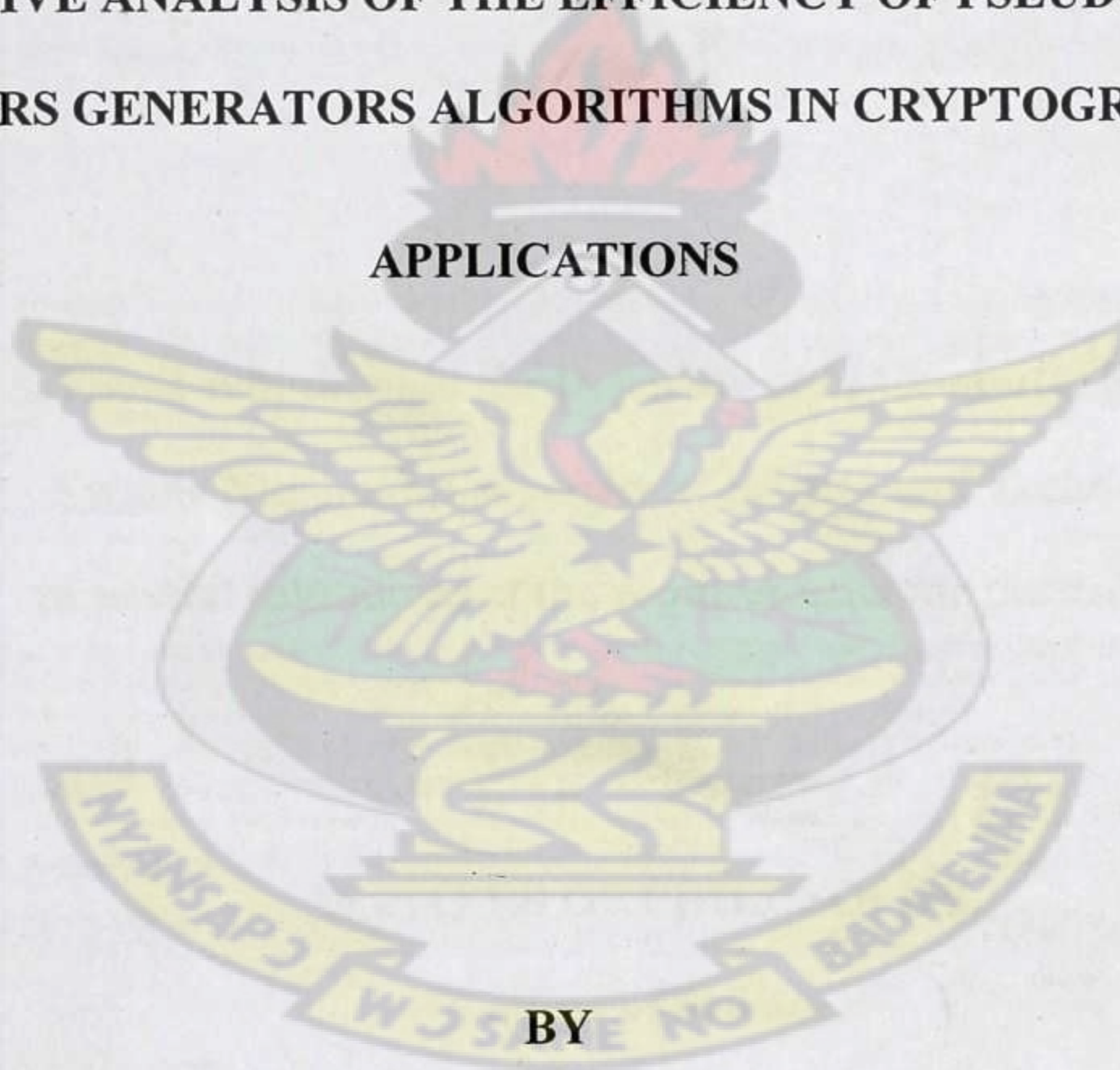


KWAME NKRUMAH UNIVERSITY OF SCIENCE AND TECHNOLOGY

KNUST

**COMPARATIVE ANALYSIS OF THE EFFICIENCY OF PSEUDO RANDOM
NUMBERS GENERATORS ALGORITHMS IN CRYPTOGRAPHIC
APPLICATIONS**



ABILIMI AYAABA CHRISTOPHER

NOVEMBER, 2012

**COMPARATIVE ANALYSIS OF THE EFFICIENCY OF PSEUDO RANDOM
NUMBERS GENERATORS ALGORITHMS IN CRYPTOGRAPHIC
APPLICATIONS**

By

KNUST

Abilimi Ayaaba Christopher (BSc.)

**A Thesis submitted to the Department of Computer Science,
Kwame Nkrumah University of Science and Technology
in partial fulfillment of the requirements for the degree of**

