8-2017

Hybrid Quantum Encryption Device using Radioactive Decay

Anthony B. Kunkel
Saint Cloud State University, kuan0902@stcloudstate.edu

Hybrid Quantum Encryption Device using Radioactive Decay

by

Anthony Kunkel

A Thesis

Submitted to the Graduate Faculty of

St. Cloud State University

in Partial Fulfillment of the Requirements

for the Degree of

Master of Information Assurance

August, 2017

Thesis Committee:
Dennis Guster, Chairperson
Kevin Haglin
Renat Sultanov

Abstract

The future in how computing is done is heading in the direction of quantum computing given that the space used to store information is finite. Data will eventually be encoded using particles that are on the atomic scale. Objects of these scales are governed by the laws of quantum mechanics. Computing can be done exponentially faster using the properties provided by quantum mechanics. Unfortunately, the increase in computing power creates a security risk for modern encryption standards. Thus, to continue the transfer of data securely one must look to innovative encryption methods that protect information from the speed of quantum computers. This paper is focused on a method that secures information using radioactive decay events in conjunction with an encryption algorithm. The main purpose of this method is the develop an encryption device that holds quantum properties and is interfaceable with a computer system.

Acknowledgement

I would like to thank Dr. Dennis Guster, Dr. Kevin Haglin, Erich Rice, Karthik Paidi, and Dr. Renat Sultanov for all of their help and continued support in the completion of this project.

Table of Contents

Chapter Page

List of Tables

List of Figures

Chapter I: Introduction

Introduction

Security of personal information is a constant concern in everyday life. Without the secure transfer of this information, services like social media and online banking would not be possible. To reap the benefits of the internet one must accept the security risks of storing personal information (Wright, 2017). For the most part, current encryption algorithms have been effective at keeping sensitive information protected from potential malicious entities (Singh & Garg, 2005). However, recently limitations have been revealed in current encryption algorithms (Blumenthal, 2007). One specific algorithm that encrypts a significant amount of information, designed by Rivest, Shamir, and Adleman (RSA), has been shown to have issues. The algorithm has been extensively tested and continues to securely protect data, but due to the potential quantum computing future it may not remain secure (Bimpikis &Jaiswal, 2005; Lenstra, Lenstra Jr., Manasse, & Pollard, 1990; Mone, 2013; Oppliger, 2014; Sengupta & Das, 2017). Progress being made in quantum physics, as it is related to encryption technology, is changing the way we currently use cryptography (Edwards, 2017). Albeit, quantum computers are still in their infancy it is imperative that new encryption methods are designed. Some of these encryption methods have been designed but are difficult to implement and expensive (Haw et al., 2016). A solution proposed herein is to bridge the gap between the quantum and classical world. The proposed method is in the design of an inexpensive encryption device that holds quantum properties and is easily implemented on classical computers.

Problem Statement

A new way to encrypt information must be designed as current encryption schemes are at risk of becoming obsolete. The design of a hybrid quantum encryption device is proposed to take the first steps in a future of new encryption techniques.

Nature and Significance of the Problem

The secure transfer of data is of the utmost importance and a future that puts this process at risk must be addressed. With the progression of quantum computers and their threat to RSA encryption it is necessary that a hybrid encryption device be designed. The hybridization allows for the continued protection of information without changing the information transfer systems that we already have in place. The development of this device paves the way for new encryption algorithms to be designed that are resistant against quantum computer attacks.

Objective of the Study

The creation of a device that utilizes the quantum properties of nature but is implemented on classical computer systems is the main objective. To further solidify the usefulness of this device some goals should be reached. The radioactive element must produce reliably random numbers as a seed to be used by the algorithm. The information encrypted must also be complex enough to protect from guessing.

Study Questions/Hypotheses

The null hypothesis of this study is that the radioactive decay events generate random numbers.

Limitations of the Study

Limitations of the study is in part the vast data that was required to analyze the randomness of the generated numbers. The other issue comes from the length of time it takes to generate enough data points. The weak radioactive source outputs roughly 5,000 bits per hour. Statistical tests require on the order of 10^6 to 10^7 bits to provide significant results.

Summary

Personal information is constantly at risk of being obtained and now faces an even greater risk in the future of quantum computers. Current encryption methods work well but they will not work forever. As the quantum computing world continues to break ground it is now time to investigate new encryption techniques. The first step in this process is to transition to an area of cryptography that is between the quantum and classical worlds. It is proposed in this study that using a quantum random number generating source as a seed for an encryption algorithm, can be implemented on a classical computer. Moreover, the algorithm designed is not dependent on integer factorization and is strong against quantum computer attacks where RSA is weak. To further investigate this idea, it is useful to discuss the literature surrounding quantum computing, quantum key distribution, and the proposed encryption algorithm itself.

Chapter II: Background and Review of Literature

Introduction

In this chapter, the background information and literature reviewed in this study. Topics that will be discussed are: quantum key distribution (QKD), quantum computing, RSA encryption, and radioactive decay detection. Along with this, the literature from the hybrid encryption algorithm is also discussed.

Background and Literature Related to the Problem

Quantum cryptography began by the ideas presented by Stephen Wiesner. In the 1983 Wiesner published an article “Conjugate Coding” which argued that quantum systems isolated from the environment are irreproducible (Wiesner, 1983). Wiesner believed that a real-world application of this idea was to encode money on quantum systems. Therefore, if a quantum system is irreproducible then it would be impossible to make a counterfeit copy of the money. Charles Bennet brought this idea to Gilles Brassard and they developed the first quantum cryptography protocol called BB84 (Bennett & Wiesner, 1992; Brassard, 2005; Svozil, 2006). Conjugate coding is still quite important and work has been done to enhance its basic process (Hamada, 2006).

BB84 combined the ideas of public-key distribution and quantum mechanics to form QKD (Bennett & Brassard, 1984). The protocol transmits a quantum state $|\psi\rangle$ through an assumed secure quantum channel. The quantum state $|\psi\rangle$ is a two-state quantum particle known as a quantum bit (qubit) (Ballentine, 1970). A qubit, like a classical bit, hold values of a “0” or “1” but the quantum property of superposition allows both values to exist simultaneously. The property of superposition is only one of three

important quantum rules that BB84 utilizes. The second property exploited is that any observation on a quantum particle transforms the state of the particle (Nisticò & Sestito, 2016). Last, quantum mechanics forbids the reproducibility of any quantum state known as the no-cloning theorem (Wootters & Zurek, 1982). The second and third property illustrates the usefulness of quantum mechanics in cryptography. As an example, if a communication channel is insecure and an eavesdropper tried to access information from the quantum state, the transformation of the state would alert the sender and receiver of the insecure channel (Anghel, 2011). Additionally, it would be impossible for the eavesdropper to record information from the quantum state, reproduce an exact copy, and hide its presence on the channel. If the quantum states transform the sender and receiver must stop their communication and establish a new channel.

The theory of the BB84 protocol is very powerful but quantum physics makes it difficult and expensive to develop (Barde, Thakur, Bardapurkar, & Dalvi, 2012). Isolation of quantum systems from the environment is one of the greatest challenges faced in quantum cryptography. Quantum particle's tendency to interact with the environment often requires the system to be placed in a vacuum and held at very low temperatures, only a few degrees Kelvin (Dressel, Malik, Miatto, Jordan, & Boyd, 2014). In addition to the difficulties of protecting the quantum states it is also very expensive to keep the systems cooled and in a vacuum.

Ideas using QKD helped to facilitate the development of the first efficient quantum computer. It was Richard Feynman who suggested that the unique properties of quantum mechanics could allow for an extremely efficient computer (Feynman,

1986). He argued that the qubit would be faster than its classical counterpart, given the qubit inherently holds twice as many binary values. To provide a better illustration of this process a simple example is discussed. The example is designed to provide a quick review of the superposition properties.

In this example, a qubit is represented by an electron and quantum state can be described by the following equation:

$$|\psi\rangle = \frac{1}{\sqrt{2}}|0\rangle + \frac{1}{\sqrt{2}}|1\rangle,$$

where $|0\rangle$ and $|1\rangle$ are the electron's respective spin states. $|0\rangle$ represents the electron in a spin up state and $|1\rangle$ is the electron in the spin down state. The factors of $\frac{1}{\sqrt{2}}$ are the square root probability of observing each state. In quantum computing unitary quantum operators act on the qubit states, the operators are known as quantum gates. Like classical computer logic gates that transform bits, quantum gates transform qubits. One or more qubits are sent through the quantum gates until the algorithm that was implemented is complete. The qubits final state is then observed to output a desired $|0\rangle$ or $|1\rangle$. Since each qubit contains two values simultaneously, each quantum gate completes two operations at once. Compared to classical computations on a single bit a qubit increases the computations by two.

However, the number of qubits inside a quantum computer can be extended. In fact, as the number is extended to n-qubits, a quantum computer processes 2^n times more information than its classical counterpart. Such a speed increase is the main reason why the idea of quantum computers is so tantalizing. Quantum computer's

incredible computational power will allow for solving problems that are difficult and time consuming on classical computers.

A difficult problem central to this work is the integer factorization problem. Classical computers take an extremely long time to factor integers. RSA encryption is based on the classical computers lengthy factoring issue. RSA uses the product of two large prime numbers to encrypt keys for public-key distribution. The problem for classical computers gets even more difficult as the number of digits in the prime numbers increase. According to Kirsch (2015), “factoring time grows exponentially with input length in bits”.

However, the issue that RSA faces is that it is only secure and reliable if the speeds of computers stays relatively slow. Peter Shor illustrated this potential vulnerability. Using a quantum algorithm, known as Shor’s Algorithm, he showed that integers could be factored much faster. Shor’s Algorithm utilized the computational speed of qubits to solve the integer factorization problem in polynomial time, as opposed to classical algorithms which require exponential time (Shor, 1999). If Shor’s Algorithm is implemented on a quantum computer with enough qubits, it poses a direct threat to RSA encryption. At this time, quantum computers are not large or stable enough to solve the 2048 binary digit semi-prime used currently in RSA encryption. However, the designs of quantum computers are progressing and they appear to have the potential to solving this problem in the near future (Nordrum, 2016).

