 \cdot 2 = 6 \\
C_5 &= C_4 \cdot M_5 &&= 6 \cdot 6 = 2 \\
C_6 &= C_5 \cdot M_6 &&= 2 \cdot 4 = 5 \\
C_7 &= C_6 \cdot M_7 &&= 5 \cdot 5 = 6 \\
C_8 &= C_7 \cdot M_8 &&= 6 \cdot 3 = 5
\end{aligned}
$$

(1.1) Quasigroup Stream Cipher Example

Unrolling the algorithm, with the sample data, produces the operations show in (1.1), when computed using tbl. 1.2.

### 1.3.2 Quasigroup Stream Cipher Decryption

For the decryption operation, an inverse quasigroup matrix is constructed (table 1.3). We construct the $invQG[][]$ matrix, by doing the following: in the $j^{th}$ column of the $i^{th}$ row in $invQG[][]$ matrix write the column number of element $j$ from the $i^{th}$ row in $QG[][]$. Then to decrypt we perform the algorithm shown in Alg. 2.

| ∘ | 1 | 2 | 3 | 4 | 5 | 6 |
|---|---|---|---|---|---|---|
| 1 | 1 | 3 | 2 | 5 | 6 | 4 |
| 2 | 5 | 1 | 6 | 3 | 4 | 2 |
| 3 | 6 | 2 | 1 | 4 | 5 | 3 |
| 4 | 3 | 5 | 4 | 1 | 2 | 6 |
| 5 | 2 | 4 | 3 | 6 | 1 | 5 |
| 6 | 4 | 6 | 5 | 2 | 3 | 1 |

Table 1.3: Inverse for the quasigroup in Table 1.2

**Data**: invQG[][] – A two dimensional array containing the inverse quasigroup
**Data**: S – The Seed
**input** : M – The Plain Text
**output**: C – The Cipher Text
$M_0 = invQG[s][c_0]$;
**for** $i = 1to|C|$ **do**
  $M_i = invQG[C_{i-1}][C_i]$;
**end**

**Algorithm 2:** Quasigroup Stream Cipher Decryption

In general, the direct application of the above encryption algorithm is very effective in randomizing the input data stream. However, given an input data stream and its corresponding output data stream a known plain text attack can be launched because $QG[C_{i-1}][M_i] = C_i$, which directly leads the attacker to assess the quasigroup's definition. Consequently, qausigroups as stream ciphers may provide only limited security, if ever the attacker were to gain knowledge of the quasigroup itself.

## 1.4 On Theoretical Security of Quasigroup Ciphers

The total number of Latin squares of order $N$, $N > 2$, is given by $L_N = N!(N-1)!T_N$, where $T_N$ denotes the number of reduced Latin squares of order $n$. The numbers $T_N$ and $L_N$ increase very quickly with $N$ [8]. Table 1.4 gives the number of reduced Latin squares.

From table 1.5 we see that the number of possibilities for the Latin squares is astronomical. Therefore, if the quasigroup is kept secret along with the 256 bit key (32 random seeds) the system provides very good security.

| $N$ | $T_N$ |
|---|---|
| 2 | 1 |
| 3 | 1 |
| 4 | 4 |
| 5 | 56 |
| 6 | 9048 |
| 7 | 16942080 |
| 8 | 535281401585 |
| 9 | 377597570964258 |
| 10 | 7580721483160132811489280 |
| 11 | $5.36 \times 10^{33}$ |
| 12 | $1.62 \times 10^{44}$ |
| 13 | $2.51 \times 10^{56}$ |
| 14 | $2.33 \times 10^{70}$ |
| 15 | $1.50 \times 10^{86}$ |

Table 1.4: Number of reduced Latin squares of order 2 to 15.

| |
|---|
| $0.689 \times 10^{138} \geq LS(16) \geq 0.101 \times 10^{119}$ |
| $0.985 \times 10^{785} \geq LS(32) \geq 0.414 \times 10^{726}$ |
| $0.176 \times 10^{4169} \geq LS(64) \geq 0.133 \times 10^{4008}$ |
| $0.164 \times 10^{21091} \geq LS(128) \geq 0.337 \times 10^{20666}$ |
| $0.753 \times 10^{102805} \geq LS(256) \geq 0.304 \times 10^{101724}$ |

Table 1.5: Bounds for number of Latin squares for orders 16, 32, 64, 128 and 256.

# Chapter 2

# Quasigroup Block Cipher

In this chapter we will examine the Quasigroup Block Cipher (QGBC) that was designed as part of this research effort. We will first cover the algorithm with out Cipher Block Chaining (CBC) and the statistical analysis of the test implementation (see § 2.1). Then we will explore a modification that enables CBC and outline the statistical analysis of the QGBC-CBC based test implementation (see § 2.2).

## 2.1 Proposed Algorithm 1: Quasigroup Block Cipher

Our goal is to make a quasigroup cipher similar in functionality to the popular AES system. To this end, we use 32 different seeds for each round of encryption. Utilizing multiple rounds of encryption, with different seeds in different rounds, finesses the known-plain-text attack and provides a higher level of security; just as in the case of Triple DES and AES. We choose 32 seeds, because we assume that each seed is one byte in size and 32 bytes is quivalent to 256 bits, which is the commonly used key length for AES systems.

In order to introduce dependencies between bytes of input data, we divide the plain text into 128 bit (16 byte) blocks and encrypt each block separately using Algorithm 3, below.

**input** : Cipher Key – 256 bits
**input** : Plain Text – A stream of 128 bit blocks
**Result**: Plain Text encoded as Cipher Text
Construct a 256x256 size quasigroup;
Generate a random 256 bit encryption key and divide it into 8 bit (1 byte) blocks which will be used as seed elements at every round of encryption. This results in 32, 1 byte, seeds;
Divide the source data into 128 bit (16 byte) blocks;
**for** *Each* block *of* Plain Text **do**
    **for** *Each 8-bit word in the* Cipher Key **do**
        Using the current *block* as a stream of 16, 8-bit integers, apply the current 8-bit key as the quasigroup cipher seed and encrypt the block;

        Left shift the currently encrypted block by 1, 3, 5 or 7 bits depending on the index of the current 8-bit key block modulo 4;
    **end**
**end**

**Algorithm 3:** Quasigroup Block Cipher

Note that although each block is 128 bits long, when applying quasigroup encryption we further divide the block into 16, 1 byte sub-block. Then we apply algorithm 1 to the block once for each word in the Cipher Key. After every round of encryption (application of the stream cipher), all the bits are taken together and then left-rotated. A pseudo code is given below:

```
define : BlockSize = 16
define : KeySize = 32
input  : PlainText – The entire plain text buffer
input  : Key – An Array length KeySize
output: CipherText – The enter cipher text buffer
Data: QGMS – An Array(256,256)
Data: ShiftDistance as [1,3,5,7]
for each Block in PlainText do
    CipherText = Block;
    for  each K in Key do
        CipherText = QuasiGroupCipher(QGMS, K,CipherText)
        CipherText = LeftShift(CipherText, ShiftDistance[Key.IndexOf(K)
        Modulo 4])
    end
    Output[IndexOf(Block,Source)] = CipherText
end
```

**Algorithm 4:** Pseudo code algorithm for the QGBC

The shift distances of 1, 3, 5, and 7 are each relatively prime to 2 and thus to 8 (size of a word in this system). Their sum is 16 (size of 2 bytes) and if each shift is applied 8 times, their sum becomes 128, which is equal to the block size of 128 bits (16 bytes) into which the input data was divided. Therefore, one full rotation of block occurs with shifts of 1, 3, 5 and 7 when all the 32 seeds are used. This ensures that all the bytes in the encrypted block become interdependent. Later, in § 5.1, we will see that the shift distance is critically important.

Figure 2.1, below demonstrates this algorithm graphically. Here again, we see the process of selecting a block performing the Quasigroup transformation, bit-shifting, and repeating.

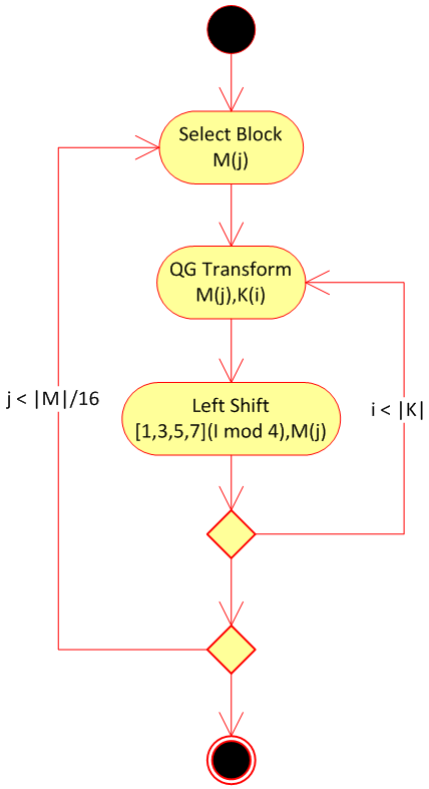



Figure 2.1: Flowchart for the quasigroup block cipher (proposed algorithm 1). Here $M$ is the entire message, $M(j)$ is the $j^{th}$ block in the message, $K$ is the key, $K(i)$ is the $i^{th}$ seed in the key string, $|M|$ is the size of message in bytes, $|K|$ is the size of key string in bytes, $i$ is the iterator of key bytes and $j$ is the iterator of message blocks.

### 2.1.1 Test Implementation

A test implementation was developed in C#.net, because of the popular adoption of C# and the pre-existing AES cipher suite. Also, Microsoft Visual Studio 2010 has built in unit-testing facilities, which combined with Test-Driven-Development, produced well-tested code in reduced increments of time. The test implementation has the ability to overwrite the plaintext buffer in place, limiting the memory footprint required to encode a buffer. Keys were generated using the random-number generator, System.Random, allocating 16 random bytes per request. Full $n \times n$ Quasigroup matrices were constructed for both encryption and decryption using the Knuth/ Fisher-Yates Shuffle [16]. Both the encryption and decryption routines were constructed and tested.

### 2.1.2 Statistical Analysis

We used the National Institute of Technology - Statistical Test Suite (NIST-STS) suite to evaluate the randomness introduced by the system in the cipher. The NIST-STS package gives a P-value and Success/Fail status for various standardized tests. Based on the null hypothesis that the tested sequence is random. Thus, the P-value is the probability that a perfect random number generator would have produced a less random sequence than the one being tested [17].