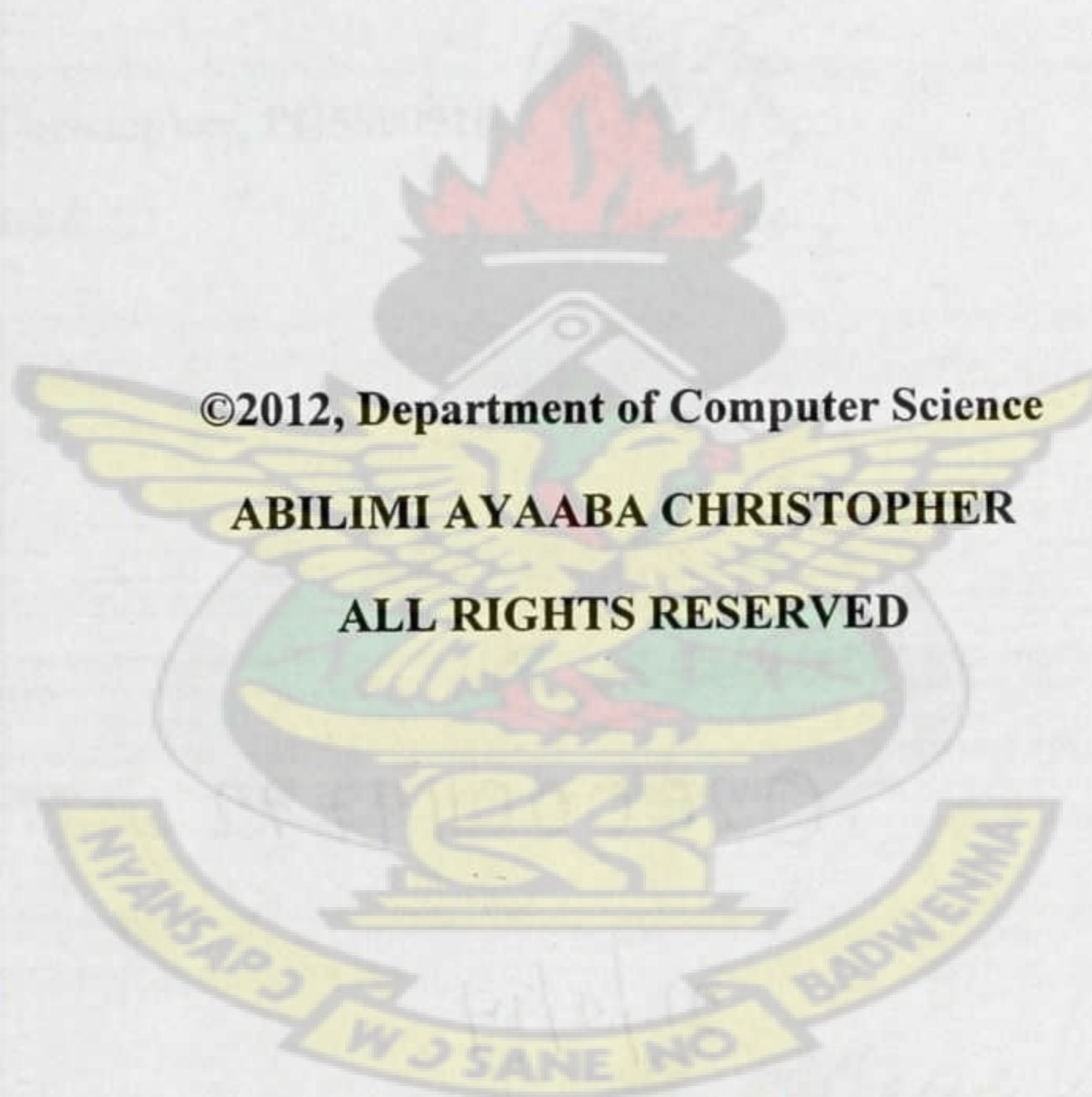
MASTERS OF PHILOSOPHY

College of Science

November 2012

Copyright

KNUST



©2012, Department of Computer Science

ABILIMI AYAABA CHRISTOPHER

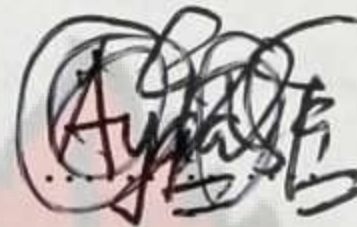
ALL RIGHTS RESERVED

Certification

I hereby declare that this submission is my own work towards the Masters and that, to the best of my knowledge, it contains no material previously published by another person nor material which has been accepted for the award of any other degree of the University, except where due acknowledgment has been made in the text.

Abilimi A. Christopher, PG5090910

Student Name & ID



Signature

17-04-13

Date

Certify by:

Dr. M. Asante

First Supervisor



Signature

24/04/13

Date

Certify by:

Dr. J. B. Hayfron-Acquah

Second Supervisor



Signature

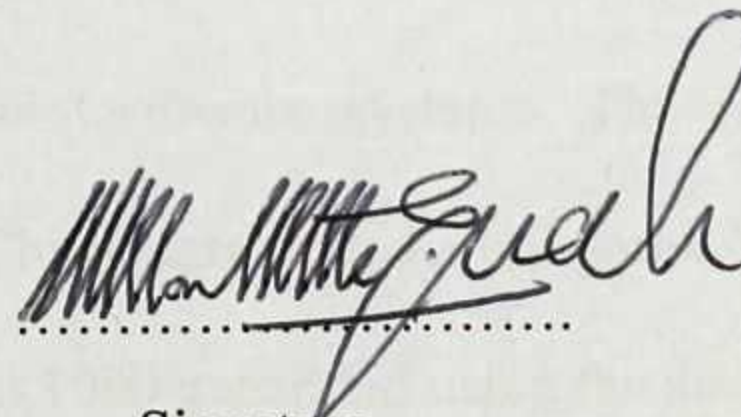
26/4/13

Date

Certify by:

Dr. J. B. Hayfron-Acquah

Head of Department



Signature

26/4/13

Date

Abstract

The importance of information security management systems is well recognized by the information security literature. The goals of information security are protecting the confidentiality, integrity and availability of information. Information security is concerned with the confidentiality, integrity and availability of data regardless of the form the data may take: electronic, print, or other forms. Cryptography is a science of protecting information by encoding it into an unreadable format. It is an effective way of protecting sensitive information as it is stored on media or transmitted through network communication paths. Random Numbers determines the security level of Cryptographic Applications as they are used to generate padding schemes in the encryption and decryption algorithms as well as used to generate cryptographic keys. The more randomness in the numbers a generator produces, the more effective the Cryptographic Algorithms and the more secured it is for protecting confidential data. Sometimes developers finds it difficult to be able determine which Random Numbers Generators (RNGs) can provide a much secured Cryptographic Systems for secured enterprise application implementations. This research aims to find an effective Pseudo Random Number Generator algorithm among Fibonacci Random Numbers Generator Algorithms, Gaussian Random Numbers Generator Algorithm, Specific Range Random Numbers Generator Algorithms, and Secure Random numbers Generators, which the most common Pseudo Random Numbers Generators Algorithms, that can be used to improve the security of Cryptographic software systems. The researcher employed statistical tests like Frequency test, Chi-Square test, Kolmogorov-Smirnov test on the first 100 random numbers between 0 to 1000 generated using the above generators.

Table of Contents

Contents	Pages
Copyright	ii
Certification	iii
Abstract	iv
Table of Contents	v
List of Tables	vii
List of Figures	viii
List of Abbreviations	ix
Acknowledgement	xi
CHAPTER ONE:INTRODUCTION	1
1.1 Introduction	1
1.2 Problem Statement	4
1.3 Justification of the study	5
1.4 The Objectives of the Study	5
1.4.1 General Objectives	5
1.4.2 Specific Objectives	5
1.5 Significance of the research	6
1.6 Scope of the study	6
1.7 Research questions	7
1.8 Limitation and Delimitation of the Study	7
1.8.1 Limitation of the study	7
1.8.2 Delimitation of the study	8
1.9 Outline of the Thesis	8
CHAPTER TWO:LITERATURE REVIEW	9
2.1 Introduction	9
2.2 Cryptography	10
2.3 Rivest Shamir Adleman (RSA) cryptography	12
2.3.1 RSA Cryptosystem	12
2.3.2 Chinese Remainder Theorem (CRT) Based RSA	14
2.4 Attacks on the CRT-Based RSA	16
2.4.1 Fault Attack and the Existing Countermeasures	16
2.4.2 Timing Attack	20
2.4.3 Power Attack	21
2.5 Random Number and Random Bit Generators	21
2.6 Categories of Random Numbers Generators	25
2.7 Properties of Good PRNGs	28
2.8 Conclusion	30
CHAPTER THREE:RESEARCH METHODOLOGY	31
3.1 Introduction	31
3.2 Test for Random Number Generators	31
3.2.1 Tests for Randomness of Random Number Generators (RNGs).	31
3.2.2 Chi-square Test	32
3.2.3 Kolmogorov-Smirnov test (KS-test)	35
3.3 Frequency Test of Random Numbers Generators	37
3.4. Conclusion	38
CHAPTER FOUR:RESULTS AND DISCUSSIONS	39
4. 1 Introduction	39
4.2 Frequency Test of Random Numbers Generators	39