Literature Related to the Methodology

To prepare for a future where the use of RSA encryption is no longer reliable, it is the purpose of this discussion to propose a hybrid approach. A hybrid approach helps bridge the gap between classical and quantum cryptography. The proposed solution suggests the design of an encryption device that uses quantum principles that is then implemented on classical computers. The non-deterministic time between two consecutive radioactive decay events acts as the quantum property in the hybrid scheme (Rohe, 2003).

A radioactive source is an unstable element that decays to a more stable element by α-decay, β-decay, or γ-decay. These decay reactions can then be detected using a Geiger-Müller (GM) detector which converts detections into electrical pulses that can then be recorded. The time between two decay events is probabilistic by nature and can therefore be treated as a random number source. The random numbers are generated by comparing the time difference between consecutive decay events. Table 1 shows how each digit is represented.

Table 1: Radioactive Decay Time Bit Value. Associated bit value based on the relationship of consecutive decay events.

Relationship Between Decay Times	Bit Value
$\Delta t_1 < \Delta t_2$	0
$\Delta t_1 > \Delta t_2$	1
$\Delta t_1 = \Delta t_2$	Nothing

Δt_1 represents the time difference between the first and second recorded decay times, and Δt_2 represents the time difference between the third and fourth recorded decay

times. Assigning a bit value to the time difference relationship allows to generate decimal integers by converting the base-2 binary form to the base-10 equivalent.

The random numbers generated are used as a seed for the encryption algorithm developed by Paidi et al. (Paidi, Kunkel, Guster, Sultanov, & Rice, 2016). The original algorithm proposed used photon polarizations as the quantum source but it is far less expensive to use radioactive elements, as used in this study. Illustrating the randomness of radioactive decay will further validate the use of this algorithm. Further, when using the algorithm in conjunction with the random number generator it is critical to motivate the use of this method, rather than integer factorization, since it is not easily broken by quantum computers.

Summary

The ideas presented in the literature show how quantum mechanics can be exploited and used in the field of cryptography. Not only does quantum mechanics aid in the secure transfer of information but it also plays a role in shaking the basis of current encryption schemes. The material presented, further motivates the need for a new encryption scheme to continue protecting transferred data. Given that a purely quantum encryption system is expensive and difficult to implement, developing a hybrid system is a suitable first step. The following chapter will present the design of such a hybrid system and attempt to further motivate its complexity and usefulness.

Chapter III: Methodology

Introduction

To provide structure in presenting the methodology, this chapter will have three sections. The first section is focused on the design of the device used for the random number generator which will include several electronic components. The second section describes the encryption algorithm tailored for the use of the radioactive source device. The last section presents the randomness test suite that is used to analyze the numbers generated.

Design of the Study

Several electronic components are utilized to design a quantum random number generator device which supports the encryption algorithm. These components include: a Cesium-137 source, a GM detector, an Arduino, an Arduino interface shield, and a Raspberry Pi. Additionally, the Raspberry Pi and Arduino have programs that were developed to initiate the detection process and to collect the data. A block diagram of the device is illustrated in Figure 1.

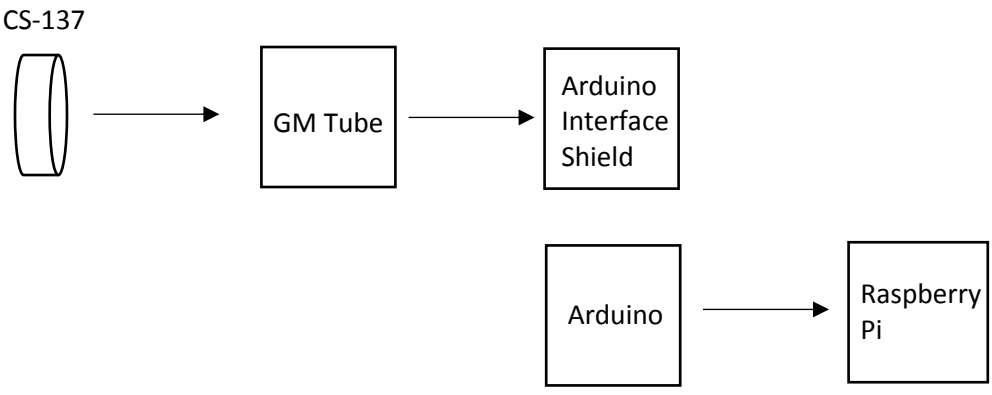

Figure 1: Device Block Diagram. Block diagram of the components of the encryption device.

The GM detector converts each decay event into a voltage signal that is then passed to the Arduino through the interface shield. After each detection, the detector is discharged for a finite period of time, which is known as the dead time. During this phase, no detections can be made. In the interest of efficiency, the selection of a detector with a relatively short dead time is ideal. The detector in the experiment had a dead time of 90 microseconds (μs) which is adequate for the data collection process. For the sake of device simplicity, the detector operates using a 5-volt (V) power source which is supplied by the Arduino. In fact, all the components, excluding the Raspberry Pi, run on 5V. The radioactive source chosen is a 1 microcurie Cesium-137 radioactive isotope. This radioactive isotope provides two advantages. First, the source is relatively weak so that it does not pose significant health concerns for extended exposure. Second, Cesium-137 has a longer half-life of 30.17 years and will therefore produce consistent decay events for several years.

The Arduino acts as a micro-controller that takes the signals from the GM detector and records the data. To generate the numbers the Arduino records the time difference between two consecutive decay events, following the program uploaded on the system. The program sends the recorded information to the serial port, which is subsequently extracted using a python program initiated on the Raspberry Pi. Each time difference is recorded in microseconds to allow for the highest accuracy in detection events.

Classical computing is done on the Raspberry Pi, which is a mobile computer interface that easily communicates with other computer systems. The extracted random

numbers from the Arduino's serial port is stored and sent to the encryption algorithm, which is described in more detail in the next section. At this data collection stage, the numbers are manually extracted and passed to the encryption algorithm. However, the Raspberry Pi's OS will allow the whole process to be automated on startup.

Given that the device is built with a Raspberry Pi and relatively small components, it can be effectively designed to be portable. The portability of the device allows it to act as a black box that could be connected to a computer system using a USB cable. A small amount of shielding could encase the system to protect users from the radiation source. However, merely distancing the user from the source may be enough, due to the considerable weakness of the source. The device may also be converted to a card that could be placed into a PCI Bus with limited change to the device configuration.

A second device could also be created for maximum efficiency. The second device acts as a receiver of the encrypted information sent by the first device. Advantages of a second device include the absence of the radioactive decay element. This greatly reduces cost and promotes greater safety to the user. The second device would only require that it includes the decryption algorithm. In a server-client setup, the first device would be housed on the server side while multiple second devices could be used on the client systems.

Encryption Algorithm

The first step in the algorithm accepts a user specified set of bits to be encrypted. In an operational system, the set of bits would be analogous to a secret key that would

be sent to the receiver. The complexity of the algorithm is customized by the user to create a unique and more robust encryption process. However, for this initial demonstration the simplest case will be presented as to not impede the reader's understanding.

Separating the randomly generated numbers, R, into two sections around an average number, N, four different cases arise for the encryption process.

Table 2: Four Cases. Four possible case conditions utilized in the encryption process.

	Relation Between the Generated Number and the Average	The First Bit
Case 1	$R \leq N$	0
Case 2	$R \leq N$	1
Case 3	$R > N$	0
Case 4	$R > N$	1

To reiterate again, this is the simplest case but the number of cases can be expanded as the number of sections the random numbers are separated into increase.

Based on which case the algorithm finds true, a conversion of bits phase initiates to the bit string.

- For Case 1 all bits are converted by a pseudo-randomly generated number between 100-549 if the bit is a 0, else a number between 550-999 is generated for a 1.

- For Case 2 all bits are converted by a pseudo-randomly generated number between 550-999 if the bit is a 0, else a number between 100-549 is generated for a 1.
- For Case 3 all bits are converted by a pseudo-randomly generated number between 550-999 if the bit is a 0, else a number between 100-549 is generated for a 1.
- For Case 4 all bits are converted by a pseudo-randomly generated number between 100-549 if the bit is a 0, else a number between 550-999 is generated for a 1.

The range of the pseudo-randomly generated numbers is arbitrary, but for simplicity in the string processing the ranges are chosen to contain three digits. Each range contains equal amount of numbers so that no bias is introduced that favors a particular range.

The decryption of bits is done using the same cases presented before but the process is reversed. Knowing the generated time and the first bit in the string, one can convert back to the original bit string using the following set of instructions.

- For Case 1 each converted bit is checked and if it lies in the range of 100-549 then it becomes a 0, else it becomes a 1.
- For Case 2 each converted bit is checked and if it lies in the range of 550-999 then it becomes a 0, else it becomes a 1.
- For Case 3 each converted bit is checked and if it lies in the range of 550-999 then it becomes a 0, else it becomes a 1.

- For Case 4 each converted bit is checked and if it is lies in the range of 100-549 then it becomes a 0, else it becomes a 1.

This method holds advantages in that each random decay event could restart the process, further complicating the ability to guess which method was used. Moreover, the quantum source used to seed the algorithm is independent of the encryption process, eliminating the predictability of which cases were used. However, for each random number used in the algorithm subsequently increases the communication complexity, as it requires a greater volume of information sent to the receiver. Remembering that the receiver must have knowledge of both the generated time and first bit to properly decrypt the message.

Data Analysis

To properly analyze the random numbers generated via the radioactive element it is important that they be subjected to a series of statistical tests. The way in which true- and pseudo-random number generators are tested is by test suites developed over the years. Some of the most widely used test suites include: Dieharder, NIST, ENT, and TESTU01 (Brown, Eddelbuettel, & Bauer, 2013; L'Ecuyer & Simard, 2007; Walker, 2008). The test suites each have their own advantages and disadvantages, but they all contain numerous random number statistical evaluations. To focus the scope of this discussion the NIST test suite, which is the cryptography standard, will be utilized. The primary goal is to use the tests in the suite to evaluate the validity of using the random numbers in the encryption algorithm and further, using the device itself.