Based on the research by the NIST-STS team, each test was given a P-value threshold. When a P-value result from a test crossed these thresholds, the test was considered successful, otherwise it is flagged a failure. Control tests were performed against the plain text source (it should be noted the control failed each test). The NIST-STS test suite is available freely in C source code, and downloadable from `http://csrc.nist.gov/groups/ST/toolkit/rng/index.html`. The tool can be configured to read a source file as a stream of bits, and evaluate the randomness of that stream. We report the results for the following tests, where the parameters used for the tests are given in table 2.1:

- Approximate Entropy (AE) - A test comparing all overlapping m-bit patterns.
- Block Frequency (BF) - A test which evaluates the proportion of 1's in $m$-bit blocks.
- Cumulative Sums, Forward (CSF), Reverse (CSR) - Evaluates whether the maximal cumulative sum of partial sequences is outside the range for expected behavior of a random sequence.
- Discrete Fourier Transform (FFT) - Implemented as a Fast Fourier Transform, detects repeating or periodic features that are near to each other.
- Frequency (FREQ) - Evaluates the frequency of 1's and 0's in the entire sequence.
- Longest Run - Comparison of longest contiguous run of 1's in $m$-bit blocks to expected frequency of same.
- Rank - The rank of disjoint sub-matrices within the entire sequence.

- Runs - Finds and evaluates the longest sequence of contiguous 1's in the entire sequence and compares the oscillation between 1's and 0's to a standard frequency.
- Serial - Compares the frequency of all $m$-bit overlapping patterns in the full sequence. Two variations are applied.

| | |
|---|---|
| Block Frequency Test - block length(m) | 128 |
| Non-overlapping Template Test - block length(m) | 9 |
| Overlapping Template Test - block length(m) | 9 |
| Approximate Entropy Test - block length(m) | 10 |
| Serial Test - block length(m) | 16 |
| Linear Complexity Test - block length(m) | 500 |

Table 2.1: Parameters for the NIST-STS test

Upon completion of each test, a P-value result is rendered. If a P-value for a test is determined to be equal to 1 or 0 then an error condition has occured[17]. Otherwise, P-values greater 0.01 demonstrate the test was passed.

Initially 20 encrypted data sets each were produced, from input files containing binary zero (0x00), binary ones (0xFF) and text from Aseop's fable, "From the Goose and the Golden Eggs" for a total of 60 files.

Table 2.2 shows the P-values for the various tests. In the table the first three columns show the average P-values for all 60 files, ranking QGBC results against AES results.. The first column lists the various tests done, second column is the average P-values for encryption of all three inputs using quasigroups, third column is the average P-value for all three inputs using AES and the fourth column is the ratio of the P-value of encryption using quasigroups to that using AES multiplied by 100. The last four columns are P-values for all zero (0x00) and 0xFF inputs alone.

| Test | QGBC | AES | QG:AES | 0x00-AES | 0x00-QG | 0xFF-AES | 0xFF-QG |
|---|---|---|---|---|---|---|---|
| BF | 0.57189 | 0.53593 | 106.71% | 0.59109 | 0.57530 | 0.48253 | 0.64041 |
| CS-F | 0.47759 | 0.45340 | 105.33% | 0.47739 | 0.42955 | 0.36766 | 0.50679 |
| CS-R | 0.47995 | 0.46111 | 104.08% | 0.48052 | 0.43870 | 0.36949 | 0.49906 |
| FFT | 0.15798 | 0.15622 | 101.12% | 0.03377 | 0.043198 | 0.05215 | 0.05501 |
| FREQ | 0.40314 | 0.40006 | 100.77% | 0.38935 | 0.34988 | 0.29779 | 0.39156 |
| LR | 0.30803 | 0.29188 | 105.53% | 0.24881 | 0.21313 | 0.17118 | 0.27998 |
| Runs | 0.40384 | 0.40136 | 100.62% | 0.37347 | 0.37045 | 0.38143 | 0.35849 |

Table 2.2: The table shows average P-values (over 20 runs) for quasigroup encryption as compared to AES256 encryption system when the same encryption key is used for both cryptosystems without Cipher-Block-Chaining (CBC). Each source data set consists of 288 bytes of sample data.

### 2.1.3 On memory and computational requirements:

The $n \times n$ matrix consumes 64 KB ram. Also, test implementation was developed in such away that the input data could be directly overwritten, no additional buffers were required. As the solution is a block cipher, only one block must be in memory at any given time.

Processing efficiency is as follows; for each byte in the block, lookup the QG re-encoded value from the matrix, then left shift the block. Table 2.3 lists the number of operations necessary when encrypting data. The number of operations to decrypt is similar.

| | | |
|---|---|---|
| Encrypt: | one 2D array lookup | 1 op |
| Left shift: | two 64-bit left shift | 2 ops |
| Total Ops 16 byte block: | $3 \times 16$ | 48 ops |
| Total Ops 32 byte key: | $48 \times 32$ | 1536 ops |

Table 2.3: Operations necessary to encrypt a 16 bite block with a 32 byte key, note left shift can be greatly reduced using integers wider than 8 bits.

## 2.2 Proposed Algorithm 2: Quasigroup Block Encryption with Cipher Block Chaining

When we compared the QGBC to the AES system, we were unable to collect data for the Approximate Entropy and Serial tests. Thus to improve the performance of QGBC in these tests, we extended algorithm 4 to include cipher block chaining (CBC). Mathematically, CBC is written as:

$$C[0] := e(k, M[0] \oplus V)$$
$$C[i+1] := e(k, M[i+1] \oplus C[i])$$

Where, $C[i]$: an indexed cipher text block, $M[i]$: an indexed plain text block, $K$: the cipher key (here seed), $V$: A random initialization vector, where $|V| = |C[i]| = |M[i]|$, $e(K, M)$: the encryption function, QGBC in this case.

### 2.2.1 Test Implementation

After implementing quasigroup block cipher with cipher block chaining, tests were repeated 20 times using a 256 bit random key (32, 1 byte seeds) each time. The resulting encrypted data was tested for randomness using the NIST-STS test suite, using the same parameters as before.

Table 2.4 compares the average P-value results from the NIST-STS test suite. The quasigroup block cipher with CBC produced larger P-values than AES256 with CBC in almost all cases.

| Test | QG | AES | QG:AES | 0x00:AES | 0x00:QG | 0xFF:AES | 0xFF:QG |
|---|---|---|---|---|---|---|---|
| BF | 0.48822 | 0.51274 | 95.22 | 0.52155 | 0.47478 | 0.50250 | 0.48499 |
| CS-F | 0.51939 | 0.50588 | 102.67 | 0.50527 | 0.49851 | 0.48968 | 0.48843 |
| CS-R | 0.52502 | 0.48904 | 107.36 | 0.49205 | 0.51126 | 0.47860 | 0.49353 |
| FFT | 0.50188 | 0.48532 | 103.41 | 0.46172 | 0.48304 | 0.49187 | 0.49118 |
| FREQ | 0.50190 | 0.47353 | 105.99 | 0.48847 | 0.47584 | 0.46486 | 0.48745 |
| LR | 0.50468 | 0.47228 | 106.86 | 0.47476 | 0.46822 | 0.46320 | 0.53736 |
| Runs | 0.54392 | 0.51232 | 106.17 | 0.53926 | 0.55004 | 0.51784 | 0.54467 |
| Serial 1 | 0.53571 | 0.53584 | 99.98 | 0.53300 | 0.51054 | 0.54146 | 0.56533 |
| Serial 2 | 0.51635 | 0.49246 | 104.85 | 0.49903 | 0.52310 | 0.47274 | 0.51659 |

Table 2.4: The table shows average P-values (over 20 runs) for quasigroup encryption as compared to AES256 encryption system when the same encryption key is used for both cryptosystems with Cipher-Block-Chaining (CBC). Here data sets were of a short variety, constructed from a sequence of 288 bytes.

One should noted that the variance of P-values between different test results can be misleading. For this reason, the NIST-STS package provides a Success/ Fail determination. Thus, a second evaluation of the AES and QGBC cryptosystems (both in CBC mode) was also run. Here, source data sets of 295KB are encrypted and then assessed by the STS.

One thousand (1000) encrypted data sets were produced from files consisting of all binary zeros, all binary 0xFF's, all ASCII letter E's, and the Project Gutenberg imprint of Beowulf [1]. Each of the 1000 runs used a unique 256 bit key and initialization vector (IV), for a total of 1000 keys and IVs. With each of the key/IV pairs, the four files were encrypted with AES-CBC and QGBC-CBC, for a total of 8000 files.

The NIST-STS documentation tells us that when we evaluate the results we must look at the proportion successful tests. The authors is suggested that the confidence interval for our test should be defined by:

$$\hat{p} \pm 3\sqrt{\frac{\hat{p}(1-\hat{p})}{m}}$$

where $\hat{p} = 1 - \alpha$ and $m$ is the magnitude of the sample [7]. We generated $m =$

1000 tests, and with the recommended $\alpha = 0.01$, our confidence interval is $.99 \pm 3\sqrt{\frac{.\hat{9}9(.01)}{1000}} = .99 \pm 0.0094392$ or in other words the proportion should lie above 0.9805670.

Table 2.5 compares the success rates for these assessments. Success rates of AES and QGBC are comparable, both scoring in the 98 percentile or better, indicating successful evaluation of both systems.

| Test | AES | | | | QGBC | | | |
|---|---|---|---|---|---|---|---|---|
| | 0x00 | E | 0xFF | Beowulf | 0x00 | E | 0xFF | Beowulf |
| AE | 988 | 989 | 986 | 985 | 986 | 995 | 988 | 992 |
| BF | 992 | 990 | 994 | 991 | 991 | 991 | 986 | 991 |
| CSF | 990 | 993 | 990 | 994 | 988 | 992 | 996 | 992 |
| CSR | 994 | 989 | 991 | 994 | 986 | 994 | 994 | 994 |
| FFT | 990 | 988 | 989 | 986 | 984 | 981 | 990 | 980 |
| FREQ | 992 | 992 | 989 | 994 | 991 | 992 | 996 | 992 |
| LR | 991 | 987 | 991 | 989 | 990 | 988 | 987 | 991 |
| Rank | 989 | 989 | 996 | 989 | 994 | 995 | 982 | 995 |
| Runs | 994 | 988 | 993 | 991 | 987 | 993 | 989 | 993 |
| Ser1 | 990 | 992 | 995 | 995 | 991 | 990 | 989 | 994 |
| Ser2 | 986 | 993 | 990 | 987 | 984 | 993 | 991 | 988 |

Table 2.5: Successes per 1000 encryption tests. 295 KB of 0x00, 'E', 0xFF, and the text of Beowulf[1] were encrypted with 1000 different keys via the Quasigroup Block Cipher and AES, both in CBC mode, to demonstrate the ability to produce randomized data sets for long input data sequences.