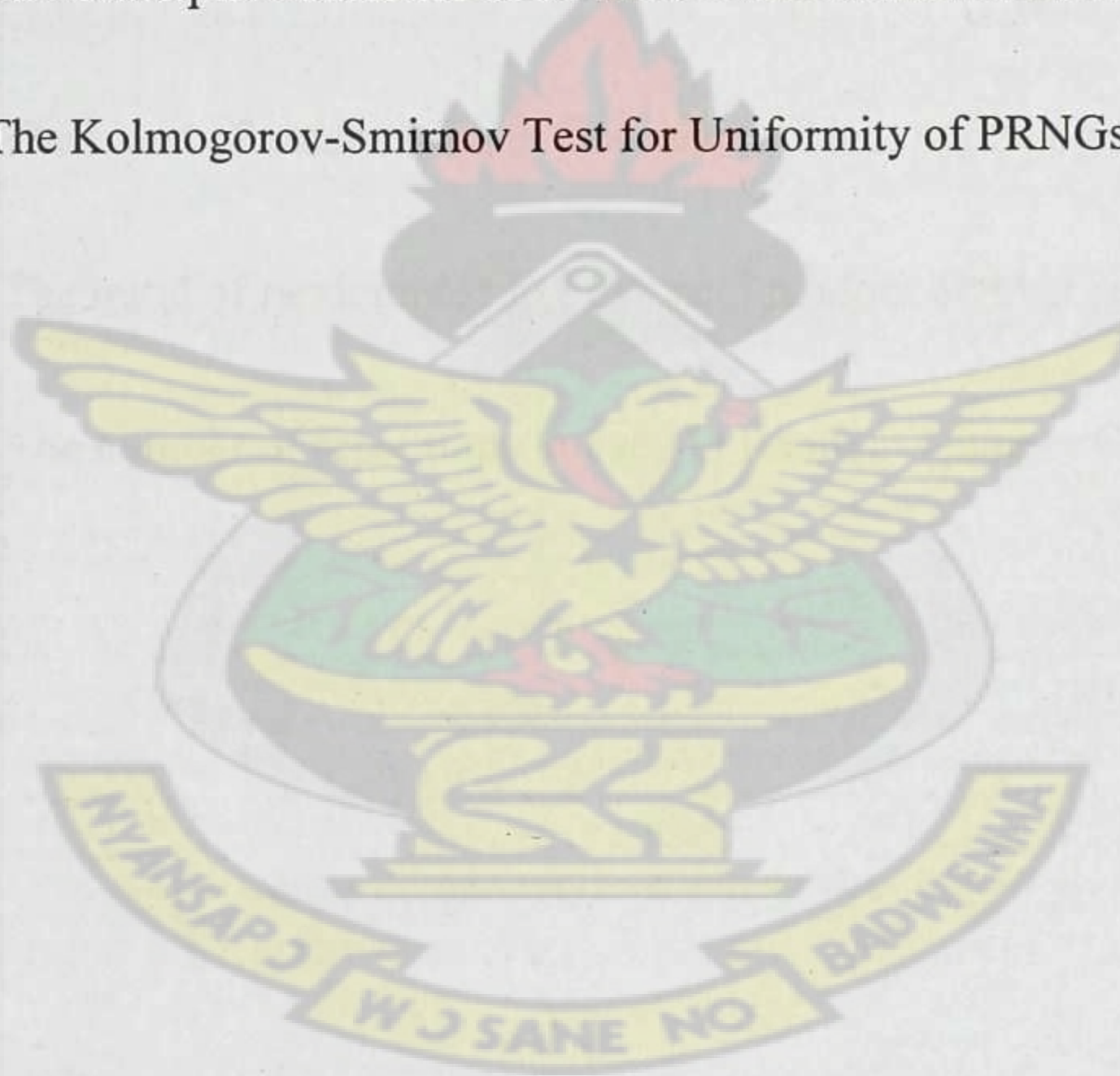
4.3 The Chi-Square Test for Uniformity of random numbers.....	44
4.4 The Kolmogorov-Smirnov Test for Uniformity of random numbers.....	46
4.5 Summary of Findings and Discussions.....	48
4.6 Conclusion.....	50
CHAPTER FIVE:CONCLUSIONS AND RECOMMENDATIONS.....	51
5.1 Conclusions	51
5.2 Recommendations.....	52
References	54
Appendices	62