Summary

The components used in the device design hold the advantage of being developed and implemented compactly and portable. Using the random numbers generated by the radioactive source as a seed in the algorithm, allows for the development of several cases that dictates the encryption process. Each radioactive decay has the potential to restart the encryption process further complexing the ability to predict the cases used. To be confident in the true randomness of the numbers generated, one must subject them to statistical test suites used in randomness testing. In the next chapter, the results from the encryption process, proposed complexities to the algorithm, and the analysis of the random numbers will be presented and discussed.

Chapter IV: Data Presentation and Analysis

Introduction

In this chapter, several aspects of the data collected are analyzed. An example of the previously described encryption algorithm is presented in its simplest working form. Complexities to the algorithm our introduced and investigated to further protect against decryption predictability. The radioactive decay events collected from the device are analyzed using the NIST randomness tests.

Data Presentation

As a demonstration of the algorithm's encryption process, an example is presented on a subset of hypothetical secret key bits. Only a subset is presented for simplicity and readability. The conversion phase of the algorithm on the bits is displayed in Table 3. In these examples, the device generated several 8-bit stings which were then converted to decimal integers.

An R of 183 and an average N of 128 is used in the example. Based on the cases and given that the first bit is a 1, the bits are converted using Case 4. According to the algorithms instructions, all bits that are 0 are converted to a range between 100-549 and all that are 1 are converted

between 550-999.

Table 3: Encrypted Bits Example. Table of encrypted bits with a random number of 183.

Bits	Encrypted Value
1	666
1	557
0	113
1	767
1	766
0	432
1	857
0	541
0	531
1	554
1	618
1	650
1	929
1	911
1	553
1	558
1	754
1	730
1	572
1	808
1	883
1	620
1	953
0	243
0	304
0	389
0	110
0	392
1	612
0	496
0	292
1	566

With knowledge of which case was used in the conversion phase it is straightforward to decrypt back to the original bits. The decryption phase is processed in a similar manner as the conversion phase but in reverse. Therefore, all converted bits in the range of 100-549 will be reverted to a 0 and all in the range of 550-999 will become a 1.

Extending the number of bits to the order of a several thousand, it is important to investigate the way the algorithm acts in a more complex situation. As it is not feasible or easily readable to present this information in tables, it is more useful to present the information graphically. The set of converted bits are displayed in Figure 2. The x- and y-axis represent the number of bits converted and the encrypted value, respectively.

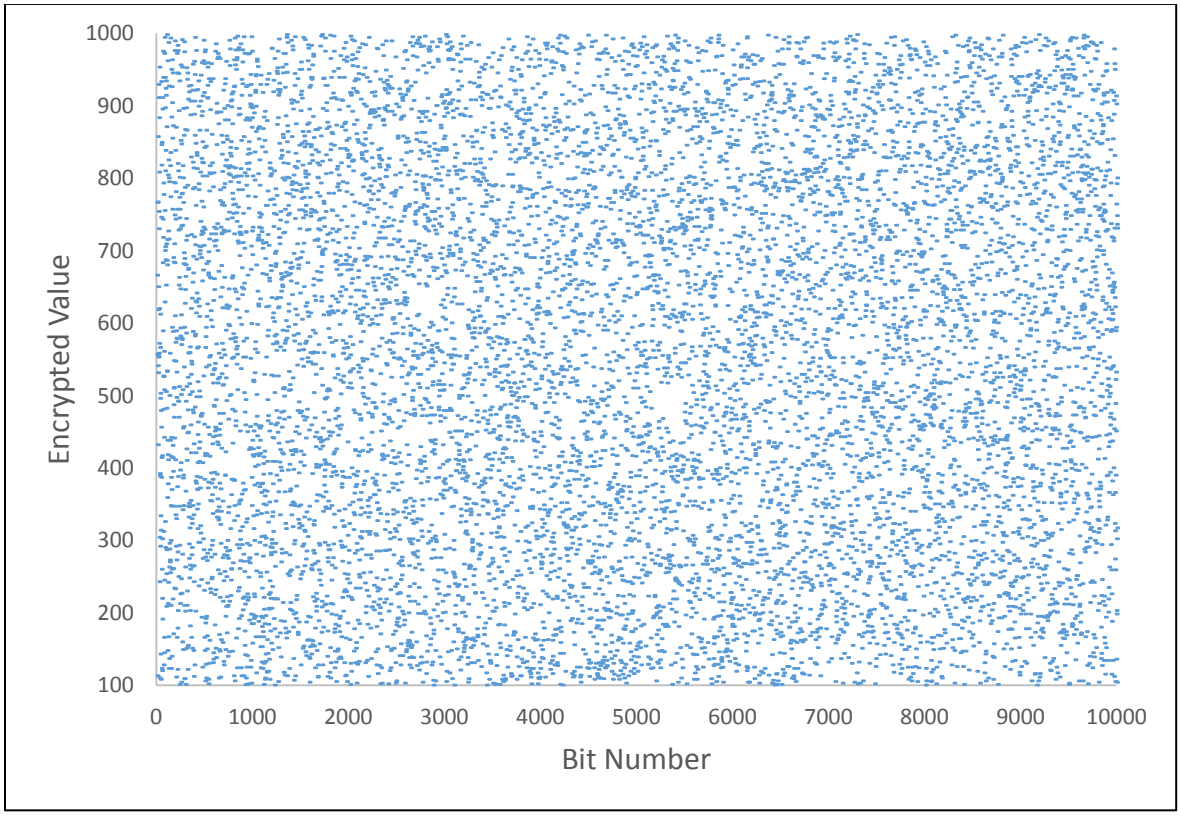

Figure 2: 10,000 Encrypted Bits. Graphical representation of 10,000 encrypted bits.

Using the data plotted in Figure 2 it is important to investigate how the distribution of encrypted bits relates to one another. Specifically, the values that are encrypted must not favor a certain number. A number that appears more frequently could create a bias and an attacker could focus more on that number and possible obtain more information about the original bit string. Figure 3 displays the data of how frequently each encrypted value is used in the 10,000-bit conversion process.

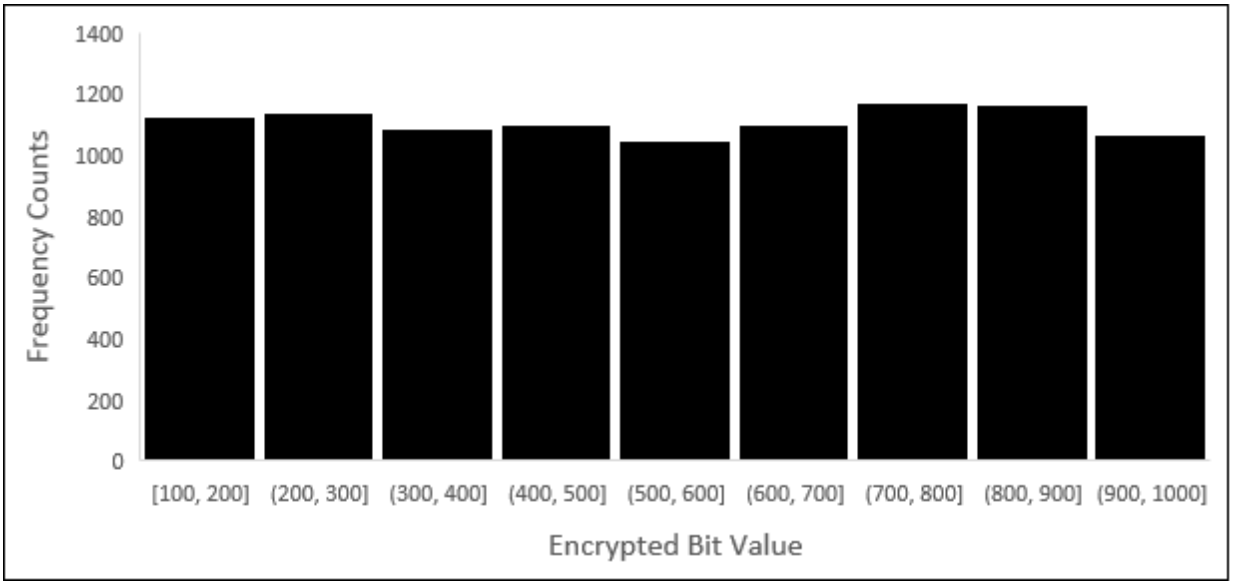

Figure 3: Frequency of encrypted 10,000-bit string.

The data shows nearly uniform frequency for all bits converted. Uniform frequency is desirable in this case since each encrypted value is equally likely to appear.

Given that the encrypted values are uniform in frequency the next logical question to ask is if each case used is equally likely. Assume that there is a four-sided die with each case written on a respective side. Probability states that there is a 25% chance of rolling any one side. The following data was collected by testing the algorithm several times and recording the number of times each case was used. The cases

required that the algorithm used different random numbers in conjunction with a binary string. Two different binary strings were used in these tests. One string was generated by a C++ random bit generator. The other string was taken from the binary numbers generated by the device. Using the different strings probes the idea that the algorithm may be dependent on using a true random binary string. The bit lengths of each string were 168-, 256-, and 1024-bit to mimic the actual size of a key that would be used in the algorithm.

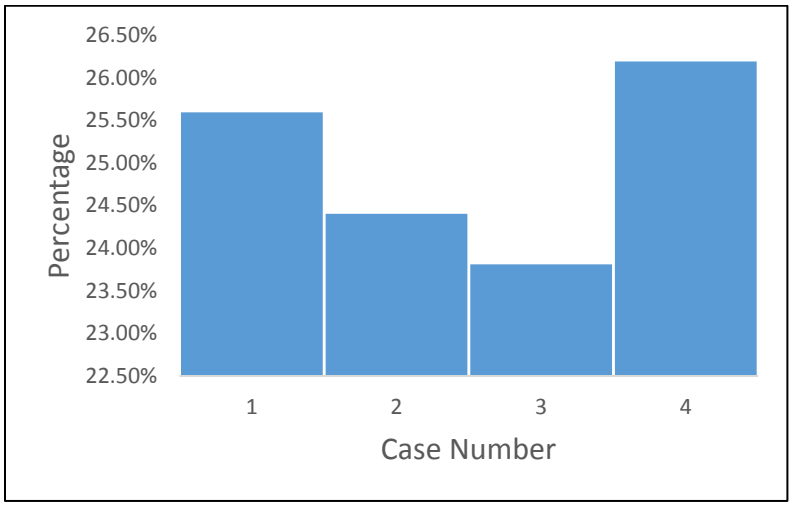

Figure 4: C++ 168-bit String Cases. Percentage of cases used with a C++ generated 168-bit string.