### 2.2.2 Waveform Analysis

Another means of evaluation is to visually inspect the waveform of audio data. Here one compares the initial audio wave form to that of an encrypted waveform. The source [18] and the encrypted audio waveforms are plotted in Figures 2.2 and 2.3 respectively.

Both plots demonstrate the waveforms using the same bit-rate per unit on the horizontal axis. As we can see the quasigroup encryption system is very good at distributing the amplitude of the audio signal over the entire time domain.

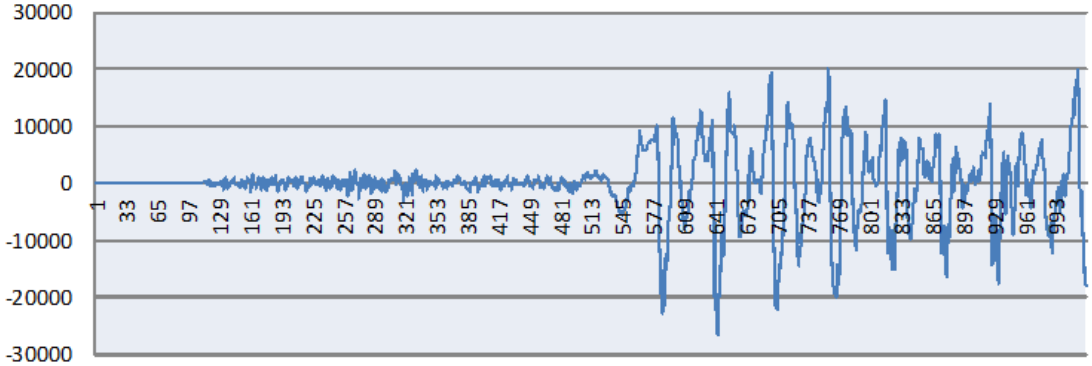



Figure 2.2: Plot of original input audio waveform

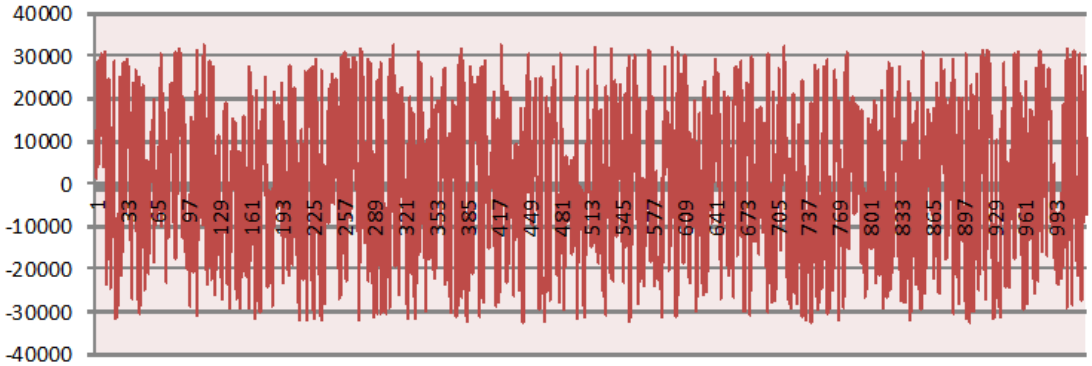



Figure 2.3: Plot of encrypted output audio waveform

# Chapter 3

# Storage Optimization:Low-Overhead Quasigroup representation

Recalling that the number of quasigroups of a certain order is given by $T_N(N-1)!N!$, we must store the entire quasigroup in memory, either as a set of $N^2$ ordered triples, or as a $N \times N$ matrix. Neither of these is very attractive, from a transmission standpoint, nor from a memory consumption standpoint if we were to implement the algorithm in hardware. In this chapter we explore the Low-Overhead Quasigroup substitution cipher and its savings.

## 3.1 Low-Overhead Quasigroup (LOQG)

First, let us consider some group $\mathbb{G}$, containing $N = |G|$ elements. To abstractly identify members, we will assign each an ordinal from $(0...N-1)$. Next consider the mathematical operation $c \equiv a + b \mod N$. In this operation we see that the following axioms are met: closure, associativity, identity and invertibility. Interestingly, this group also defines exclusive-or, when $n = 2$. The number of combinations in this group $(a, b)$ is $N^2$ producing $N$ results, a polyalphabetic substitution system.

Next, let's consider the following quasigroup definition which makes use of $H$ an randomly ordered tuple, such that each $H_i$ is distinct as in (3.1).

$$H = (H_0, H_1, ..., H_{n-1})$$

$$\begin{aligned} &\forall\ i, j \in \mathbb{G}, \qquad \forall\ H_i \in \mathbb{G}, \\ &\quad i = H_i \nLeftrightarrow (i+1) = H_{i+1}, \\ &\quad i \neq j \Leftrightarrow H_i \neq H_j : \\ &\qquad c \equiv H_i + H_j \mod N \end{aligned}$$

(3.1) Low Overhead Quasigroup defined

Because $H$ is randomly ordered, this quasigroup still remains closed and invertible, but has lost identity and associativity. The total number of permutations for this model is equal to the total possible permutations of $H$, $|H|! = N!$.

Now, let us expand this model further by considering $I$ also as an ordered tuple, identical to $H$ in all characteristics save it is shuffled in another manner. Thus

$$c \equiv H_i + I_j \mod N$$

remains closed and invertible, but the total number of permutations has increased by a factor of the permutations of $I$, $|I|! = N!$, giving the total number of permutations as $(N!)^2$.

## 3.2 Defense of the LO-QG

A quasigroup matrix of order $N$ requires the storage of a matrix of size $N \times N$. If we consider each element to be one byte in size ($N$=256) then the matrix required is of size 256x256, resulting in a storage requirement of 64 KB or $N^2$ elements.

In order to reduce the amount of storage, we take the advantage of the fact that if we set $v_{ij} = x_i + y_j \mod N$, then a matrix preserves the quasigroup structure; where $x_i$ and $y_j$ are row and column indices, respectively, and $v_{ij}$ is the value in the cell denoted by row $y_i$ and column $x_j$. Now, one could shuffle the columns and rows using Fisher-Yates[16] shuffling algorithm to generate a random quasigroup. In essence, if we were to use the initial identity $v_{ij} = x_i + y_j \mod N$ and only

store the shuffled states of the indices of rows and columns then we can reduce the storage requirement to $2N$ from $N^2$, which is a savings of $O(N^2)$.

This comes at the cost that total number of quasigroups that can be created by shuffling of rows and columns is $(N!)^2$ (which is less than $N!(N-1)!T_N$ as $T_N > N, \forall\ N > 4$). However, for all practical purposes for our implementation this gives $(256!)^2$ possibilities for the quasigroup, which is very large and still provides practical security.

Table 3.1 is the initial starting matrix given by the identity $v_{ij} = x_i + y_j \mod n$. Table 3.2 shows a randomly shuffled state of the quasigroup matrix in table 3.1. The top row and the left most column are the row and column indices of the matrix. Table 3.2 shows the shuffled state of the indices from table 3.1. Our storage savings arise from the fact that we can store only the initial identity equation and the $2n$ shuffled indices for the entire quasigroup.

| $\cdot$ | 0 | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|---|
| 0 | 0 | 1 | 2 | 3 | 4 | 5 |
| 1 | 1 | 2 | 3 | 4 | 5 | 0 |
| 2 | 2 | 3 | 4 | 5 | 0 | 1 |
| 3 | 3 | 4 | 5 | 0 | 1 | 2 |
| 4 | 4 | 5 | 0 | 1 | 2 | 3 |
| 5 | 5 | 0 | 1 | 2 | 3 | 4 |

Table 3.1: A un-shuffled quasigroup corresponding to $v_{ij} \equiv x_i + y_j \mod n$

| · | 2 | 0 | 5 | 4 | 3 | 1 |
|---|---|---|---|---|---|---|
| 4 | 0 | 4 | 3 | 2 | 1 | 5 |
| 1 | 3 | 1 | 0 | 5 | 4 | 2 |
| 3 | 5 | 3 | 2 | 1 | 0 | 4 |
| 5 | 1 | 5 | 4 | 3 | 2 | 0 |
| 0 | 2 | 0 | 5 | 4 | 3 | 1 |
| 2 | 4 | 2 | 1 | 0 | 5 | 3 |

Table 3.2: A shuffled quasigroup resulting from $x_i$ and $y_j$ having been shuffled. Note that while the values within the Quasigroup still conform to the $v_{ij} \equiv x_i + y_j \mod n$, but have lost the regularity of the un-shuffled reduced Quasigroup.

# Chapter 4

# Quasigroup Pseudo Random Number Generator

Pseudo random number generators (PRNGs) are essential to almost all digital systems. Random numbers are useful in games of chance: shuffling cards, altering the behavior of video game "enemies", etc. as well as for secure communications. PRNGs are deployed for creating digital signatures [19] [20], eliminating network congestion [21] [22], securing RFID communication [23] and facilitating cryptographic measures for key generation and even implementing stream ciphers. Unlike a true random number generator, PRNGs are defined by an algorithm.

The US Government, through the National Institute of Standards and Technology (NIST), is tasked with researching and recommending random number generators for use in governmental operations. The most recent recommendation is based on SHA-1, a hashing feedback algorithm [5]. However, the security of SHA-1 has been broken (theoretically), and NIST has recommended the move to another platform [2]. Although not implemented in major cryptographic suites, for the purpose of random number generation, NIST has identified SHA-2 and more recently SHA-3 successors to SHA-1 [5][24]. Similarly, ARC4 is used in many mobile devices, and it to has proven to be insecure [3].

Modern operating systems and development platforms offer PRNG algorithms as an included service. A survey reveals that Java and OpenSSL both implement

the SHA-1 PRNG [25] [26] [27]; Microsoft.NET used SHA-1 prior to Windows 6.0 (Vista), but switched to an AES variant afterwards [28]; and Apple's MacOS and iOS devices rely on ARC4Random [29]. Software vendors tend to default to US government recommendations as a baseline for cryptographic tools.