KNUST



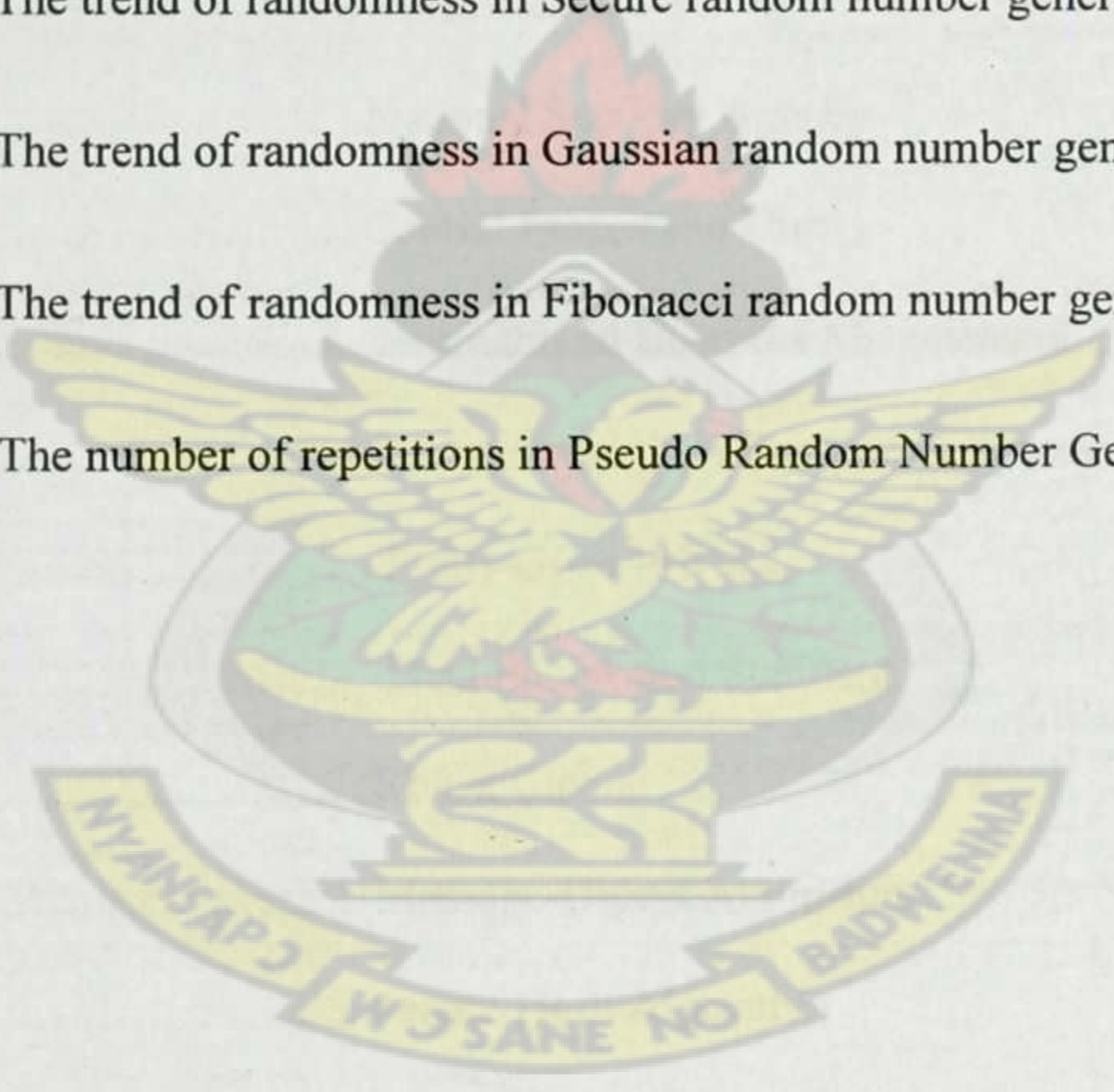
List of Tables

Table 3.1: The Chi-Square test results for PRNGs.....	34
Table 3.2: The descriptive Statistics for four random Generators.....	36
Table 3.3: The Kolmogorov-Smirnov Test for Uniformity of Random numbers...	37
Table 4.1: The Chi-Square Test for Independence of PRNGs.....	45
Table 4.2: The descriptive statistics of PRNGs.....	46
Table 4.3: The Kolmogorov-Smirnov Test for Uniformity of PRNGs.....	47



List of Figures

Figure 2.1: The process of hardware/software fault attack.....	18
Figure 2.2: Pseudo Random Numbers.....	27
Figure 2.3: Quasi Random Numbers.....	27
Figure 4.1: The trend of randomness in Specific range random number generator....	40
Figure 4.2: The trend of randomness in Secure random number generator.....	41
Figure 4.3: The trend of randomness in Gaussian random number generator.....	42
Figure 4.4: The trend of randomness in Fibonacci random number generator.....	43
Figure 4.5: The number of repetitions in Pseudo Random Number Generators.....	44



List of Abbreviations

AES.....	Advanced Encryption Standard
FIPS.....	Federal Information Processing Standard
PCI DSS.....	Payment Card Industry Data Security Standard
TDE.....	Transparent Data Encryption
ATM.....	Automatic Teller Machine
WWI.....	World War I
WWII.....	World War II
NSA.....	National Security Agency
CPU.....	Central Processing Unit
IBM.....	International Business Management
IDEA.....	International Data Encryption Algorithm
PGP.....	Pretty Good Privacy
DES.....	Data Encryption Standard
RC.....	Rivest Cipher
MD.....	Message-Digest Algorithm
SHA.....	Secure Hash Algorithm
RSA.....	Revest- Shamir- Adleman
NIST.....	National Institute of Standards and Technology
GCHQ.....	Government Communication Headquarters
SSL.....	Secure Socket Layer
TLS.....	Transport Layer Security
DSA.....	Digital Signature Algorithm

ECC.....elliptic curve cryptography

PIN.....Personal Identification Number

CPA.....Chosen-Plaintext Attack

IMAP.....Internet Message Access Protocol

POP.....Post Office Protocol

S/MIME.....Secure/Multipurpose Internet Mail Extension

DRM.....digital rights management

DMCA.....Digital Millennium Copyright Act

HD DVD.....High-Definition/Density Digital Versatile Disk

PKCS.....Public-Key Cryptography Standards

ISAAC.....Indirection, Shift, Accumulate, Add, and Count.

ISO.....International Standard Organisation

IKE.....Internet Key Exchange

RNG.....Random Number Generators

PRNGs.....Pseudo Random Numbers Generators

ANSI.....American National Standards Institute

MAC.....Media Access Control

IEEE.....Institute of Electrical and Electronic Engineers

IETF.....Internet Engineering Task Force

IPS.....Instruction Prevention Systems

IPsec.....Internet Protocol Security

Acknowledgement

During the past two years, I have had the privilege to be taught by Lecturers and assisted by Non-teaching staff in the Department of Computer Science who have given me their friendship, professional advice, and helped me to grow as a researcher.

I would first like to thank my Supervisor, Dr. M. Asante. He has been an ideal Supervisor, allowing and encouraging me to investigate topics of my choice. His unrelenting support of my research and guidance in its dissemination has been instrumental in any success I can claim as a graduate student.

Dr. J. B. Hayfron-Acquah, my Co-supervisor and Head, Computer Science Department, I would like to thank for his assistance and encouragement to do this research. His support has been crucial in this thesis.

I would like to thank my fellow graduate students in the Computer Science Department that have helped me with anything from encryption, to software security. To all the non-teaching staff in the Department who has supported me for anything concerning this research, I say thank you.

Finally, I would like to thank my family and Church: without their loving support, understanding and prayers throughout the years I would not have been able to complete the program and for this, I thank you.