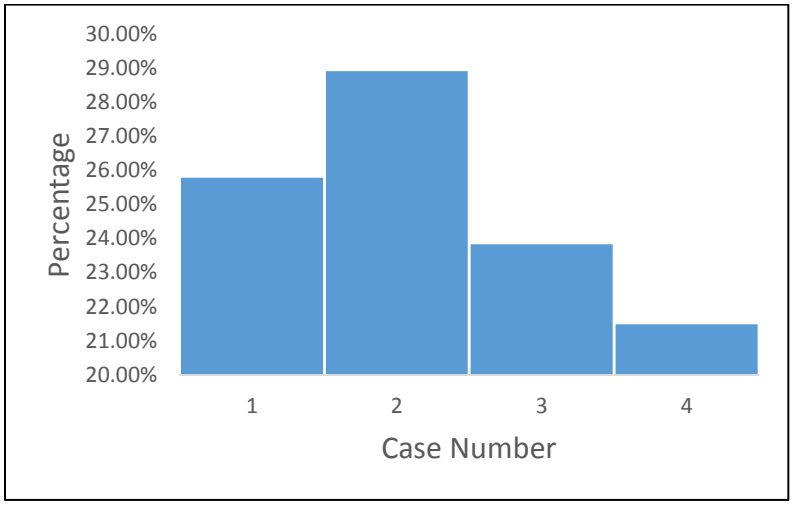

Figure 5: C++ 256-bit String Cases. Percentage of cases used with a C++ generated 256-bit string.

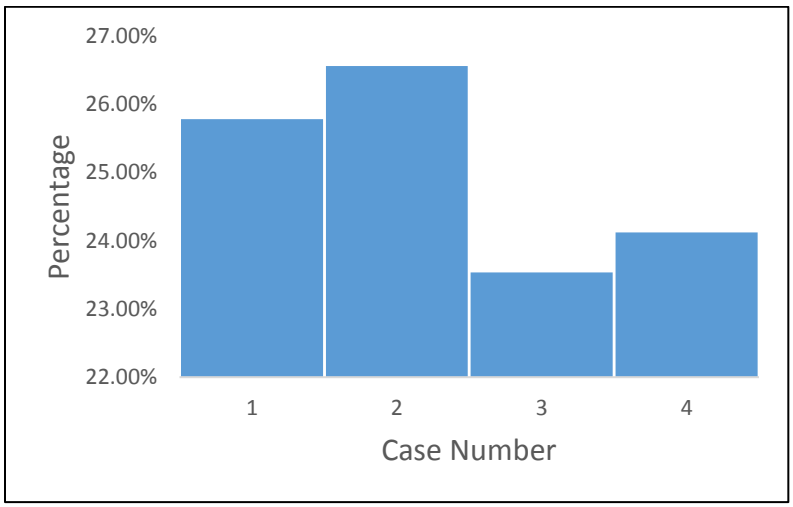

Figure 6: C++ 1024-bit String Cases. Percentage of cases used with a C++ generated 1024-bit string.

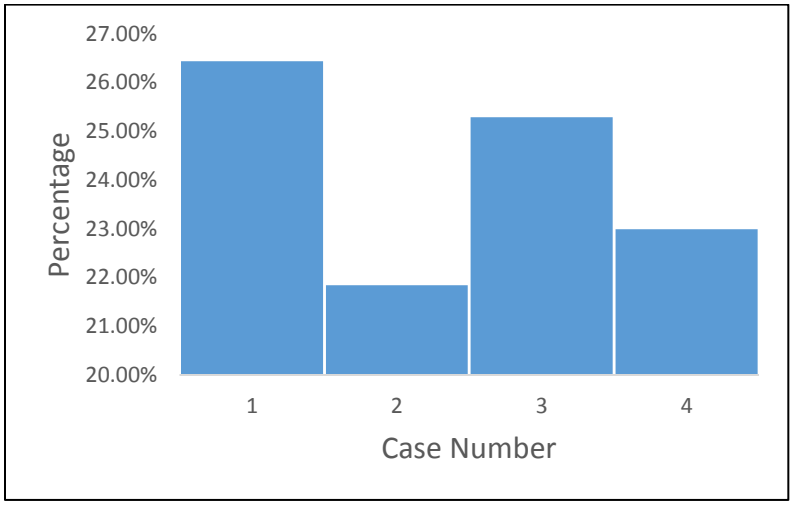

Figure 7: Device 168-bit String Cases. Percentage of cases used with a device generated 168-bit string.

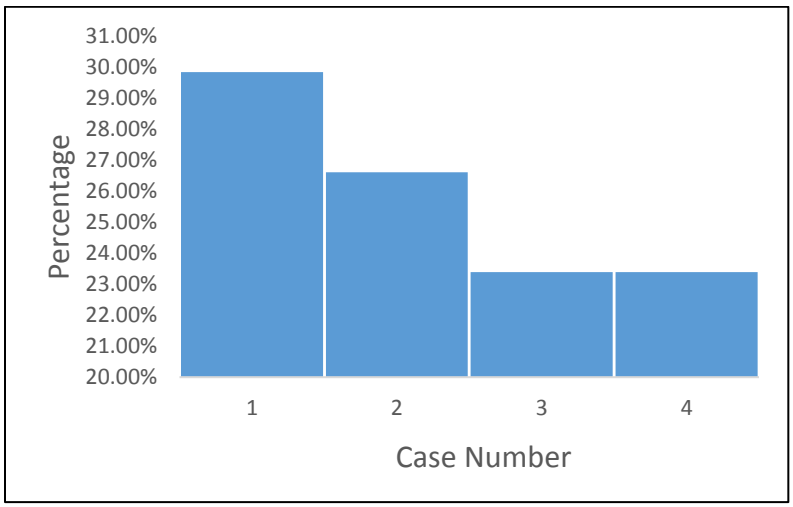

Figure 8: Device 256-bit String Cases. Percentage of cases used with a device generated 256-bit string.

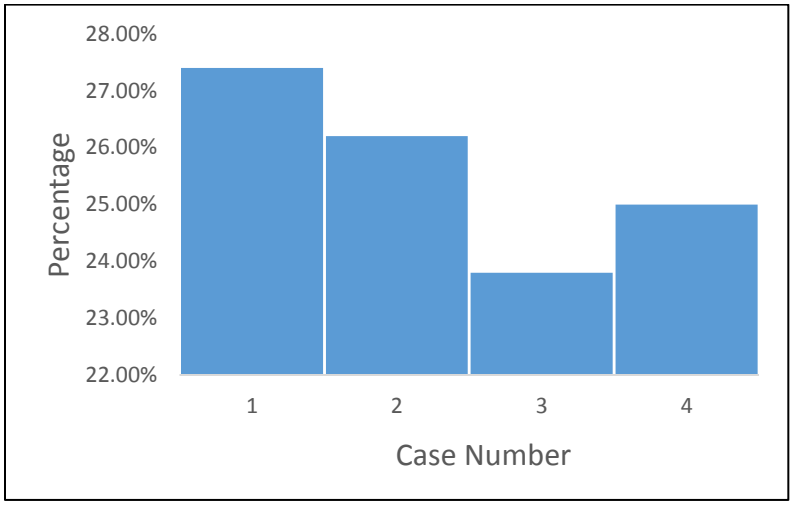

Figure 9: Device 1024-bit String Cases. Percentage of cases used with a device generated 1024-bit string.

The percentages displayed using the C++ generated bits are displayed in Figures 4-6. These percentages do not equate to each case being used 25% of the time as expected. Cases 2 and 4 are more prevalent in these figures as compared to cases 1 and 3. All cases are above 21% and below 30% for each bit string, which is encouraging. Figures 7-9 also do not meet the 25% expectation. Interestingly, case 1 is most prevalent in all three figures. The case percentages were above 21% and below 30% just as the C++ generated produced. The usefulness of these results indicate that the cases do not vary too strongly away from the ideal 25% case distribution. They also present little difference in the choice of bit strings used. Such results could indicate that the algorithm is more dependent on the random numbers generated, rather than the input binary strings.

A distribution of the cases used in the encryption process is an important step in increasing the complexity of the device. In the original example of showing the encryption process of the algorithm only one case was used for simplicity. Although, a

simplistic approach would not work in a real-world situation. If an attacker were to obtain or “guess” the case used, they would be able to obtain the whole original bit string through the decryption process. Since this is clearly undesirable, it would be more beneficial to use several cases, more specifically, use more random generated numbers. Such a feat can be achieved by dividing the bit string into several pieces and using a new random number for each piece.

In the following example, a binary string is divided up into several 8-bit sections and a new case is used at the start of each section. Table 4 displays a sample of original bits, their encrypted value, and the cases used in each section.

Table 4: Encrypted Bits Multiple Cases Example. Table of encrypted bits with multiple cases used.

Original Bits	Encrypted Value	Case Number
1	337	
1	138	
0	925	
1	474	Case 2
1	201	
0	552	
1	228	
0	378	
0	532	
1	893	
1	618	Case 1
1	639	
1	586	
1	602	
1	785	
1	594	
1	748	
1	911	
1	608	Case 4
1	883	
1	884	
1	777	
1	663	

The table illustrates three important ideas about a potential attacker obtaining the original bit string. First, with the correct guess of a case the attacker only uncovers a small section of the original data. Second, the attacker must continually guess a new case in order to obtain the exact original string. Finally, the 8-bit division suggested in

the example is completely arbitrary in nature. One could choose 2-, 6-, or 24-bit divisions if so desired. The number of sections also do not have to be constant and can change as the bit string progresses. The only requirement is that the information be hard-coded into the decryption algorithm.