In this chapter we review the use of the QGBC as a pseudo random number generator (PRNG). Additional we demonstrate an updated version of the QGBC (see § 4.1), which prevents attackers from playing the QGBC algorithm in reverse if the quasigroup is known. This also moves the QGBC algorithm further from keeping the quasigroup secred. Further, as we consider the QGBC as a PRNG, we will focus on the Low Overheade QGBC (LO-QGBC).

## 4.1 Updated Quasigroup Block Cipher

Central to the proposed PRNG is the Quasigroup Block Cipher, which we have redefined as follows:

$$
\begin{aligned}
&C_1 := K_i \cdot (K_i \oplus M_1) \\
&\forall\ j \in \{2, 3, ..., 16\}, \\
&\qquad C_j := C_{j-1} \cdot (M_{j-1} \oplus M_j)
\end{aligned}
$$

(4.1) Improved QGBC

Let $C$ represent 128 bits cipher text ($C_n$ a single byte in $C$), $M$ 128 bits of plain text ($M_n$ a single byte from $M$), $K_i$ a key byte, $\cdot$ is the quasigroup operation and $\oplus$ is a bit-wise exclusive-or (XOR). Notice that we have added the additional XOR operation in an attempt to prevent an attacker from playing the algorithm in reverse if the quasigroup is known.

This algorithm is an improvement of the G&M stream cipher[4], as it allows us to publish the quasigroup. For instance, if the quasigroup were publicly know for the G&M stream cipher, an attacker could take any $C_j$ and $C_{j-1}$ and compute $M_j$, effectively recovering the plain text by replaying the algorithm in reverse. In the improved algorithm each $C_j$ is dependent not just on $C_{j-1}$ and $M_j$ but $M_{j-1}$

as well. This prevents reverse auto-decryption, requiring the knowledge of a fully decrypted word to decrypt the following word. As we are using the LO-QGBC the remainder of the algorithm is shown in algorithm 4 but uses the LO-QG cipher substitution from algorithm 5.

**Data**: H,I – LO-QG randomly ordered tuples
**input** : s – a single word from the Key
**input** : M – A block of plain text
**output**: C – A block of cipher text
$C_0 = H_s + I_{M_0} \mod N$;
**for** $i = 1to|M| - 1$ **do**
  $C_i = H_{C_{i-1}} + I_{M_i} \mod N$;
**end**

**Algorithm 5:** The LO-QG cipher

Figure 4.1, a S-P block diagram, depicts the one step in multi-byte key application of the block cipher. In this diagram, the S blocks represent the quasigroup $\cdot$ operation and the P block represents the bit rotation. An additional block, the $\oplus$ block, represents the exclusive-or substitution block.

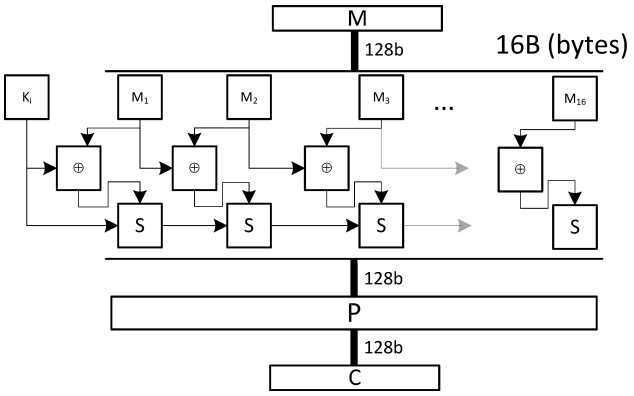



Figure 4.1: Block Diagram: Multi-byte Key Quasigroup Block Cipher w/o cipher block chaining

Provided to demonstrate the QGBC graphically, fig 4.1 depicts how the 128 bit plain-text message block $M$ is subdivided into 16 equal size 8-bit bytes.In the diagram we see that each of message byte is combined with the previous message byte ($\oplus$ *block*), and then polyalphabetically substituted (S-block) with either the

previous cipher text $C_{j-1}$ or the key $K_i$. Then in the P-block, the entire $C_{i'}$ sequence is rotated to become $C_i$, finally becoming $C$ after the all bytes in $K$. The diagram does not depict the looping behavior post P-block. Instead, the cipher text $C$ would be passed into an identical S-P network, along with the next $K_i$.

## 4.2 Feedback Generator

The quasigroup block cipher allows us to generate 128 random bits at a time. To generate more, we must construct a mechanism which is self-sustaining and statistically random. For this case, we deploy a feedback generator. The following steps occur in such a mechanism:

Select a random initialization vector $(V)$;
Select a random key $K$;
Select a plain text $(M)$;
Calculate $O_1 = QGBC(V \oplus M, K)$ and report as first 128 bits;
**for** $\forall\ i > 0, i \in \mathbb{Z}$ **do**
  $O_x i = QG(O_{x-1} \oplus M, K)$;
  Report as $i^{th}$ 128 bits;
**end**

**Algorithm 6:** QGBC PRNG Feedback Generator

Here $QGBC(M, K)$ is the multi-byte quasigroup block cipher. This algorithm is depicted in figure 4.2. This feedback generator takes a random seed and initialization vector, and is self-sustaining from this point on.

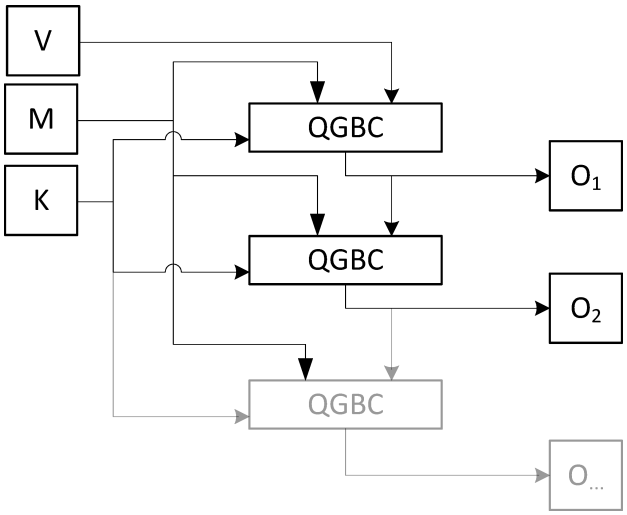



Figure 4.2: Block Diagram: Feedback Generator for self sustaining PRNG

Figure 4.2 demonstrates the process of feeding back the previously generated random sequence $O_{x-1}$, while cipher-block-chaining it to the input $M$. To limit the amount of data required to seed our QGBC-PRNG system, we can choose $M = (K_0, K_1, ..., K_{15})$ and $V = (K_{16}, K_{17}, ..., K_{31})$.

Nowadays, 256 bits of random data are simple to come by. We can use SHA-256 [5] as a source method to combine information such as the current time in seconds, and other data from the source system, like MAC addresses, memory consumption, TCP/IP address, etc. For the best hash possible, one should acquire at least 256 bits of source data.

## 4.3 Processing Time

First let us consider the cost in operations for the LO-QGBC, in this case modeling a modern CPU with a random-access memory. We will define the following costs:

$M$ – Cost of memory retrieval
$O$ – Cost of single byte $\cdot$ and $\oplus$
$R$ – Cost of 128-bit rotation $(p(C, x))$
$O_{\oplus}$ – Cost of 128-bit $\oplus$

Modern processors can calculate $m = a + b \mod 256$ in a single step, for this reason we consider $\cdot$ and $\oplus$ to have equivalent cost. Thus recurrence for the

quasigroup block cipher is given as:

$$T(q(K,M)) = 2|K||M|(M+O) + R$$

Substituting $|K| = 32, |M| = 16$ we see:

$$T(q(K,M)) = 2^{10}(M+O) + R$$

Now let us consider the feedback generator that incorporates cipher block chaining, thus the recurrence for each output block $C$ becomes:

$$\begin{aligned} T(q(K,M)) &= 2|K||M|(M+O) + R + O_{\oplus} \\ &= 2^{10}(M+O) + R + O_{\oplus} \end{aligned}$$

As one may see, the time to produce each $C$ is fixed, giving $T(C = q(K,M)) = O(c)$. Further the recurrence to produce $m$ bits is:

$$\frac{T(C)m}{8|C|} = O(\frac{cm}{2^7}) = O(c'm) = O(m)$$

## 4.4 Evaluation

Common practice has the researcher compare the output of one PRNG to other well established PRNG systems. Again, we will use the NIST-STS this purpose. Each of the STS tests focuses on a different aspect of randomness through out the input sequence. Weighing any one test over the others could be a mistake, instead, some balance between the evaluations should be sought. The STS team points to the reason for this: First, they identify a Type I error (denoted as $\alpha$, also known as the level of significance for a given test. The team chose P-values of 0.01 as significant for cryptographic work. Second, are Type II errors which they denote as $\beta$. $\beta$ errors occur when a sequence is falsely identified as random, and there is no fixed value for this. One approach to reducing $\beta$ errors is to elevate the significance of $\alpha$, another is to increase the breadth of testing [30] [17].

### 4.4.1 Evaluation Process

Preparation for evaluation QGBC-PRNG involved selecting a set of well established PRNG systems. For comparison, we chose `arc4random` from Free BSD/MacOS X [29], Microsoft.Net's `RandomNumberGenerator` [31], OpenSSL `RAND` [27], and Java's `SecureRandom` [26]. These PRNG systems were chosen because each is well accepted by industry, including all major modern OS platforms (MacOS X, Linux, Windows, and Java). Further, `RAND` and `SecureRandom` are both implemented to conform with FIPS 180-4 [5], in which the NIST has specified the minimum requirements, for secure random number generation in US government cryptosystems. It should be noted that the system function, `RandomNumberGenerator`, was called on a system running *Windows 7*, and therefore utilized an AES based PRNG, instead of the SHA-1 system utilized in Microsoft systems prior to *Microsoft Vista* [28].

For each of the five PRNGs, we generated one-thousand (1000) sequences of random data, each containing 512 kilobytes ($2^{22}$ bits). For each of these five-thousand files, we generated unique random seeds, so that each run would produce a unique sequence of data.

After generating the random outputs, each file was passed through the NIST-STS system. Here again, we used the same settings that were used when testing the QGBC as cipher (see tbl. 2.1).