CHAPTER ONE

INTRODUCTION

1.1 Introduction

According to Gary C. Kessler (2012), in the world of information security, we often see statements such as 'secured by 128-bit AES' or 'protected by 2048 bit authentication'. We are used to people asking about the strength of the cryptographic algorithms deployed in a security solution. Algorithms such as the AES, RSA and ECC have a proven track record of being difficult to break. They are successfully deployed in protocols that protect our identity and the integrity and confidentiality of our data, on a daily basis. Consider, for instance, the use of SSL or TLS when you buy a book at Amazon, or when you connect to your bank account to transfer a sum of money (Fisnik Hasani, 2011). Or the use of IKE and IPsec when you connect your laptop to the company network to check on your email and read documents stored on the company network.

AuthenTec Embedded Security Solutions (2010) stated that what we see very rarely, unfortunately, are statements about the strength of the random number generator used by a security system. System designers are typically more concerned with the power consumption and bit generation speed, than with the actual randomness of the bits generated.

This is strange, considering that in most, if not all, cryptographic systems, the quality of the random numbers used directly determines the security strength of the

system. In other words, the quality of the random number generator directly influences how difficult it is to attack the system (Thomas B., 2006).

This can be easily seen if you realize that modern cryptographic algorithms and protocols are designed around a well-known principle by Kerckhoff, which roughly translates into the statement that *"The security of the system must depend solely on the key material, and not on the design of the system"* (AuthenTec Embedded Security Solutions, 2010). This means that all modern security algorithms and protocols have their 'cryptographic strength' expressed in the number of keys (bits) that an attacker needs to guess before he can break the system. This expression of strength implicitly assumes that the attacker has no knowledge of the bits of the original key used. The 'effective strength' of an algorithm is diminished when better attacks against it are found and more key bits can be derived from looking at (a limited amount) of output data.

Take, for example, the effective strength of 3DES. Although it uses 3 keys of 56 bits each, 3DES is currently only expected to provide 112 bits of 'effective' security, since the best attack against it today has a complexity of 2^{112} bits. This still assumes that all 168 bits of the key used, are unknown and unpredictable by the attacker. So what happens if we start with key material that is partly predictable to the attacker? Immediately, the security of the system is weakened, regardless of the algorithm or protocol used. If your 128-bit key contains 16 predictable bits, using it in AES-128 doesn't give you 128-bit protection; it only gives you 112 bits of protection, making the security of your system 'only' as strong as 3DES today.

And it doesn't stop with cryptographic key material. Many security protocols require random bits to remain secure, even though the protocol definition will not always call it random; typically, a protocol description will use the term

'unpredictable' to indicate that a certain value should be difficult to guess by an attacker. True random numbers may be required if your application uses one of the following:

- Keys and initialization values (IVs) for encryption;
- Keys for keyed MAC algorithms;
- Private keys for digital signature algorithms;
- Values to be used in entity authentication mechanisms;
- Values to be used in key establishment protocols;
- PIN and password generation;
- Nonces.

Exactly for the reasons mentioned above, the IETF has written a 'Best Practices' document (RFC 4086 (Eastlake, et al, 2005)) to explain the importance of true randomness in cryptography, and to provide guidance on how to produce random numbers. NIST has a section on Random Number Generation in their Cryptographic Toolbox pages, and a number of standards bodies such as IETF, IEEE, NIST, ANSI, and ISO have, or are working on, standards related to random number generation. This goes to show the importance of proper random number generation. This research therefore compares the effectiveness of Pseudo Random Numbers Generators so as to get a Secured Cryptographic System, when implemented in Cryptography.

1.2 Problem Statement

Software development and engineering is one of the fastest growing areas in the field of computer science, and particularly in software development industry (John, H., 2006). As such more and more enterprises prefer the software systems of storing their confidential information to the manual information system. This ensures the confidentiality, integrity and availability of data regardless of the form the data may take: electronic, print, or other forms.

Protecting confidential information is a business requirement, and in many cases also an ethical and legal requirement. This calls for the choice of a better security tool in the software development stage. Cryptography is effectively used to assure secrecy. Wax seals, signatures, and other physical mechanisms were typically used to assure integrity of the media and authenticity of the sender. With the advent of electronic funds transfer, the applications of cryptography for integrity began to surpass its use for secrecy. A secure-cryptographic application requires a secure encryption and decryption keys algorithms, and this needs a good random number generator to be associated with the algorithm (Elaine B. & Allen R., 2011). A good random numbers generator produces a sequence of numbers that cannot be easily guessed or determined by an adversary (Yehuda L., 2006)

It is for such reason that the researcher has thought it wise to compare the effectiveness of the Pseudo-Random Numbers Generators that can be used in to improve security of Cryptographic algorithms in the Software Industry.

1.3 Justification of the study

Data thefts as result of unauthorized access to data that leads to violations of both data protection laws and business or ethical requirement of an Organization (Robert S. & Corey C, 2009). These losses in data integrity, can lead to the loss of billions of Ghanaian Cedi which inflates budgets for organizations.

Therefore this research addresses this crucial issue by investigating the effectiveness of Random Numbers Generators that should be used on a cryptographic application in order to enhance security of data in organizations.

1.4 The Objectives of the Study

1.4.1 General Objectives

The general objective is finding effective Pseudo Random Number Generator algorithms that can be used to improve the security of Cryptographic software systems.

1.4.2 Specific Objectives

The specific objectives of the study are:

- I. To test for the repetition of Pseudo-Random Numbers Generators.
- II. To test for the uniformity of a Pseudo-Random Numbers Generators (PRNGs) algorithms.
- III. To test for independence of numbers generated by the Pseudo-Random Numbers Generators algorithms.

Therefore the researcher based on the specific objectives will be able to identify a pseudo random number generator with lesser repetition of numbers,

unevenly distribution numbers and highest independence of numbers produced by the generator used in the research and hence that generator is established as the effective pseudo random numbers generator which can be used in cryptographic applications to improve data security.

1.5 Significance of the research

According to Elizabeth A. Smith (2001) a highly valued asset is loss when companies lose their important and confidential data in enterprise applications software. As such the security of the Cryptographic System is needed to be transparent or high for Companies for key exchange, digital signatures, or encryption of small blocks of data.

However, the randomness of the keys provide data or information security in cryptographic applications, which protects data access by unauthorized users, and that depends on the independence of numbers, lesser repetition of numbers and unevenly distribution of numbers in the PRNGs algorithms used (M. Jayakumar & T. Christopher, 2012). This research is done to identify those factors that determine an effective pseudo random numbers generator so as to enhance the security of cryptographic applications when applied in the security process.

1.6 Scope of the study

The area of the research is limited to Pseudo-Random Numbers Generators, and it is concentrated on Cryptographic Application Systems, and also applied to Organisations that use enterprise software applications. These gives a logistic problem of data being accessed and used by unauthorized users or by attackers based

on weaker cryptographic algorithms, hence the reason for the researcher choice of the sector.