Another step in a more complex and secure algorithm comes from expanding the number of cases used. Using only four cases would give an attacker a relatively easy trial-and-error problem that would not take too long to solve. Since the value of a bit is fixed, the next logical step in case expansion would be to split the average of the random numbers into more sections. Currently, the average is only split into two sections: either the random number is less than or equal to, or greater than the average. For simplicity, we will split the average in half to create a total of four new sections. On top of that, there are two values a bit can hold which translates to eight conditions in all. However, there is still an issue that must be addressed. In the original example, the encrypted value is placed into two sections, 100-549 and 550-999. Only two sections are not ideal since one can easily see that some cases will share the same ranges for both a zero or one bit value. A solution to this problem is to create four sections of encrypted values just as we split the average into. Table 5 breaks down each possible condition and their respective encrypted values. Since each case does not extend the entire range from 100-999, multiple cases should be used to further hide original bits.

Table 5: Eight Cases. Eight conditions that can be used to increase security in the algorithm.

	Random Generated Number	Bit	Encrypted Value For 0	Encrypted Value For 1
Case 1	$0 \leftrightarrow \frac{N}{2}-1$	0	100-324	325-549
Case 2	$\frac{N}{2} \leftrightarrow N-1$	0	325-549	100-324
Case 3	$N \leftrightarrow N+\frac{N}{2}-1$	0	550-774	775-999
Case 4	$N+\frac{N}{2} \leftrightarrow 2N-1$	0	775-999	550-774
Case 5	$0 \leftrightarrow \frac{N}{2}-1$	1	775-999	100-324
Case 6	$\frac{N}{2} \leftrightarrow N-1$	1	550-774	325-549
Case 7	$N \leftrightarrow N+\frac{N}{2}-1$	1	325-549	550-774
Case 8	$N+\frac{N}{2} \leftrightarrow 2N-1$	1	100-324	775-999

Data Analysis

The data generated by the device was tested to determine if the information collected is random. 361 MB of data was collected over a five-month period. All the data was stored in a text file which contains several million 8-bit strings of ASCII 0's and 1's. As a preliminary test, the binary strings were converted to decimal integers between 0 and 255. The integers were then plotted in a histogram as shown in Figure 10.

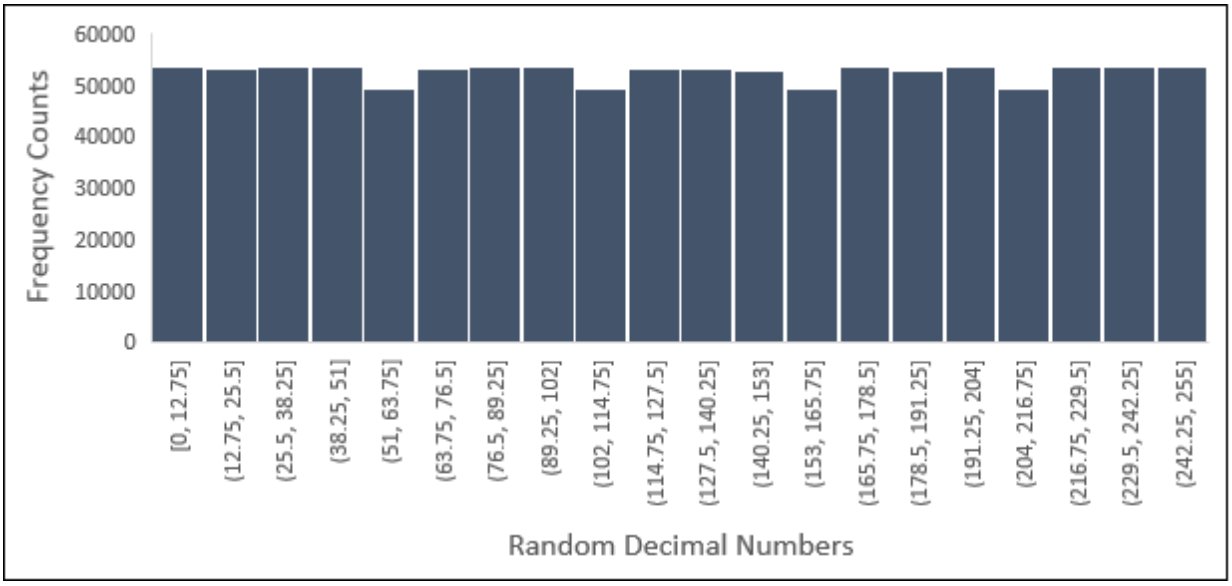

Figure 10: Histogram of Decimal Integers. Frequency counts of random decimal integers generated by the device.

The histogram shows that the distribution of decimal numbers is nearly uniform for all the data in the file. Much like the encrypted bits graph, the uniform distribution is most desirable. If the data that seeds the algorithm has an equally likely chance to be selected, it makes it much more difficult for an attacker to guess the seed with high certainty.

Although, preliminary tests are encouraging the random numbers generated need to be subject to more robust tests. These tests are well established in the NIST randomness test suite. NIST includes fifteen different tests in the suite, each test investigating different types of non-randomness. A final report is given as a text file which contains several p-values for the sub-tests of each of the fifteen tests. The report also includes the proportions of the p-values that passed the significance level. The significance level, α, used in the tests was set at 0.01. Therefore, p-values that are greater than or equal to α accept the null hypothesis (the binary sequence is random). A

histogram of the p-values generated from the NIST test suites for all the tests and sub-tests are shown in Figure 11.

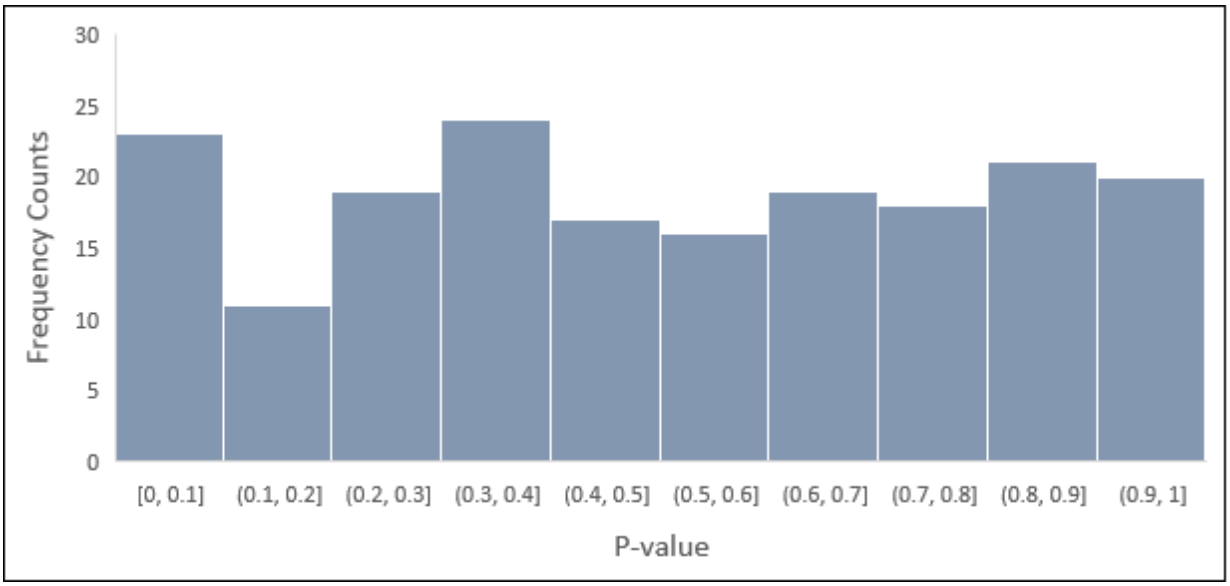

Figure 11: Histogram of NIST P-values. Frequency counts of the p-value calculations from the NIST randomness test suite.

Only one test failed the p-value test which was the Maurer's "Universal Statistical" Test with a p-value of 0. Such a distinct p-value indicates that the number of input bits was insufficient to compute a proper p-value. Therefore, for a correct p-value computation requires more data to be collected which is part of the limitations of the study.

The proportion of p-values that passed each test and sub-test was also given in the final analysis text file. Based on the input bits subjected to the tests required that 96 out of the 100 binary sequences p-values must be greater than or equal to α. One test and one sub-test failed to meet the proportion requirement. The Maurer's Test and a Non-Overlapping Template sub-test resulted in 0 and 95 p-values passed, respectively. The final analysis of the NIST test suite is given in the appendix.

Summary

The data collected in this study was used to illustrate an example of the encryption algorithms process. Two forms of the example were displayed, one being a simple example using only a few bits. The other was a more complex example with a large number of bits used. Complexities added to the algorithm were investigated to make it more difficult to break the encryption. Case frequency was tested to determine if each case had an equal chance of being chosen via the random numbers generated. The algorithm was shown to change cases multiple times within the same bit string to decrease the amount of information obtainable by a single case. Further expansion of the number of cases were developed to slow an attacker trying to obtain the original bit string. Using the NIST test suite to analyze the data presented the p-values computed from each test. In the next chapter, the data analysis will be discussed and conclusions will be made on the findings.

Chapter V: Results, Conclusion, and Future Work

Introduction

In this chapter, the data that was analyzed will be discussed further. Conclusions will also be made on the findings from the collected data. Finally, work that still needs to be done in the future will be presented.

Results

The initial example was simply an active implementation of the algorithm. It is quickly realized that as the encryption works as designed but it lacks complexity and could be easily broken. A plot and histogram of 10,000 bits encrypted using the initial algorithm shows that there is minimal bias in the values that the bits are encrypted to. Recording the cases used for several implementations of the algorithm on multiple bit strings motivated the use of several device generated random numbers to increase the encryption's complexity.

The second step in raising the complexity of the algorithm was designed by creating more conditions and therefore more cases. The new algorithm would make it more difficult to crack and could translate to even more than eight cases if constructed properly. However, more cases required sectioning the encrypted value ranges and lead to each case only spanning a portion of the entire 100-999 possible values. Therefore, the algorithm should use the expanded cases and multiple random numbers to span all possible values between 100-999.