### 4.4.2 Evaluation Results

We have captured the success rate of each of the five PRNG systems tested in Table 4.1. QGBC-PRNG performs in the $99^{th}$ percentile for all of the tests evaluated. Just as the STS performs statistical tests, the results should be considered statistically as well [17].

Any system testing in this range 980-1000 is considered "acceptably random", based on the confidence interval identified by the NIST team[17]. Review of the test results showed that the commercially available PRNG systems pass the NIST-STS test suite as well, which should be expected, as these systems have been

| | QGBC-PRNG | ARC4 | OSSL | Java | MS.Net |
|---|---|---|---|---|---|
| AE | 990 | 986 | 987 | 990 | 991 |
| BF | 990 | 993 | 988 | 993 | 997 |
| CSF | 985 | 990 | 985 | 988 | 989 |
| CSR | 987 | 993 | 987 | 986 | 989 |
| FFT | 993 | 985 | 983 | 988 | 987 |
| FREQ | 986 | 989 | 985 | 990 | 990 |
| LR | 995 | 987 | 995 | 989 | 994 |
| Rank | 994 | 988 | 992 | 992 | 989 |
| Runs | 986 | 986 | 995 | 991 | 990 |
| Ser1 | 990 | 984 | 988 | 988 | 989 |
| Ser2 | 988 | 984 | 988 | 985 | 993 |

Table 4.1: NIST-STS Test Success Rates for 1000 Samples

vetted through rigorous use. Therefore, it should be noted that even though the RC4 and SHA-1 based systems have been broken, they are still capable of passing the statistical tests. Additional inspection is required to demonstrate strength (see Security below). Also, cryptoanalysis of the QGBC/QGBC-PRNG should be performed, and appears in chapter 5.

## 4.5 Autocorrelation

Autocorrelation proves to be another successful examination of the randomness of a sequence. Like the *Cumulative Sums* test from the NIST-STS suite, autocorrelation works best when we evaluate adjusted bits (i.e. 0 transforms to -1). While performing the evaluation, we may observe any adjusted sequence $S$. Thus

autocorrelation may be defined by the following:

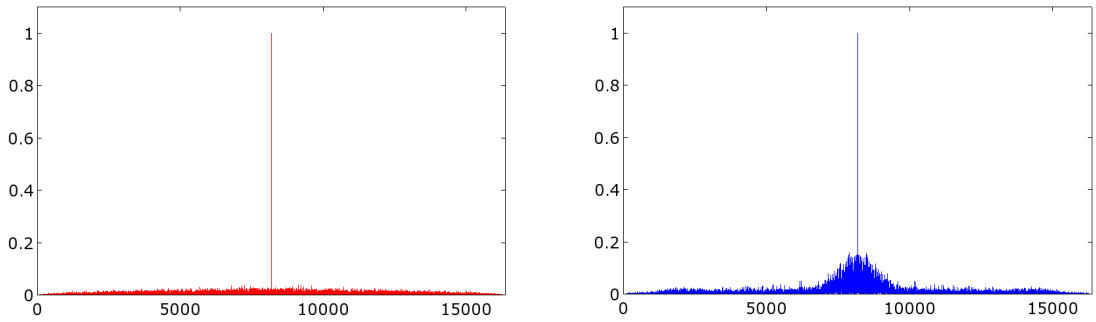
$$n = |S|$$
$$1 \leq i \leq n : r_i = \sum_{j=1}^{i} S_j + S_{n-j}$$
$$n+1 \leq i \leq 2n : r_i = \sum_{j=1}^{n-i} S_j + S_{j+(i-n)}$$

Autocorrelation procedure produces $2n$ data points. To examine randomness, we should consider the ratio between the number of data points $n = |S|$ and each correlation $r_i$. We examine the autocorrelation results for $2^{13}$ bits from QGBC-PRNG output (fig. 4.3a), PCM data from sample audio file (fig. 4.3b) [18], a cross correlation between two QGBC-PRNG outputs (fig. 4.3d), and an OTP encryption of the sample audio using the QGBC-PRNG output as the pad (fig. 4.3c).

Examination of an autocorrelation plot should show a single spike at $n$, where the sequence is directly correlated with itself. Peaks other than the $n$-spike indicate higher degrees of correlation showing repeating patterns, and is expected.

Our results show that the QGBC-PRNG output (fig. 4.3a) has a plot of a random data set. Meanwhile, sample audio PCM data does not (fig. 4.3b), and shows higher degrees of correlation based on locality. However, once we apply an OTP encryption to the sample audio, with the QGBC-PRNG output, we see that this sequence now matches the expected pattern for random data. This would suggest that we have inserted entropy into the audio sequence, providing for an acceptable encryption.

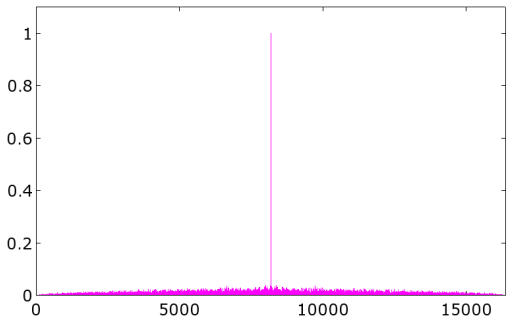
Finally, we should consider cross-correlation between outputs from the QGBC-PRNG with different seeding. Whenever an OTP encryption is applied, it is essential to use very different random sequences for each encryption application. This is an issue particular to OTP cryptosystems, as comparisons between runs with identical pads, will render both the pad and plain text. Figure 4.3d shows



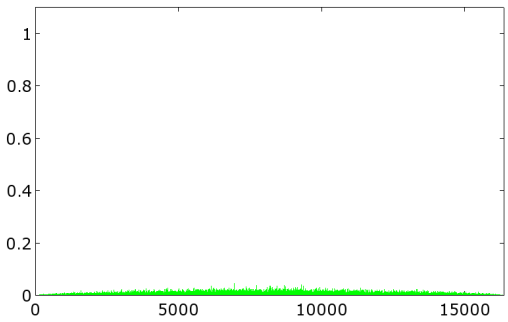

(a) QGBC-PRNG Output




(b) Sample Audio PCM Data




(c) Encrypted Audio




(d) Cross-Correlation

Figure 4.3: Autocorrelation Plots

that the cross-correlation between two QGBC-PRNG output has very low correlation, and shows no $n$-spike, which is also to be expected, given the two sets do not correlate.

## 4.6 Security of QGBC-PRNG

Let us consider 8-bit words ($n = 256$), and a 32 word (256 bit) key ($k = |K| = 32$) as input to a quasigroup block cipher. The probability of correctly selecting $M_j$ given $C_{j-1}$ and $C_j$ is greater than or equal to $1 : 2^{16}$. Also, the probability of correctly selecting $M_1, M_2, ...M_j$ is given $C_1, C_2, ..., C_j$ is greater than or equal to $1 : 2^{16(j-1)}$. Further, using bit rotation (found in the QGBC cipher) and multiple $K_i$ applications, guarantees that the probability of correctly selecting any one byte in the sequence to be $1 : 2^{16(j-1)}$. With a block size of 16 bytes, $j = 16$, thus the probability is greater $1 : 2^{240}$ for correctly selecting the sequence.

When cipher block chaining is used (as in the case of the feedback generator),

the probability of correctly selecting a sequence of $l$ blocks is greater than $1 : 2^{16(j-1)l}$, thus an output sequence of 32 bytes has a probability of $1 : 2^{480}$, a sequence of 64 bytes, $1 : 2^{960}$, and so forth. This leads us to conclude that an attacer would rather attempt to attack the input seed which has 256 bits. We can conclude that the QGBC-PRNG maximal strength is $2^{256}$, making it key efficient.

# Chapter 5

# Cryptanalysis of the QGBC

Vetting of a cryptosystem requires review through multiple forms of analysis. In this chapter, we attack the QGBC through Linear Crypt Analysis § 5.1 and then through Algebraic Analysis § 5.3. Through these means we identify several problems with the QGBC system and then propose solutions to eliminate these.

## 5.1 Linear Cryptanalysis

Linear cryptanalysis examines relationships between plain text and cipher text, with the goal of determining affine approximations of the cipher [32]. First identified as a method to assess the FEAL cipher by Matsui and Yamagishi [33] and later applied to the DES cipher [6]; linear cryptanalysis uses the probability of linear combinations of message and cipher text to attack individual rounds and by extension the cipher as a whole.

When we apply a linear cryptanalysis, we compute the probability of every combination of input $M$ and output $C$ bits, for every possible input $M$ and key $K$. For simplicity, let us consider $n = |M| = |C|$ and $|K| = k$, which represent the size in bits of $M$, $C$, and $K$. The number of linear combinations from $M$ and $C$ is then $2^{2n} - 2^{n+1}$, while the number of inputs $M$ and $K$ is $2^{nk}$. Overall we see the complexity of exhaustively performing a linear analysis would be $O(2^{kn^2})$. Based on the exponential nature of the tabulation, the analyst must limit his/her scope to some portion of the cipher and then develop a strategy to attack the cipher as

a whole.

As a straw man let us consider an order 4 substitution matrix such as the one shown in :

| | 0 | 1 | 2 | 3 |
|---|---|---|---|---|
| 0 | 2 | 3 | 0 | 1 |
| 1 | 0 | 1 | 2 | 3 |
| 2 | 1 | 2 | 0 | 3 |
| 3 | 0 | 1 | 3 | 2 |

Table 5.1: Substitution matrix with odd-bit bias

If we consider $c_0 \oplus\ m_0 = 0$ (least order bits of $C$ and $M$) we see it has $p = \frac{3}{4}$; clearly demonstrating a bias on the odd bit. With this knowledge, the analyst can pass along probability to subsequent passes, tracking the actions on the bit and formulate an overall approximation[32].

## 5.2 Linear Cryptanalysis of QGBC

When we apply linear cryptanalysis to the QGBC cipher, we will use it to help identify minima for the cipher's order. To this end, we construct linear combinations of plain text bits and cipher text bits. Although the past implementations of QGBC used 128 bit blocks with 256 bit keys, and quasigroups of order 256, the number of linear combinations is beyond the grasp of our available computational power, as this would require reviewing $(2^{256} - 2^{128})$ linear combinations for each of the $2^{384}$ results, or $(2^{640} - 2^{512})$ comparisons! For this reason we we will review straw-man versions of the QGBC to review the effect of alterations to $S()$ and $P()$, by varying the quasigroup choice and rotational distance. We've used a shorthand to describe the quasigroup being evaluated.