1.7 Research questions

The researcher in an attempt to draw valid conclusion will find answers to the following questions:

1. How can recurrence of numbers in PRNGs algorithms be determined?
2. How is the efficiency of PRNGs algorithm determined?
3. How can the uniformity of PRNGs algorithm be determined?
4. How can the dependency of numbers in PRNGs algorithms affect its efficiency?

1.8 Limitation and Delimitation of the Study

1.8.1 Limitation of the study

Cryptographic Systems use both Probabilistic (True) and Deterministic (Pseudo) Random Numbers Generators algorithms to generate keys and algorithms that are used to protect data from attackers. Unfortunately the researcher concentrates on PRNGs algorithms for Cryptographic Systems.

The implementation of the information security management using Probabilistic Random Numbers Generators involves a lot of complex technology which cannot be afforded by the researcher.

1.8.2 Delimitation of the study

This research only covers the four Pseudo-Random Numbers Generators algorithms that can be applied in information security for data protection. The researcher used four Pseudo Random Numbers Generators for the thesis.

There are also many statistical tests for the effectiveness and efficiency of Random Numbers Generators algorithms. However the researcher only employed frequency test, Chi-Square test and Kolmogorov-Smirnov test.

1.9 Outline of the Thesis

This thesis starts with an Introduction to the research background, stating the Research Problem, Objective as well as describing the Scope of Study. The second Chapter looks at the literature that was reviewed to come with relevant information that supported the research objective and methods of carrying out the research. Methods of carrying out the research are covered in the third Chapter. Results of the research are presented in the fourth Chapter. The results were analyzed and discussed in the fifth Chapter of the thesis. The thesis ends with general conclusions and recommendations made based on the research results.

CHAPTER TWO

LITERATURE REVIEW

2.1 Introduction

A Good Cryptography requires good random numbers (H. Mike et al, 2012). This thesis evaluates software security based on Pseudo Random Number Generator (PRNG) for use in Cryptographic Applications.

Almost all cryptographic protocols require the generation and use of secret values that must be unknown to attackers. For example, random numbers generators are required to generate public/private key pairs for asymmetric (public key) algorithms including RSA, DSA, and Diffie-Hellman. Keys for symmetric and hybrid cryptosystems are also generated randomly. RNGs are used to create challenges, nonces (salts), padding bytes, and blinding values. This is because security protocols rely on the unpredictability of the keys they use; random number generators for cryptographic applications must meet stringent requirements. The most important property is that attackers, including those who know the RNG design, must not be able to make any useful predictions about the RNG outputs. In particular, the apparent entropy of the RNG output should be as close as possible to the bit length. The thesis compare the efficiency of different random numbers generators in Cryptographic Algorithm to see which them is the best to be used for cryptography.

This Chapter gives some background materials about Cryptography and its applications, RSA cryptosystems, literature pertaining to the RSA cryptography and the different types of Random Numbers Generators (RNGs), and qualities of good

RNGs used in Cryptographic Applications and specifically the RSA Cryptographic Algorithm.

2.2 Cryptography

According to A. Kapadia , P. Tsang, and S. W. Smith (2005), Cryptography is the practice and study of hiding information. Modern cryptography intersects the disciplines of mathematics, computer science, and electrical engineering. Applications of cryptography include Automatic Teller Machine (ATM) cards, computer passwords, and electronic commerce.

Cryptology prior to the modern age was almost synonymous with *encryption*, the conversion of information from a readable state to apparent unpredictable. The sender retained the ability to decrypt the information and therefore avoid unwanted persons being able to read it. Since World War I (WWI) and the advent of the computer, the methods used to carry out cryptology have become increasingly complex and its application more widespread (Yuval,I., 2011).

Mohammed, A., & Annapurna, P., P. (2012, February) stated that modern cryptography follows a strongly scientific approach, and designs cryptographic algorithms around computational hardness assumptions that are assumed hard to break by an adversary. Such systems are not unbreakable in theory but it is infeasible to do so for any ~~practical~~ adversary. Information-theoretically secure schemes that provably cannot be broken exist but they are less practical than computationally-secure mechanisms. An example of such systems is the one-time pad. Alongside the advancement in cryptology-related technology, the practice has raised a number of legal issues, some of which remain unresolved.

Until modern times cryptography referred almost exclusively to *encryption*, which is the process of converting ordinary information (called plaintext) into incomprehensible code (called ciphertext)(David Kahn, 2007). Decryption is the reverse, in other words, moving from the unintelligible ciphertext back to plaintext. A *cipher* (or *cypher*) is a pair of algorithms that create the encryption and the reversing decryption. The detailed operation of a cipher is controlled both by the algorithm and in each instance by a *key*. This is a secret parameter (ideally known only to the communicants) for a specific message exchange context.

According to David, K. (2007), a "cryptosystem" is the ordered list of elements of finite possible plaintexts, finite possible cyphertexts, finite possible keys, and the encryption and decryption algorithms which correspond to each key. Keys are important, as ciphers without variable keys can be trivially broken with only the knowledge of the cipher used and are therefore useless (or even counter-productive) for most purposes. Historically, ciphers were often used directly for encryption or decryption without additional procedures such as authentication or integrity checks.

In colloquial use, the term "code" is often used to mean any method of encryption or concealment of meaning. However, in cryptography, *code* has a more specific meaning. It means the replacement of a unit of plaintext (that is, a meaningful word or phrase) with a code word (for example, wallaby replaces attack at dawn). Codes are no longer used in serious cryptography—except incidentally for such things as unit designations (example: Bronco Flight or Operation Overlord)—since properly chosen ciphers are both more practical and more secure than even the best codes and also are better adapted to computers. Cryptanalysis is the term used for the study of methods for obtaining the meaning of encrypted information without

access to the key normally required to do so; that is, it is the study of how to crack encryption algorithms or their implementations.

Some use the terms *cryptography* and *cryptology* interchangeably in English, while others (including US military practice generally) use *cryptography* to refer specifically to the use and practice of cryptographic techniques and *cryptology* to refer to the combined study of cryptography and cryptanalysis (Oded Goldreich, 2008). English is more flexible than several other languages in which *cryptology* (done by cryptologists) is always used in the second sense above. In the English Wikipedia the general term used for the entire field is *cryptography* (done by cryptographers). The study of characteristics of languages which have some application in cryptography (or cryptology), that is frequency data, letter combinations, universal patterns, and so on, is called cryptolinguistics.