The p-values computed from the final results of the NIST test suites are very encouraging but the data did fail in two areas. Maurer's Test failed to produce a p-value

for any of its bit sequences and therefore also failed the proportion test. Further, the Non-Overlapping Template sub-test missed the proportion test by one p-value. Fortunately, the histogram of p-values, for the most part, was uniform as expected by the NIST test suite. Collecting a greater sample of data would lead to the passing of all tests if the analysis is correct.

Conclusion

A key point in the discussion of this study is that the proposed device uses hybrid properties to combat the difficulty in the current state of quantum encryption implementation. The device has advantages in that the encryption process does not depend on the factorization of integers like RSA encryption. This fact alone makes it more powerful against quantum computer attacks. The size of the device is advantageous as it can be portable and interfaced easily with a computer system. With little change to the device configuration one can easily make the device include a plug-and-play interface to simplify its use for a user. Moreover, the cost of the encryption device is extremely low and could be done for less than $300.

Increasing the complexity of the algorithm as proposed would further solidify its use in a real-world scenario. Although, the encryption process should be extensively tested against standard hacking attacks to be more conclusive. The results from the NIST test suite allows for the acceptance of the null hypothesis, except in two tests.

The security risk posed by quantum computers to current encryption schemes further motivates the idea that new algorithms must be pursued. Given the complex

nature of quantum encryption, it is apparent that the development of hybrid systems is the logical first step in securing personal information.

Future Work

To improve this study more data must be collected to further analyze the randomness of the numbers generated. Using a larger sample of random numbers will produce a value for the Maurer's Test and hopefully provide a passing p-value. It would also give a new proportion test for the Non-Overlapping Template sub-test. A useful next step would be to automate the device so that information can be encrypted just by providing power to the device. This step was not taken in the study as it was more focused on data collection rather than user simplicity. Further steps can also be taken to design a second device without the radioactive element or detector. The second device would act as a receiver and obtain information from the first device and decrypt what is received. Once developed, the two devices could create a local system that could be probed for potential holes in the encryption not currently investigated in this study.

References

Anghel, C. (2011). New eavesdropper detection method in quantum cryptography. *Annals of Dunarea De Jos, Vol 34, Iss 1, Pp 1-8 (2011),* (1), 1.

Ballentine, L. E. (1970). The statistical interpretation of quantum mechanics. *Reviews of Modern Physics, 42*(4), 358-381. doi:10.1103/RevModPhys.42.358

Barde, N., Thakur, D., Bardapurkar, P., & Dalvi, S. (2012). Consequences and limitations of conventional computers and their solutions through quantum computers. *Leonardo Electronic Journal of Practices and Technologies, Vol 10, Iss 19, Pp 161-171 (2012),* (19), 161.

Bennett, C. H. and Brassard, G. (1984). Quantum cryptography: Public key distribution and coin tossing. In *Proceedings of the IEEE International Conference on Computers, Systems and Signal Processing*, pages 175–179, New York. IEEE Press.

Bennett, C. H., & Wiesner, S. J. (1992). Communication via one-and two-particle operators on einstein-podolsky-rosen states. *Physical Review Letters, 69*(20), 2881.

Bimpikis, K., & Jaiswal, R. (2005). Modern factoring algorithms. *University of California, San Diego.*

Blumenthal, M. (2007). Encryption: Strengths and weaknesses of public-key cryptography. *CSRS 2007,* 1.

Brassard, G. (2005). Brief history of quantum cryptography: A personal perspective. *Theory and Practice in Information-Theoretic Security, 2005. IEEE Information Theory Workshop on,* 19-23.

Brown, R. G., Eddelbuettel, D., & Bauer, D. (2013). Dieharder: A random number test suite. *Open Source software library, under development*.

Dressel, J., Malik, M., Miatto, F. M., Jordan, A. N., & Boyd, R. W. (2014). Colloquium: Understanding quantum weak values: Basics and applications. *Reviews of Modern Physics, 86*(1), 307.

Edwards, C. (2017). Secure quantum communications. *Communications of the ACM, 60*(2), 15-17. doi:10.1145/3022179

Feynman, R. P. (1986). Quantum mechanical computers. *Foundations of Physics, 16*(6), 507-531.

Hamada, M. (2006). Conjugate codes and applications to cryptography. *arXiv preprint quant-ph/0610193.*

Haw, J. Y., Zhao, J., Dias, J., Assad, S. M., Bradshaw, M., Blandino, R., . . . Lam, P. K. (2016). Surpassing the no-cloning limit with a heralded hybrid linear amplifier for coherent states. *Nature Communications, 7*, 13222-13222. doi:10.1038/ncomms13222

Kirsch, Z., & Chow, M. (2015). *Quantum Computing: The Risk to Existing Encryption Methods.*

L'Ecuyer, P., & Simard, R. (2007). TestU01: AC library for empirical testing of random number generators. *ACM Transactions on Mathematical Software (TOMS), 33*(4), 22.

Lenstra, A. K., Lenstra Jr, H. W., Manasse, M. S., & Pollard, J. M. (1990). The number field sieve. In *Proceedings of the twenty-second annual ACM symposium on Theory of computing* (pp. 564-572). ACM.

Mone, G. (2013). Future-proof encryption. *Communications of the ACM, 56*(11), 12-14.

Nisticò, G., & Sestito, A. (2016). “Evaluations” of observables versus measurements in quantum theory. *International Journal of Theoretical Physics, 55*(3), 1798-1810. doi:10.1007/s10773-015-2819-4

Nordrum, A. (2016). Quantum computer comes closer to cracking RSA encryption. *IEEE Spectrum.*

Oppliger, R. (2014). Secure messaging on the internet. (pp. 57-58) Artech House.

Paidi, K., Kunkel, A., Guster, D., Sultanov, R., & Rice, E. (2016). A hybrid quantum encryption algorithm that utilizes photon rotation to insure secure transmission of data. In *Proceedings of the 2016 Midwest Instruction and Computing Symposium.*

Rohe, M. (2003). RANDy-A true-random generator based on radioactive decay. *Saarland University,* 1-36.

Sengupta, B., & Das, A. (2017). Use of SIMD-based data parallelism to speed up sieving in integer-factoring algorithms. *Applied Mathematics and Computation, 293*, 204-217. doi:http://dx.doi.org.libproxy.stcloudstate.edu/10.1016/j.amc.2016.08.019

Shor, P. W. (1999). Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer. *SIAM Review, 41*(2), 303-332.

Singh, D. G., & Garg, D. (2005). Soft computing. *Allied Publishers*.

Svozil, K. (2006). Staging quantum cryptography with chocolate balls a. *American Journal of Physics, 74*(9), 800-803.

Walker, J. (2008). Ent: A pseudorandom number sequence test program. *Software and Documentation Available at/www.Fourmilab.ch/random/S,*

Wiesner, S. (1983). Conjugate coding. *ACM Sigact News, 15*(1), 78-88.

Wootters, W. K., & Zurek, W. H. (1982). A single quantum cannot be cloned. *Nature, 299*(5886), 802-803.

Wright, A. (2017). Mapping the internet of things. *Communications of the ACM, 60*(1), 16-18. doi:10.1145/3014392