In the following sections we will explore seven experiments. The $1^{st}$ experiment (exp.) will be a quasigroup order 2, the $2^{nd}$ a quasigroup defined by addition modulo 4, the $3^{rd}$ and $4^{th}$ – quasigroups order 4 that have been randomized and shifted by two bits, the $5^{th}$ uses addition modulo 4 but rotations that are not

two bits in distance, the $6^{th}$ uses the quasigroup from experiment 4 but shifts by distances other than two bits and finally in the $7^{th}$ experiment, we examine a single round of a quasigroup order 16.

### 5.2.1 Exp. 1: QG order 2

We begin analysis with a divide and conquer approach [32]. Hence, first consider a trivial QGBC cipher, with the following conditions:

- $N = 2$
- $|M| = 4$
- Single round of the substitution function $S()$

| | 0 | 1 |
|---|---|---|
| 0 | 0 | 1 |
| 1 | 1 | 0 |

Table 5.2: Quasigroup $N = 2$, similar to exclusive-or

Since the quasigroup is order is $N = 2$, the size of each word is a single bit ($\frac{|M|}{|log_2 N|}$ =1). We are able to perform an exhaustive examination, capturing all $K$, $M$ and $C$. Next we count the the linear combinations of plaintext and ciphertext bits which "sum" to zero. We see that certain combinations have probability $p = 1$, while the remainder have probability $p = \frac{1}{2}$. The combinations with high probability are shown in (5.1).

$$\begin{aligned} &\forall i, j \in (0, 1, 2, 3), i \neq j; \\ &\quad c_i \oplus c_j \oplus m_i \oplus m_j = 0; p = 1 \\ &c_0 \oplus c_1 \oplus c_2 \oplus c_3 \oplus m_0 \oplus m_1 \oplus m_2 \oplus m_3 = 0; p = 1 \end{aligned}$$

(5.1) High probability linear combinations for $N = 2$ QGBC

The results show that if we consider a pair of any two input bits, with the corresponding output bits, these form a balanced linear combination, in all cases.

From this we interpret this to be a trivially weak cryptosystem [32]. We will investigate the cause of this later in the thesis, via algebraic analysis.

### 5.2.2 Exp. 2: Addition Modulo 4, shift 2

Next, we will examine a QGBC system of order 4. Here each word is now represented with two bits. Consequently $+(mod 4)$ (denotes addition mod 4) is not identical to bitwise $\oplus$ as was the case with $N = 2$ QGBC.

- $N = 4, |K| = 8, |M| = 8, w = 4$
- $P(C, 2)$ performs a left shift by a whole word (2 bits)
- 4 rounds of the $S()$ and $P()$ functions
- Quasigroup is defined by $+(mod\ N)$ seen in table 5.3

Through an exhaustive comparison of all $K$, $M$ and $C$ for this system, tabulations show that an "odd bit" problem appears. In every message word, there is an independent, 1:1 correlation ($p = 1$) of low order bits in the corresponding ciphertext word. The "odd bit" issue occurs because there is always an even number of additions applied to each word, causing the low-order bit to never fluctuate.

| | 0 | 1 | 2 | 3 |
|---|---|---|---|---|
| 0 | 0 | 1 | 2 | 3 |
| 1 | 1 | 2 | 3 | 0 |
| 2 | 2 | 3 | 0 | 1 |
| 3 | 3 | 0 | 1 | 2 |

Table 5.3: Quasigroup order $N = 4$, defined by $+(mod\ N)$

### 5.2.3 Exp. 3: Randomized QG 'A', shift 2

For this sampling, we again perform an exhaustive combination of all $K$, $M$, and $C$, but this time, the quasigroup is not defined by $+(mod\ N)$ but by a randomized Latin square, which does not reduce to $+(mod\ N)$ [34][35], and is defined by the Latin square specified in table 5.4.

- $N = 4, |K| = 8, |M| = 8, w = 4$
- $P(C, 2)$ performs a left shift by a whole word (2 bits)
- 4 rounds of the $S()$ and $P()$ functions

| | 0 | 1 | 2 | 3 |
|---|---|---|---|---|
| 0 | 1 | 2 | 0 | 3 |
| 1 | 2 | 1 | 3 | 0 |
| 2 | 0 | 3 | 1 | 2 |
| 3 | 3 | 0 | 2 | 1 |

Table 5.4: Quasigroup with poor linear analysis performance

We see that this configuration also fails linear analysis based on the high probability linear combinations, for brevity, three of which shown in (5.2).

$$m_0 \oplus m_1 \oplus m_3 \oplus m_5 \oplus m_7 \oplus c_2 = 0$$
$$m_1 \oplus m_2 \oplus m_4 \oplus m_6 \oplus c_2 \oplus c_3 = 0$$
$$m_1 \oplus m_5 \oplus m_6 \oplus c_1 \oplus c_5 \oplus c_6 = 0$$

(5.2) $p = 1$ Combinations for $N = 4, QG \rightarrow +(mod\ N) randomized$

### 5.2.4 Exp. 4: Randomized QG 'B', shift 2

Like Example A, Example B uses a randomized quasigroup that does not reduce to $+(mod\ n)$, but is defined by table 5.5.

- $N = 4, |K| = 8, |M| = 8, w = 4$
- $P(C, 2)$ performs a left shift by a whole word (2 bits)
- 4 rounds of the $S()$ and $P()$ functions

| | 0 | 1 | 2 | 3 |
|---|---|---|---|---|
| 0 | 1 | 2 | 0 | 3 |
| 1 | 2 | 3 | 1 | 0 |
| 2 | 0 | 1 | 3 | 2 |
| 3 | 3 | 0 | 2 | 1 |

Table 5.5: Quasigroup with improved linear analysis performance

This configuration resulted in 24,954 linear combinations with $p = \frac{1}{2}$, a maximum $p = \frac{5}{8}$ and a minimum $p = \frac{3}{8}$ and average $p = 0.5 \pm 0.0058$.

### 5.2.5 Exp. 5: Addition modulo 4, shift $\neq$ 2

In this configuration, we once again use a quasigroup based equivalent to $+(mod\ N)$, but to counter the "odd-bit" problem, we perform the sequence $S(), P(C, 1), S(), P(C, 3), S(), P(C, 3), S()$.

- $N = 4, k = 8, |M| = 8, w = 4$
- $P(C, r), r = 1, 3, 3$ performs a left shift by 1 then 3 then 3 again
- 4 rounds of the $S()$ and $P()$ functions

This solution completely eradicated the "odd-bit" problem as well as removing all $p = 1/p = 0$ linear combinations. Instead we find that 55,511 of $2^{16}$ linear combinations have $p = \frac{1}{2}$. Of the remainder the average probability is $p = 0.5 \pm 0.0138$ with a maximum of $p = 0.0688$ and 0.344.

### 5.2.6 Exp. 6: QG 'B', shift $\neq$ 2

For this experiment we revisited the Example B Latin square, but this time used the 1-3-3 rotation pattern used in the previous example. The following are the configuration data:

- $N = 4, k = 8, |M| = 8, w = 4$
- $P(C, r), r = 1, 3, 3$ performs a left shift by 1 then 3 then 3 again

- 4 rounds of the $S()$ and $P()$ functions
- Quasigroup defined in tbl. 5.5

In this case we discovered 50,798 linear combinations had $p = \frac{1}{2}$, the minimum probability $p = \frac{3}{8}$ and a maximum probability of $p = 0.563$ and an average $p = 0.5 \pm 0.0101$. From these results we conclude, in conjunction with identification of the "odd bit" problem, we can see that $P()$ should not rotate on the word boundary.

### 5.2.7 Exp 7. QG Addition modulo 16

In the final exhaustive linear analysis experiment, we evaluate an $+(mod\ N), N = 16$ quasigroup. Because of the limitation of processing capabilities, we use a block of 8 bits, but now with only two words.

- $N = 16, k = 8, |M| = 8, w = 2$
- Single round of $S()$

We see that a single linear combination $m_0 \oplus m_4 \oplus c_0 \oplus c_4 = 0$ has $p = 1$, i.e. suffers the "odd bit" problem. We can speculate that with $P()$ steps that rotate by less than a whole word will remove this issue. With this experiment concluding the linear analysis, let us now look to algebraic analysis of the QGBC.

## 5.3 Algebraic Cryptanalysis

Algebraic cryptanalysis allows us to examine the QGBC algorithm directly, identifying possible defects introduced by the mathematical interaction. We will examine the $N = 2$ quasigroup and also larger order quasigroups which have four words per block.

### 5.3.1 Algebraic Analysis QGBC $N = 2$

Algebraic analysis of the QGBC system can lead us to an understanding of the high probability of a linear combination found using exhaustive linear analysis.

Let us first consider QGBC order 2. Here S becomes a bitwise XOR or Not XOR, for simplicity we will consider:

$$v = w$$

$$\begin{aligned} &S(C, M, K, i) : \\ &\quad c_0 = k_i \oplus (m_1 \oplus k_{w-i-1}) \\ &\quad \forall 1 \leq j < w : c_j = c_j - 1 \oplus (m_{j-1} \oplus m_j) \end{aligned}$$

Previously, there was an assumption that $\cdot$ was neither associative nor distributive over $\oplus$. However, we know that $\oplus$ is associative. Thus if we expanded $S()$ we see:

$$\begin{aligned} k' &= k_0 \oplus k_1 \oplus \ldots \oplus k_{n-1} \\ \forall 0 \leq j < w &: c_j = k' \oplus m_j \end{aligned}$$

This would indicate that we have introduced a single bit of randomness in the system, hence the system is trivially weak.