2.3 Rivest Shamir Adleman (RSA) cryptography

2.3.1 RSA Cryptosystem

RSA is an algorithm for public-key cryptography that is based on the presumed difficulty of factoring large integers, the factoring problem. RSA stands for Ron Rivest, Adi Shamir and Leonard Adleman, who first publicly described it in 1977. Clifford Cocks, an English mathematician, had developed an equivalent system in 1973, but it was classified until 1997. A user of RSA creates and then publishes the product of two large prime numbers, along with an auxiliary value, as their public key. The prime factors must be kept secret. Anyone can use the public key to encrypt a message, but with currently published methods, if the public key is large enough, only someone with knowledge of the prime factors can feasibly decode the message.

The basic RSA cryptosystem has two public quantities referred to as n (modulus) and e (public key), as well as private quantities d (private key) and $\lambda(n)$.

$\lambda(n)$ is defined as the Least Common Multiple (LCM) of all the prime factors of n .

The secret exponent d is chosen as an integer smaller than $\lambda(n)$ and relatively prime to $\lambda(n)$. The public key e is the “multiplicative inverse” of d and can be calculated as $d = e^{-1} \bmod \lambda(n)$.

There are two processes in the RSA cryptosystem, one is encryption/decryption and the other is signing/signature-verification process (Kaliski & B, 1992). They stated in their Research that before the message is encrypted or signed, it is split into several blocks (the message to be sent): m_1, m_2, \dots, m_j ($m_k < n$ for $k \in [1, j]$) with the same wordlength in the case it has larger wordlength than the modulus n . During the encryption/decryption process, the public key e is used to encrypt the message m as $c = m^e \bmod n$, and the secret key d is used to recover the message m from the encrypted information c as $m = c^d \bmod n$. In the signing/signature-verification process, the secret key d is used to obtain the signature s from the message m by using $s = m^d \bmod n$, and the public key e is used to verify the signature s by checking whether $s^e \bmod n$ equals to the message m .

The public quantity n of the two-prime RSA cryptosystem has two large prime factors referred to as p and q respectively such that $n = p \cdot q$. The two-prime RSA also has another public quantity e and the secret quantities d and $\lambda(n)$. These two positive integers p and q are usually chosen to have similar wordlength. Public quantities $\{n, e\}$ are made public and $\{p, q, \lambda(n), d\}$ are kept private in the two-prime RSA cryptosystem.

For the multi-prime RSA cryptosystem, the public modulus n has at least three prime factors (Robert Griffin, 2009). Usually the first three prime numbers are represented as p, q and r , so that $n = \sum_{k=1}^j i_k = p \cdot q \cdot r \cdots i_j$. Similarly, $\{n, e\}$ are made public and $\{p, q, r, \cdots i_j, \lambda(n), d\}$ are kept private (RSA Security Inc., 2000) in multi-prime cryptosystems. One of the typical cases of the multi-prime RSA cryptosystem is the three-prime RSA, in which the modulus has three prime factors p, q and r . The prime numbers p, q and r are prime numbers and need to be factorised using Chinese Remainder Theorem (CRT).

2.3.2 Chinese Remainder Theorem (CRT) Based RSA

The Chinese Remainder Theorem (CRT) can be described as follows (L. R. YU, 2002). First, we assume the number $n = \sum_{k=1}^j n_k$ and x_1, x_2, \dots, x_j are positive integers, where n_1, n_2, \dots, n_j are also positive integers and relatively prime to each other, i.e.

$\gcd(n_i, n_k) = 1$ for any $i, k \in [1, j]$ when i does not equal to k . Then, the system of congruencies

$$x \equiv x_1 \pmod{n_1}$$

$$x \equiv x_2 \pmod{n_2}$$

...

$$x \equiv x_k \pmod{n_k} \quad (k=3, \dots, j)$$

has a simultaneous solution x . x can be calculated as:

$$x = \left(\sum_{k=1}^j x_k \cdot r_k \cdot s_k \right) \pmod{n}$$

where $r_k = \frac{n}{n_k}$ and $s_k = r_k^{-1} \pmod{n_k}$ for all $k=1, \dots, j$.

The CRT can be used to speed up the decryption and signing process in two-prime or multi-prime RSA (J.-J Quisquater & C. Couvreur, 1982), (RSA Security Inc., 2000).

The RSA systems that use the CRT to speed up the calculations are called CRT-based RSA. They stated in this paper that RSA is the most widely deployed public key cryptosystem. It is used for securing web traffic, e-mail, and some wireless devices. Since RSA is based on arithmetic modulo large numbers it can be slow in constrained environments. For example, 1024-bit RSA decryption on a small handheld device such as the Palm III can take as long as 40 seconds. Similarly, on a heavily loaded web server, RSA decryption significantly reduces the number of SSL

requests per second that the server can handle. Typically, one improves RSA's performance using special-purpose hardware. Current RSA coprocessors can perform as many as 10,000 RSA decryptions per second (using a 1024-bit modulus) and even faster processors are coming out.

2.4 Attacks on the CRT-Based RSA

The attack on RSA cryptosystems is the science of breaking the encoded data. The attacks toward the smart Integrated Circuit (IC) card device of the RSA cryptosystem can be classified into two basic categories as the traditional mathematical attacks and the implementation attacks (Dan Boneh, 2000). The traditional mathematical attacks are algorithms modeled as ideal mathematical objects. Attacks of this kind are typically generalized and mostly theoretical rather than operational. The physical implementation attacks strategies are always specific instead of generalized (Dan Boneh, 2000). The vulnerabilities of the implementation attacks are relatively more difficult to control and they have been historically used to crash the cryptosystems (Peter G. Neumann, 1998). Thus, the study of this thesis is concentrated on the implementation attacks.

2.4.1 Fault Attack and the Countermeasures

Bell laboratories discovered that all tamperproof devices of cryptosystems, which use public key cryptography for user authentication without special countermeasure, are at the risk of the occurrence of hardware faults (Bell

Communications research, 1995). For example, smart cards that are used for data storage, cards that personalize cellular phones, cards that generate digital signatures or authenticate users for remote login to corporate networks are all vulnerable to this attack.

The hardware fault attack is that the adversary induces some type of fault into the devices so that the system will have erroneous responses or produce faulty results. Then the adversary is able to obtain the secret information of the system using the erroneous responses or results from the system. The hardware fault attack of the cryptosystem is composed of two steps. The first step is to inject some fault into the system at appropriate time. The second step is to exploit the erroneous responses or results to obtain the secret information of the cryptosystem. The process of the fault-based attack is shown in Figure 2. 1. The success of the hardware fault attack depends on whether the following three conditions are met or not (D. Boneh, et al, 2001):

- (i). The message to be signed is known to the attacker.
- (ii). A random fault occurs during the system calculations.
- (iii). The faulty results or erroneous responses are sent out of the system.