Appendix A

```
------------------------------------------------------------------------
RESULTS FOR THE UNIFORMITY OF P-VALUES AND THE PROPORTION OF
PASSING SEQUENCES
------------------------------------------------------------------------
   generator is </home/Quantum/RandomGen/8bit07102017ent.txt>
------------------------------------------------------------------------
 C1  C2  C3  C4  C5  C6  C7  C8  C9 C10  P-VALUE  PROPORTION
STATISTICAL TEST
------------------------------------------------------------------------
  8   6  11  16   9   6  17   4  13  10  0.051942    100/100
Frequency
 13   7   8   8  10  14  13   9  11   7  0.719747     99/100
BlockFrequency
 10   6   9  15  14  12   9  11   9   5  0.437274    100/100
CumulativeSums
  9  10  11  11   9   9  12  12   7  10  0.987896    100/100
CumulativeSums
 10   7  12   7   9   6  11  11  13  14  0.678686    100/100
Runs
 11   6   8  11   9   9  14   9  12  11  0.867692     97/100
LongestRun
  8   7  11  13  13   9   9   9   9  12  0.911413     99/100
Rank
 12  13  11  10  11   8  10   6  12   7  0.851383     96/100
FFT
 14  12   5   9   8  13  10  12  12   5  0.419021     98/100
NonOverlappingTemplate
 15  11  11  15   9   7   9  10   9   4  0.350485     97/100
NonOverlappingTemplate
 15   9  11  11  11  11   5   9  10   8  0.739918    100/100
NonOverlappingTemplate
 17  13   8   4   6  13   7  12  14   6  0.051942     97/100
NonOverlappingTemplate
 10  10  16   6  13  12  10  10   9   4  0.334538     98/100
NonOverlappingTemplate
 10  14   7  14  10   7  10   7  12   9  0.699313     97/100
NonOverlappingTemplate
  9  11  12  12  14   9   9   5  12   7  0.678686     98/100
NonOverlappingTemplate
```

```
 11  14  14  10   5  10   9  14   8   5  0.319084    100/100
NonOverlappingTemplate
 13  11   9  10   9   9   8   8   9  14  0.924076     98/100
NonOverlappingTemplate
 12  16  11   7   9  10   9   9   6  11  0.637119     98/100
NonOverlappingTemplate
 11  11   5   8   9  11  17   9   9  10  0.494392     99/100
NonOverlappingTemplate
 15   8  10  14  11   6   8  10   9   9  0.657933     99/100
NonOverlappingTemplate
 10  12   7   8   8   7  11  11  11  15  0.759756    100/100
NonOverlappingTemplate
  6  13  13   8   9   7  13  13   7  11  0.574903    100/100
NonOverlappingTemplate
 11   5  14  12  11   6   9   9   9  14  0.514124     98/100
NonOverlappingTemplate
 16  11  10  13   6   1   8  13   7  15  0.025193     97/100
NonOverlappingTemplate
  6  11  10  10  13  12   9   8  13   8  0.851383    100/100
NonOverlappingTemplate
 13  15   6  11   9  13   5   8   9  11  0.419021     98/100
NonOverlappingTemplate
  9   8  10  11   9   9  10  10   9  15  0.946308     98/100
NonOverlappingTemplate
  5   7  13  13   9  13   8  11  14   7  0.419021     99/100
NonOverlappingTemplate
  5  11  10   9   7  10  13  10  11  14  0.719747     98/100
NonOverlappingTemplate
  6  10  14  12   4  15  11   5  11  12  0.171867    100/100
NonOverlappingTemplate
  9   9  14   8  11  14   6   8  13   8  0.616305    100/100
NonOverlappingTemplate
  7  11   9  10  12  15  10   8  10   8  0.851383    100/100
NonOverlappingTemplate
 10  13  13   6  10   7  12  10  10   9  0.851383     99/100
NonOverlappingTemplate
  9   7  10  11   7   8   6  14  15  13  0.437274     99/100
NonOverlappingTemplate
 11   8  11   6   9  19   8   9   8  11  0.249284     95/100  *
NonOverlappingTemplate
 15   8   6  11   9  13   8   7  13  10  0.554420    100/100
NonOverlappingTemplate
 13   8  10  12  13   9   9  12   9   5  0.759756    100/100
NonOverlappingTemplate
```

```
 10  12  11   9  10  10   8  10   9  11  0.998821     99/100
NonOverlappingTemplate
 12   6  10  10  17  10  10  11   5   9  0.383827     98/100
NonOverlappingTemplate
 12  10  11  10  10  14   6  11   9   7  0.851383     98/100
NonOverlappingTemplate
  9  11  12  12   8  11  12  11   9   5  0.867692     99/100
NonOverlappingTemplate
 18   7   7  16   7   9  12   5   9  10  0.071177     99/100
NonOverlappingTemplate
 13   8   7   9  13  11   8  12  12   7  0.798139     99/100
NonOverlappingTemplate
 10   9   7  16  11   9   8  10   9  11  0.798139     98/100
NonOverlappingTemplate
 15   9   9  14  11   9   3  13  11   6  0.213309    100/100
NonOverlappingTemplate
 10   8  15   9  11  11  14   7   6   9  0.595549     99/100
NonOverlappingTemplate
 10  12  12  13  10  14   5   8   9   7  0.616305     97/100
NonOverlappingTemplate
 15   8   7   8  13  15  10   6  12   6  0.262249     97/100
NonOverlappingTemplate
  7   5   9  13   7  10  18  11  10  10  0.224821    100/100
NonOverlappingTemplate
  9   8  12   7  10   9   5  15  12  13  0.514124     99/100
NonOverlappingTemplate
 10   9  11  10  11  10   8  12  11   8  0.996335    100/100
NonOverlappingTemplate
  6  12  15  11  15   7  11   6   7  10  0.304126     99/100
NonOverlappingTemplate
 13   2  13  12  17  14   4   6  13   6  0.006661     99/100
NonOverlappingTemplate
 13   2  11   9  15   8  12   7   7  16  0.062821    100/100
NonOverlappingTemplate
 11   8  10   4  10   8  15  11  11  12  0.574903    100/100
NonOverlappingTemplate
  8   7   8  16  10   8   9  13  12   9  0.616305     99/100
NonOverlappingTemplate
 14  10  10   6   8  14  10   9  10   9  0.798139    100/100
NonOverlappingTemplate
 11   5  12   7  13  14  10   7  14   7  0.366918     98/100
NonOverlappingTemplate
  8   6   8  24   6  10  12  11   8   7  0.002559     99/100
NonOverlappingTemplate
```

```
  6  13  12  11  11   9  13   7  10   8  0.798139     98/100
NonOverlappingTemplate
  9  13  13  10   7  12   9  11  10   6  0.834308     98/100
NonOverlappingTemplate
  9  13   9   8  10   7  14  13   6  11  0.678686    100/100
NonOverlappingTemplate
  9   9   9  12  13   6   9  13   8  12  0.834308    100/100
NonOverlappingTemplate
 12   9  13  12   9   7   8   7  11  12  0.867692     99/100
NonOverlappingTemplate
  5  10  11   5   8   9  11  16  13  12  0.304126    100/100
NonOverlappingTemplate
  8  14   7  12   8  10   6  12  12  11  0.719747     98/100
NonOverlappingTemplate
 10  17   6   9   3  17   5  11   9  13  0.017912     98/100
NonOverlappingTemplate
  5  11  17   5   9  12  13  12   5  11  0.108791     98/100
NonOverlappingTemplate
 13  11  12   9  11   3  13  11   9   8  0.534146    100/100
NonOverlappingTemplate
 18  13  12   9  10   9  11   4  10   4  0.085587     97/100
NonOverlappingTemplate
 11  11   7   6  11   8  12  13   8  13  0.759756     99/100
NonOverlappingTemplate
  8   9   8   7   8  13  11  16  11   9  0.637119     99/100
NonOverlappingTemplate
 13   9  10   8  11   8   9  14   8  10  0.911413     97/100
NonOverlappingTemplate
  9   8  11  10  11   7   8  15  11  10  0.867692    100/100
NonOverlappingTemplate
 10   6   9  10  10  11  12   8  15   9  0.816537     99/100
NonOverlappingTemplate
  4  11   5  12  11  18  10  14  12   3  0.017912    100/100
NonOverlappingTemplate
 10   7  11  16   8   9   8  12   9  10  0.739918     99/100
NonOverlappingTemplate
 11  10  12  12   9   4  17  10  10   5  0.213309     99/100
NonOverlappingTemplate
 12  11   9  10   8   9   7  15  14   5  0.474986     99/100
NonOverlappingTemplate
  3  14   9  12  10  11  11   9  15   6  0.249284     99/100
NonOverlappingTemplate
  7  10  10  11  11  11   9   9  11  11  0.996335     97/100
NonOverlappingTemplate
```

```
 14   3   9  14  10   8   9   8  14  11  0.289667     99/100
NonOverlappingTemplate
 14  12   5   9   8  13  10  12  12   5  0.419021     98/100
NonOverlappingTemplate
 10   9  14  11   7   8  13  12   6  10  0.739918    100/100
NonOverlappingTemplate
  8  14  12  14   9   7   8   6  12  10  0.595549     98/100
NonOverlappingTemplate
 17   6   9   8  17  11   4   7  11  10  0.055361     99/100
NonOverlappingTemplate
  6  12  13  18   7  10  11   6   9   8  0.191687    100/100
NonOverlappingTemplate
 11   6  11  13   6  11   9  10  11  12  0.834308    100/100
NonOverlappingTemplate
 12   8  13  10   9  18   6   9   6   9  0.236810    100/100
NonOverlappingTemplate
  5  11   8  10   8   7   9  15  11  16  0.304126    100/100
NonOverlappingTemplate
 11  13   8   8  10  14  13   8   6   9  0.699313     99/100
NonOverlappingTemplate
 11   9   8   8  12   9  11  10  11  11  0.994250     99/100
NonOverlappingTemplate
  6   8  11   9  11   6   7  15  14  13  0.366918     98/100
NonOverlappingTemplate
 10  10   6   8  12   9  13  10  10  12  0.924076     97/100
NonOverlappingTemplate
  7  11   5   9  10  12   5  15  11  15  0.236810     99/100
NonOverlappingTemplate
 11   5  12  10  10  12  12  10   8  10  0.897763    100/100
NonOverlappingTemplate
  7  10  12   7  11   6   6   9  15  17  0.162606    100/100
NonOverlappingTemplate
 13   8   9  10  11   8  12  11  10   8  0.971699     99/100
NonOverlappingTemplate
 10   9   8  12  12  10   9  12   6  12  0.924076     99/100
NonOverlappingTemplate
  6  12   8  11  17  10   8  13   7   8  0.350485    100/100
NonOverlappingTemplate
 11  11  14   8   8  10  10  10   6  12  0.867692     98/100
NonOverlappingTemplate
  7   7  16  15   9  10  10  12   4  10  0.213309     99/100
NonOverlappingTemplate
  6   8  19  16   6  17   7   9   4   8  0.002758    100/100
NonOverlappingTemplate
```

```
 11  10  11   6   7  11  14   9  10  11  0.867692     96/100
NonOverlappingTemplate
 17   8   4  10  15   9   5   8  10  14  0.066882    100/100
NonOverlappingTemplate
 10  13   5  11   6  13  11  14   9   8  0.514124    100/100
NonOverlappingTemplate
 13  13  14   9   6   9   7  11  11   7  0.616305     99/100
NonOverlappingTemplate
 14  15   9   9   7   5  10   6  10  15  0.224821    100/100
NonOverlappingTemplate
  3   6   8  11  15   9   9  14  13  12  0.181557    100/100
NonOverlappingTemplate
 11   5  14   8   7   9  18  11   4  13  0.055361     99/100
NonOverlappingTemplate
  8   6   7  12   9  17  13   7   5  16  0.062821     99/100
NonOverlappingTemplate
  7  15  11   8  16   6   8  10   7  12  0.289667     99/100
NonOverlappingTemplate
  8  12  10  13  13   8  12   6   9   9  0.816537     99/100
NonOverlappingTemplate