### 5.3.2 Algebraic Analysis QGBC $N > 2$

As long as a QGBC system is of order greater than 2 and the quasigroup does not reduce to a bitwise XOR group, we can assume that $\cdot$ is not distributive over $\oplus$ . Thus, let us consider a worst case scenario, where there are 4 words in the key and 4 words in the block (this could represent four 64-bit words, for 256 bits in the key and block, or our four 2-bit words for an 8 bit block form before). To further review a worst-case, consider that each word in the plain text is the same and represented by the term $a$. With these conditions, the cipher collapses to the following:

Note: each additional apostrophe indicates an additional round of the cipher; since there are four words in the key, four rounds are applied. From this, we can infer some characteristics about our key. Specifically that the following hold true (see (5.4)), else an input of $a = 0$ would result in a trivially simple cipher.

$$
\begin{aligned}
&\forall\ 0 \leq j < 4, m_j = a : \\
&\quad c_j = k_0 \cdot (k_3 \oplus a) \\
&\quad c'_j = k_1 \cdot (k_2 \oplus P(k_0 \cdot (k_3 \oplus a))) \\
&\quad c''_j = k_2 \cdot (k_1 \oplus P(k_1 \cdot (k_2 \oplus P(k_0 \cdot (k_3 \oplus a))))) \\
&\quad c'''_j = k_3 \cdot (k_0 \oplus P(k_2 \cdot (k_1 \oplus \\
&\qquad\qquad P(k_1 \cdot (k_2 \oplus P(k_0 \cdot (k_3 \oplus a))))))
\end{aligned}
$$

(5.3) Expansion of the QGBC block cipher

$$
\begin{aligned}
&k_2 \neq P(k_0 \cdot k_3) \\
&k_1 \neq P(k_1 \cdot (k_2 \oplus P(k_0 \cdot k_3))) \\
&k_0 \neq P(k_2 \cdot (k_1 \oplus P(k_1 \cdot (k_2 \oplus P(k_0 \cdot k3)))
\end{aligned}
$$

(5.4) Inequalities to strengthen QGBC

Further observation shows an "identical word" problem that appears when the words of $M$ are identical to the words of $C$ are also identical. This would be an obvious attack on the cipher, and point to a plain text attack based on such a construction. Thus the ability to attack the cipher would remain on the strength of the problem described by $c'''_j$.

# Chapter 6

# Improving the QGBC

Now that we have identified issues such as the "identical word" problem, we should suggest alternatives that solve this. In section § 6.1, below, we do just this. Also we extend this improvement and suggest a high-performance variant of the QGBC suitable for FPGA implementation in § 6.1.1.

## 6.1 Improvement of the QGBC

If we make a change to the QGBC cipher we see that we can counter the "identical word" problem. Consider a construction such as show in (6.1). With this configuration we se a single pass improvement in shown in (6.2).

$$
\begin{aligned}
&S(C, M, K, i) : \\
&\quad C_0 = K_{(|K|-1)} \oplus (\overline{K_i} \cdot M_0) \\
&\quad \forall\ 1 \leq j < |M| : C_j = K_{(|K|-1-j \ mod\ |K|)} \oplus (\overline{C_{j-1}} \cdot M_j)
\end{aligned}
$$

(6.1) Improved QGBC Substitution Function

The first round ($C'_i$) shown in (6.2) shows that we have eliminated the "identical word" problem, and additionally, we have introduced more of the key into each pass of the cipher. Reviewing the expression for $C'_2$, we see that we have introduced $2^k$ bits of the key (or their inverse) into the calculation, which in subsequence rounds affects every bit in the block. In fact, by round 2 ($C''_0$) we have

$$
\begin{aligned}
C'_0 &= K_3 \oplus (\overline{K_0} \cdot A) \\
C'_1 &= K_2 \oplus ((K_3 \oplus (\overline{K_0} \cdot A)) \cdot A) \\
C'_2 &= K_1 \oplus ((K_2 \oplus ((K_3 \oplus (\overline{K_0} \cdot A)) \cdot A)) \cdot A) \\
C'_3 &= K_0 \oplus ((K_1 \oplus ((K_2 \oplus \\
&\qquad ((K_3 \oplus (\overline{K_0} \cdot A)) \cdot A)) \cdot A)) \cdot A) \\
C''_0 &= K_3 \oplus (\overline{K_1} \cdot C'_1) \\
&= K_3 \oplus (\overline{K_1} \cdot (K_2 \oplus ((K_3 \oplus (\overline{K_0} \cdot A)) \cdot A))) \\
&\ldots
\end{aligned}
$$

(6.2) Single pass of improved QGBC w/ full-word rotation distance

successfully integrated bits (or their inverse) from every word in the key into every word of the cipher text.

### 6.1.1 Application of the Improved QGBC

Although the improved QGBC substitution function 6.1 may be applied in general, a special case is particularly interesting, where the QGBC is implemented in Field Programable Gate Array (FPGA) hardware. While some FPGAs possess memory components [36], buffers large enough to hold 64 KB[37][15] typically require off-chip memory to keep the unit price under $5.00US[38]. The low-overhead QGBC (LO-QGBC) [39] provides an alternative, the number of clock cycles to implement the algorithm, as-is, is not competitive with AES in terms of clock-cycles per encrypted block [40][41].

Instead, consider the improved QGBC in a high-performance configuration (QGBC-HP), with the following parameters:

| | |
|---|---|
| **Quasigroup Order** | $N = 2^{64}$ |
| **Quasigroup Definition** | Addition Modulo $N$ |
| **Key Size** | 256 bits |
| **Block Size** | 256 bits |
| **Words per Block** | 4 |
| **Number of Rounds** | 4 |

$$
\begin{aligned}
&S(C, M, K, i) : \\
&\quad C_0 = K_{(|K|-1)} \oplus (\overline{K_i} + M_0) \\
&\quad \forall\ 1 \leq j < |M| : C_j = K_{(|K|-1-i \ mod \ |K|)} \oplus (\overline{C_{j-1}} + M_j)
\end{aligned}
$$

(6.3) QGBC-HP Substitution function

With this we can express $S()$ as in (6.3). The entire cipher is pictured via block diagram in fig. 6.1, where $P_{57}$ a 57 bit left rotation and $P_{83}$ indicates an 83 bit left rotation. The first $S$ block is expanded to show the internals, and each subsequent $S$ block is identical.

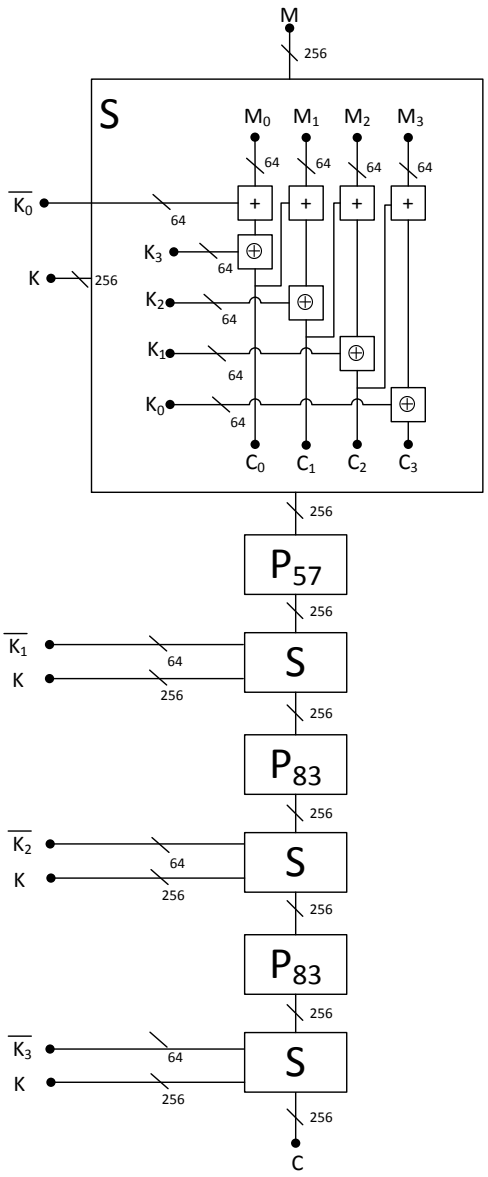



Figure 6.1: QGBC-HP Cipher Network

With this specification, an FPGA solution requires a total of 512 bits of memory divided into two register banks, K (Key, 256 Bits) and A (Accumulator, 256 bits, receives the initialization block(IV) as well as Message text), a 64-Bit Adder, 64-Bit XOR, $M$-Bit XOR (for use in loading the Accumulator, assuming the data bus is $M$ bits wide and less than 64 and is a divisor of 256), inversion and shift logic. Each round may be accomplished in 4 clock cycles (executing the XOR and Addition in a single clock) followed by a single clock to rotate bits. The core

processing for the QGBC-HP algorithm can be performed in 19 clocks, plus the overhead of read-in/read-out. The algorithm for this process is show in Algorithm 7.

**Data**: A : Accumulator Register 256 bits
**Data**: K : Key Register 256 bits

```
/* Load the Key                                                    */
K ← data bus ;                                  /* 256/M clocks */
/* Load the IV                                                     */
A ← data bus ;                                  /* 256/M clocks */
while Encrypting Data do
    /* Perform the Cipher Block Chaining Step while loading the
       message text                                                */
    A = A ⊕ ← data bus;
    /* 256/M clocks                                               */

    /* Execute 4 rounds of the block cipher                       */
    A0 = K3 ⊕ (¬K0 + A0) ;                          /* 1 clock */
    A1 = K2 ⊕ (A0 + A1) ;                           /* 1 clock */
    A2 = K1 ⊕ (A1 + A2) ;                           /* 1 clock */
    A3 = K0 ⊕ (A2 + A3) ;                           /* 1 clock */

    for i = 1 to 3 do
        if i = 1 then
            left rotate A by 57 ;                   /* 1 clock */
        else
            left rotate A by 83 ;                   /* 1 clock */
        end
        A0 = K3 ⊕ (¬Ki + A0) ;                      /* 1 clock */
        A1 = K2 ⊕ (A0 + A1) ;                       /* 1 clock */
        A2 = K1 ⊕ (A1 + A2) ;                       /* 1 clock */
        A3 = K0 ⊕ (A2 + A3) ;                       /* 1 clock */
    end

    data bus ← A ;                              /* 256/M clocks */
end
```

**Algorithm 7:** QGBC-HP FPGA Algorithm

### 6.1.2 Experimental Evaluation of QGBC-HP

As with previous versions of the QGBC, the high performance variant has been implemented in software and evaluated with the NIST-STS test suite[7]. As rec-

ommended we encrypted the first 50 KB of Beowulf[42]; creating 1000 samples, each with a randomly generated key and IV. All of the samples were passed through the analysis suite and PASS/FAIL results were tabulated (see tbl. 6.1). Based on the suggested conifdence interval, we see that the QGBC-HP algorithm passed all of the recommended statistical tests for randomness.

| **NIST-STS Test** | **Success Rate** | **Result** |
|---|---|---|
| Approximate Entropy | 980/1000 | PASS |
| Block Frequency | 991/1000 | PASS |
| Cumulative Sums-Forward | 994/1000 | PASS |
| Cumulative Sums-Reverse | 998/1000 | PASS |
| FFT | 991/1000 | PASS |
| Frequency | 994/1000 | PASS |
| Longest Run | 991/1000 | PASS |
| Rank | 991/1000 | PASS |
| Runs | 990/1000 | PASS |
| Serial 1 | 992/1000 | PASS |
| Serial 2 | 988/1000 | PASS |

Table 6.1: QGBC-HP NIST-STS Results

# Chapter 7

# Conclusion

Through this exploration of quasigroups in cryptography, we have identified a novel use of quasigroups, where we use polyalphabetic substitution to formulate a block cipher. To this end, we have demonstrated the QGBC as a means of enciphering plain text, as well as generating random data for use in an OTP stream cipher. We have evaluated our cryptosystem with industry standard tools, and performed algebraic and linear cryptanalysis of the system. Each time we have found improvements and implemented them to create a stronger cryptosystem.