  7   6   7  14   9  13  10  17  10   7  0.224821    100/100
NonOverlappingTemplate
 10   7   9  15  11  11  10   9   9   9  0.911413     99/100
NonOverlappingTemplate
  7  11   8   9  11  14  11  11  10   8  0.924076     98/100
NonOverlappingTemplate
  8   6  13  11   5  13  12  11  16   5  0.162606     98/100
NonOverlappingTemplate
 11   5   6  15  15   8  12   9   8  11  0.304126    100/100
NonOverlappingTemplate
  8   9   8   6  11  13  13   5  15  12  0.366918     99/100
NonOverlappingTemplate
  8   8   5  17   8   8  13  10  12  11  0.319084    100/100
NonOverlappingTemplate
 11  10  13  14   9   6   9  14   7   7  0.554420     99/100
NonOverlappingTemplate
 11  11   9   6   9   5  15  12   8  14  0.401199     98/100
NonOverlappingTemplate
  8   9  10  14   9  10  10  11  13   6  0.851383    100/100
NonOverlappingTemplate
 11   6  11  13  12  15  13   6   7   6  0.304126    100/100
NonOverlappingTemplate
  5  11   7  14  12  16   8   8  12   7  0.262249    100/100
NonOverlappingTemplate
```

```
  6   7   8  12  16  12  14   5   8  12  0.202268    100/100
NonOverlappingTemplate
 14  12   5   7   8   8  12  11  14   9  0.494392     99/100
NonOverlappingTemplate
 10  12  10  13  16   5  10  10   9   5  0.350485     99/100
NonOverlappingTemplate
  6   8   4   9   9  10  12  19  10  13  0.085587    100/100
NonOverlappingTemplate
 14  11   8  11   4   9   9  14   9  11  0.554420     97/100
NonOverlappingTemplate
 12  13   8   9  11  14   5  13   2  13  0.115387     98/100
NonOverlappingTemplate
  9  15   7  12   7  13   7  12   7  11  0.534146    100/100
NonOverlappingTemplate
 13  11   9   6  12   6  10   8  12  13  0.699313     98/100
NonOverlappingTemplate
  6   7   7  10   8  12  12  13  12  13  0.657933    100/100
NonOverlappingTemplate
 12  10  12   6  17   4  13   7  11   8  0.153763    100/100
NonOverlappingTemplate
 10  10  12  10   7   9   6  13  12  11  0.883171    100/100
NonOverlappingTemplate
  6   9  10  14   7  14  11  11   8  10  0.699313     98/100
NonOverlappingTemplate
  5   9  13  20   7   9   7   7  11  12  0.051942     99/100
NonOverlappingTemplate
  8  12   8  10  12  11   8   7  12  12  0.924076     96/100
NonOverlappingTemplate
  6  12  15  13  11  13   7  10   5   8  0.334538    100/100
NonOverlappingTemplate
 11   5  12   8  14  12   7   8  13  10  0.574903     99/100
NonOverlappingTemplate
  9   7  11   9   9  11  11   5  16  12  0.534146     98/100
NonOverlappingTemplate
 11  10   7   5   8   7  10  12  14  16  0.319084     98/100
NonOverlappingTemplate
 13  13  14   9   8   6   6  13  12   6  0.350485    100/100
NonOverlappingTemplate
 12   9  11  10   8  10  11   9  13   7  0.964295     98/100
NonOverlappingTemplate
 11  11   5  10  11   9  11  12   9  11  0.935716    100/100
NonOverlappingTemplate
  9   5  13  12  10   7   9  10  13  12  0.719747     99/100
NonOverlappingTemplate
```

```
  8  13   9   7  10   8  13  14  10   8  0.779188    100/100
NonOverlappingTemplate
  7   9   8  12   6  10  15  11  13   9  0.637119     99/100
NonOverlappingTemplate
 16   7  11   8  11  15   4  12  10   6  0.153763    100/100
NonOverlappingTemplate
  4  12   9   6  15  13  11   9  11  10  0.401199     98/100
NonOverlappingTemplate
  7   6   8   9  10  13  13  13  14   7  0.514124    100/100
NonOverlappingTemplate
 10   8   7  10   7  10  13  11  11  13  0.897763     98/100
NonOverlappingTemplate
  9   5   9   9   5  14  12  10  15  12  0.334538    100/100
NonOverlappingTemplate
 13   4   8   9   9  16  14  10   7  10  0.262249    100/100
NonOverlappingTemplate
 14   4   8  14  10   8   9   8  14  11  0.366918     99/100
NonOverlappingTemplate
  7  12  11   6   8  10  12  15  14   5  0.319084    100/100
OverlappingTemplate
100   0   0   0   0   0   0   0   0   0  0.000000 *    0/100  *
Universal
  7  12  11  14   8  17   8   9   8   6  0.289667    100/100
ApproximateEntropy
  2   1   2   3   3   1   2   5   0   1  0.437274     20/20
RandomExcursions
  3   1   3   3   3   1   2   0   4   0  0.437274     20/20
RandomExcursions
  1   5   1   4   2   1   2   2   1   1  0.437274     20/20
RandomExcursions
  0   0   5   2   2   0   2   4   5   0  0.025193     20/20
RandomExcursions
  3   3   3   1   3   1   2   3   1   0  0.739918     19/20
RandomExcursions
  1   5   2   2   3   1   4   0   2   0  0.213309     20/20
RandomExcursions
  2   4   3   2   1   3   3   1   1   0  0.637119     19/20
RandomExcursions
  3   1   1   1   3   4   1   3   3   0  0.534146     20/20
RandomExcursions
  2   1   1   1   4   1   0   2   4   4  0.350485     20/20
RandomExcursionsVariant
  2   1   2   2   1   3   1   4   3   1  0.834308     20/20
RandomExcursionsVariant
```

```
  2   2   1   1   2   4   3   1   2   2  0.911413     20/20
RandomExcursionsVariant
  3   1   1   2   2   3   2   2   1   3  0.964295     20/20
RandomExcursionsVariant
  2   2   0   5   3   1   1   1   4   1  0.275709     20/20
RandomExcursionsVariant
  2   2   1   2   4   2   2   1   2   2  0.964295     20/20
RandomExcursionsVariant
  2   4   3   0   4   2   2   1   2   0  0.437274     20/20
RandomExcursionsVariant
  2   3   1   5   1   2   0   1   5   0  0.090936     20/20
RandomExcursionsVariant
  1   2   4   2   4   1   1   1   2   2  0.739918     20/20
RandomExcursionsVariant
  2   4   0   3   6   0   1   1   3   0  0.035174     20/20
RandomExcursionsVariant
  2   5   3   2   3   0   2   1   2   0  0.350485     20/20
RandomExcursionsVariant
  3   2   0   4   4   1   1   0   5   0  0.066882     20/20
RandomExcursionsVariant
  4   0   1   2   4   0   4   3   0   2  0.162606     20/20
RandomExcursionsVariant
  2   3   1   1   5   2   0   1   3   2  0.437274     20/20
RandomExcursionsVariant
  2   3   0   2   4   1   2   2   3   1  0.739918     20/20
RandomExcursionsVariant
  2   1   2   4   1   1   1   8   0   0  0.002043     20/20
RandomExcursionsVariant
  1   1   3   1   3   4   0   3   2   2  0.637119     20/20
RandomExcursionsVariant
  1   0   3   1   3   1   3   4   2   2  0.637119     20/20
RandomExcursionsVariant
 11   8   9   5   9  14  11   7  15  11  0.494392    100/100
Serial
  9   2  12  13  10  13   7   7  14  13  0.162606     98/100
Serial
 13  10   6  10  15  11   6  14   6   9  0.350485    100/100
LinearComplexity


- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
- - - - - - - - -
The minimum pass rate for each statistical test with the
exception of the
random excursion (variant) test is approximately = 96 for a
```

```
sample size = 100 binary sequences.

The minimum pass rate for the random excursion (variant) test
is approximately = 18 for a sample size = 20 binary sequences.

For further guidelines construct a probability table using the
MAPLE program
provided in the addendum section of the documentation.
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
- - - - - - - - -
```

```
//Encryption Algorithm
#include <iostream>
#include <stdlib.h>
#include <time.h>
#include <fstream>
using namespace std;


int main()
     {
          int n;
          int t;
          int r;
          int c;
          int d;
          int o;
          int ave = 128;
          srand(time(NULL));
          cout << "What is the number of input bits? ";
          cin >> n;
          int data [n];
          int inbits [n];
          int outbits [n];
          ifstream myfile;
          ifstream myfile2;
          myfile.open("bin2dec06202017Diehard.txt");
          myfile2.open("inputBitsThesis.txt");
```

```
               cout << "The bits that will be converted " <<
endl << endl;
                    for (int l=0; l <= n; l++)
                         {
                              myfile >> data[l]; //data is an
array that contains radioactive event times
                         }
          myfile.close();
                    for (int p=0; p <= n; p++)
                         {
                              myfile2 >> inbits [p]; //inbits is
an array that contains the input bits
                         }
                    cout << "The first bit is a " << inbits [0]
<< endl << endl;

          const char* output_file_name = "rand_bits.out";
          const char* output_file_name2 =
"Thesis10000encryptbits.out";

          ofstream my_out(output_file_name);
          ofstream my_out2(output_file_name2);
          if   (my_out.fail())     {
          cerr << "Unable to open the file " << output_file_name
                    <<  "for writing " << endl;
          }
          if   (my_out2.fail())    {
          cerr << "Unable to open the file " << output_file_name
                    <<  "for writing " << endl;
          }


c = data [rand()%8962];
     cout << "The first random time is: " << c << endl << endl;
               //This for loop converts the random bits into
random numbers depending on c.
               if ( inbits [0] == 0 && c<=ave) {

for ( int first =0; first <= n; first++) {
     if (inbits [first] == 0) {
     outbits [first] =rand()%449+100;
     }
     else {
     outbits [first] = rand()%449+550;
          }
```

```
}
               }
               if ( inbits [0] == 0 && c>ave) {

for ( int second = 0; second <= n; second++) {

     if (inbits [second] == 0) {
     outbits [second] = rand()%449+550;
     }
     else {

     outbits [second] = rand()%449+100;
          }
}
               }
               if ( inbits [0] == 1 && c<=ave) {

for ( int third = 0; third <= n; third++) {

     if (inbits [third] == 0) {
     outbits [third] = rand()%449+550;
     }
     else {

     outbits [third] = rand()%449+100;
          }
}
               }
               if ( inbits [0] == 1 && c>ave){

for ( int fourth = 0; fourth <= n; fourth++) {
     if (inbits [fourth] == 0) {

     outbits [fourth] = rand()%449+100;
     }
     else {
     outbits [fourth] = rand()%449+550;
          }
}
               }


               for ( int m=0; m<= n; m++)  { //This for loop
displays the converted bits to the screen.
                         my_out2 << outbits [m]  << endl;
```

```
                }
            cout << endl;

    return 0;
}
```