While work on this cryptosystem may never be considered complete, we have demonstrated an array of uses and validations. Further work in this area would be worthwhile, exploring projects such as:

**Hardware test implementations** Establish a ratio of throughput to gate count in actual hardware.

**Software throughput evaluation** Benchmark the software data throughput Improved QGBC with other cyrptosystems such as 3DES, Blowfish, AES and others.

**Additional cryptanalysis** Further attempts with Linear Cryptanalysis can be explored as well as Differential Cryptanalysis, and other techniques.

The merit of further research in the QGBC cryptosystem will continue to

establish the quality of the system and prove the cryptosystem to be a viable means of protecting data.

# References


[1] F. B. Gummere, *Beowulf*. PROJECT GUTENBERG, 1997, vol. 981.

[2] NIST, "Nist brief comments on recent cryptanalytic attacks on secure hashing functions and the continued security provided by sha-1," 2004.

[3] S. Paul and B. Preneel, "A new weakness in the rc4 keystream generator and an approach to improve the security of the cipher," *Fast Sfotware Encryption 2004 : Lecture notes in Computer Science*, pp. 245–259, 2004.

[4] D. Gligoroski and S. Markovski, "Cryptographic potentials of quasigroup transformations."

[5] NIST, "Secure hash standard (fips 180-4)," http://csrc.nist.gov/publications/fips/fips180-4/fips-180-4.pdf, March 2012. [Online]. Available: http://csrc.nist.gov/publications/fips/fips180-4/fips-180-4.pdf

[6] M. Matsui, "Linear cryptoanalysis method for des cipher," in *EUROCRYPT*, 1993, pp. 386–397.

[7] NIST, "A statistical test suite for the validation of random number generators and pseudo random number generators for cryptographic applications (fips 800-22)," April 2010. [Online]. Available: http://csrc.nist.gov/groups/ST/toolkit/rng/documentation_software.html

[8] M. Satti and S. Kak, "Multilevel indexed quasigroup encryption for data and speech," *IEEE Transactions on Broadcasting*, pp. 270–281, 2009.

[9] J. Rosenhouse and L. Taalman, *Taking Sudoku Seriously - The Math Behind the World's Most Popular Pencil Puzzle*. Oxford University Press, USA, 2011.

[10] A. A. Bruen and M. A. Forcinito, *Cryptography, Information Theory, and Error-Correction*. John Wiley & Sons, 2011.

[11] F. Miller, *Telegraphic code to insure privacy and secrecy in the transmission of telegrams*. C.M. Cornwell, 1882.

[12] G. S. Vernam, "Secret signaling system - u.s. patent 1,310,719," US Patent, Sept 1919.

[13] ——, "Cipher printing telegraph systems for secret wire and radio telegraphic communications," *Journal of the IEEE*, vol. 55, pp. 109–115, 1926.

[14] S. Markovski, D. Gligoroski, and L. Kocarev, "Unbiased random sequences from quasigroup string transformations," *Fast Software Encryption: 12th International Workshop*, pp. 163–180, 2005.

[15] M. Battey and A. Parakh, "An efficient quasigroup block cipher," *Wireless Personal Communications*, vol. 73, no. 1, pp. 63–76, 2013.

[16] R. Fisher and F. Yates, *Statistical tables for biological, agricultural and medical research*. Oliver and Boyd, 1953.

[17] A. Rukhin, J. Soto, J. Nechvatal, E. Barker, S. Leigh, M. Levenson, D. Banks, A. Heckert, J. Dray, S. Vo, A. Rukhin, J. Soto, M. Smid, S. Leigh, M. Vangel, A. Heckert, J. Dray, and L. E. B. Iii, "A statistical test suite for random and pseudorandom number generators for cryptographic applications," 2001.

[18] unknown, "Waveform audio format - 11,025 hz 16 bit pcm audio file," http://www.nch.com.au/acm/11k16bitpcm.wav, 2014. [Online]. Available: http://www.nch.com.au/acm/11k16bitpcm.wav

[19] J. H. Cheon, N. Hopper, Y. Kim, and I. Osipkov, "Provably secure timed-release public key encryption," *ACM Trans. Inf. Syst. Secur.*, vol. 11, no. 2, pp. 4:1–4:44, May 2008. [Online]. Available: http://doi.acm.org.leo.lib.unomaha.edu/10.1145/1330332.1330336

[20] M. Naor and O. Reingold, "Number-theoretic constructions of efficient pseudo-random functions," *J. ACM*, vol. 51, no. 2, pp. 231–262, Mar. 2004. [Online]. Available: http://doi.acm.org.leo.lib.unomaha.edu/10.1145/972639.972643

[21] J. Lee and I. Yeom, "Avoiding collision with hidden nodes in ieee 802.11 wireless networks," *Communications Letters, IEEE*, vol. 13, no. 10, pp. 743 –745, october 2009.

[22] V. Bharghavan, "Macaw: A medium access protocol for wireless lan's," in *Proc. ACM SIGCOMM Conference (SIGCOMM '94)*, august 1994, pp. 212–225.

[23] Q. Tong, X. Zou, and H. Tong, "A rfid authentication protocol based on infinite dimension pseudo random number generator," in *Computational Sciences and Optimization, 2009. CSO 2009. International Joint Conference on*, vol. 1, april 2009, pp. 292 –294.

[24] NIST, "Nist selects winner of secure hash algorithm (sha-3) competition," http://www.nist.gov/itl/csd/sha-100212.cfm, October 2012. [Online]. Available: http://www.nist.gov/itl/csd/sha-100212.cfm

[25] S. I. Inc., "Cryptospec.html – sha1prng," http://docs.oracle.com/javase/1.4.2/docs/guide/security/CryptoSpec.html#AppA. [Online]. Available: http://docs.oracle.com/javase/1.4.2/docs/guide/security/CryptoSpec.html$\#$AppA

[26] Oracle/Sun, "Secure-random," http://docs.oracle.com/javase/6/docs/api/java/security/SecureRandom.html, 2014. [Online]. Available: http://docs.oracle.com/javase/6/docs/api/java/security/SecureRandom.html

[27] OpenSSL.org, "rand(3)," http://www.openssl.org/docs/crypto/rand.html, 2014. [Online]. Available: http://www.openssl.org/docs/crypto/rand.html

[28] I. Microsoft, "Cryptgenrandom function," http://msdn.microsoft.com/en-us/library/windows/desktop/aa379942%28v=vs.85%29.aspx 2014. [Online]. Available: http://msdn.microsoft.com/en-us/library/windows/desktop/aa379942\%28v=vs.85\%29.aspx

[29] BSD, "Library functions manual – arc4random(3)," http://developer.apple.com/library/ios/#documentation/System/Conceptual/ManPages_iPhoneOS/man3/arc4random.3.html, 2014. [Online]. Available: http://developer.apple.com/library/ios/$\#$documentation/System/Conceptual/ManPages_iPhoneOS/man3/arc4random.3.html

[30] J. Soto, "Statistical testing of random number generators," NIST, Tech. Rep., 1999.

[31] Microsoft, "Random-number-generator," http://msdn.microsoft.com/en-us/library/system.security.cryptography.randomnumbergenerator.aspx, 2014. [Online]. Available: http://msdn.microsoft.com/en-us/library/system.security.cryptography.randomnumbergenerator.aspx

[32] H. M. Heys, "A tutorial on linear and differential cryptanalysis," *Cryptologia*, vol. 26, no. 3, pp. 189–221, 2002. [Online]. Available: http://www.tandfonline.com/doi/abs/10.1080/0161-110291890885

[33] M. Matsui and A. Yamagishi, "A new method for known plaintext attack of feal cipher," in *Lecture Notes in Computer Sciences, Advances in Cryptology, proceedings of EUROCRYPT'92*, 1992, pp. 81–91.

[34] J. H. van Lint and R. M. Wilson, *A Course in Combinatorics.* Cambridge University Press, 1992.

[35] B. D. McKay and I. M. Wanless, "On the number of latin squares," *Annals of Combinatorics*, vol. 9, no. DOI 10.1007/s00026-005-0261-7, pp. 335–344, 2005.

[36] XILINX, "What is an fpga?" March 2014. [Online]. Available: http://www.origin.xilinx.com/fpga/

[37] M. Battey and A. Parakh, "Efficient quasigroup block cipher for sensor networks," in *ICCCN*, 2012, pp. 1–5.

[38] D.-K. Corporation, "Fpga catalog," March 2014. [Online]. Available: http://www.digikey.com/product-search/en/integrated-circuits-ics/embedded-fpgas-field-programmable-gate-array/2556262

[39] M. Battey and A. Parakh, "A quasigroup based random number generator for resource constrained environments," *IACR Cryptology ePrint Archive*, vol. 2012, p. 471, 2012.

[40] M. Liberatori, F. Otero, J. C. Bonadero, and J. Castineira, "Aes-128 cipher. high speed, low cost fpga implementation," in *Programmable Logic, 2007. SPL '07. 2007 3rd Southern Conference on*, Feb 2007, pp. 195–198.

[41] H. Hsing, "Project: tiny_aes," October 2013. [Online]. Available: http://opencores.org/project,tiny_aes

[42] F. B. Gummere, "Beowulf," July 2008. [Online]. Available: http://www.gutenberg.org/ebooks/981