Guaranteeing that one or more of the above three conditions is not met is one way to protect the RSA devices against such attack. Concerning the first condition, some countermeasures have been proposed to make sure the attacker has no access to the message to be signed. The Full Domain Hash (FDH) (IEEE standard 1363-2000, 2000) and Probabilistic Signature scheme (PSS) protocols (RSA Security Inc., 2000) are two of these countermeasures that have been standardized. In both FDH and PSS schemes, an original message m is converted to a hash value $mHash$ by applying a one-way hash function¹ to the message m . Then the hash value $mHash$ is transformed into an encoded message EM . Finally the signature s is generated from the encoded message EM using the private key. Therefore, the attacker cannot access the encoded message EM to factor the system.

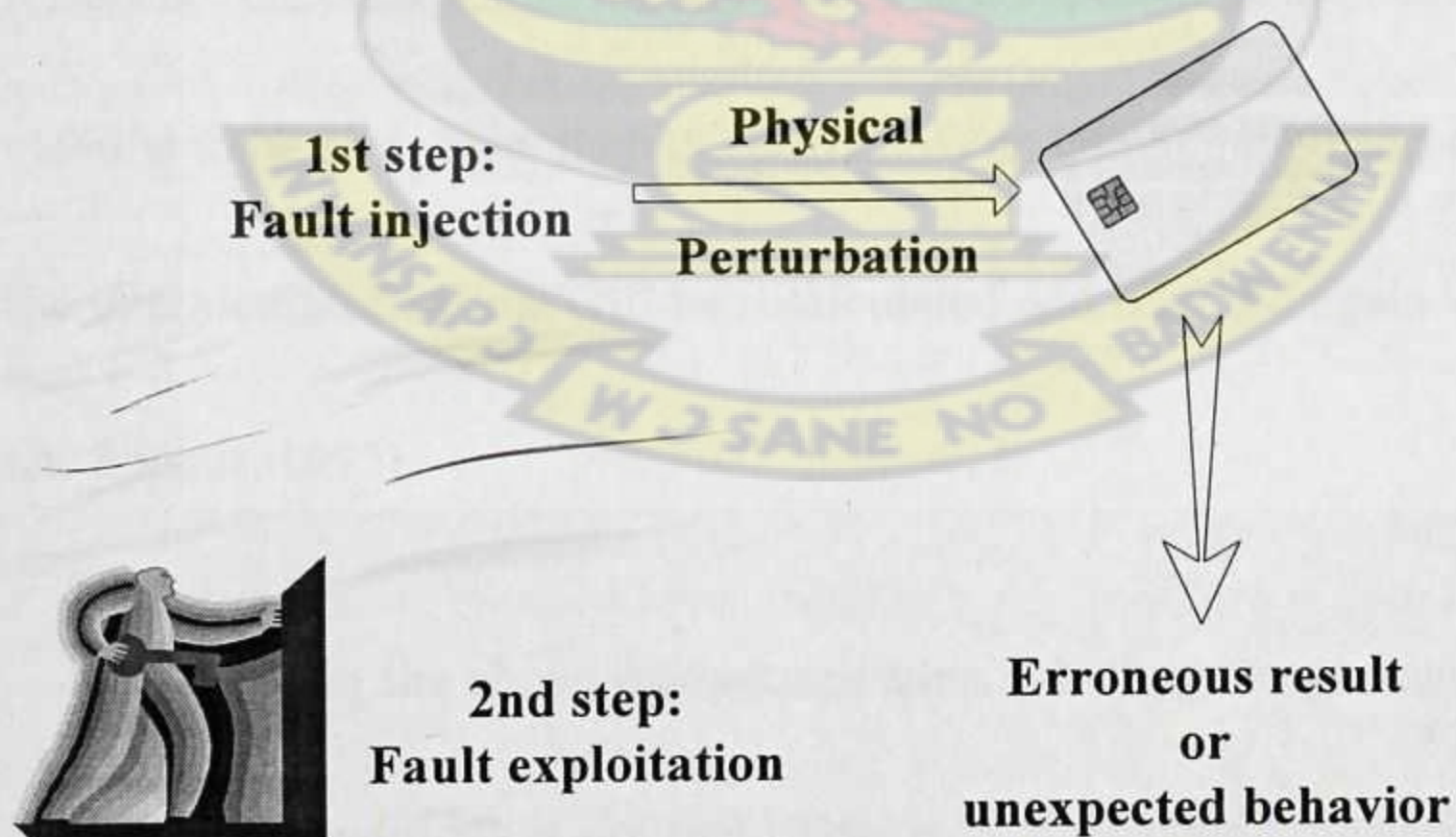


Figure 2. 1: The process of the hardware/software fault attack

As regards to the second and the third conditions, some countermeasures have been presented to avoid sending faulty signatures/erroneous responses out of the device or system. The basic idea is to use the checking method to avoid obtaining/sending out faulty results/erroneous responses (A. Shamir, 1995). The most obvious way is to repeat the computation and check whether the same signature is obtained both times, which slows down the signing operation by a factor of two. Another way is to check whether the message m can be recovered from the signature s to decide the correctness of the signature. One disadvantage of either repeating the computation or checking whether the message can be recovered from the signature is that the calculation speed is almost slowed down by a factor of two. Shamir presented a checking method with simpler calculations, in which the intermediate results are checked before the signature s is computed. If the intermediate results are claimed to be error-free, then the signature can be computed and sent out, otherwise, the intermediated results will be recalculated and checked again until it is error-free (A. Shamir, 1995).

Other than the above countermeasures, which try to guarantee that at least one of the three conditions is not met, there is another countermeasure proposed by Yen et al, (2003). The idea is to revise the signature calculation method of the CRT-based RSA, so that the faulty signature will not reveal the secret information of the CRT-based RSA cryptosystem. Yen et al. proposed two protocols (Yen et al, 2003), which

assure the occurred fault in one module will affect the other module or the overall computation, so that the faulty signature will not reveal the secret information.

2.4.2 Timing Attacks

The timing attack is basically a way of deciphering a user's private key information by measuring the time it takes to carry out cryptographic operations (E. English & S. Hamilton, 1996). By carefully measuring the amount of time required to perform private key operations in a smartcard that stores a private RSA key while the card is tamper resistant, the attacker may be able to discover the private decryption exponent d (W. Schindler, 2000). This attack is computationally inexpensive and often requires knowing only the ciphertext to be performed. Actual systems are potentially at risk, including cryptographic tokens, network-based cryptosystems, and other applications where attackers can make reasonably accurate timing measurements (P. Kocher, 1996).

There are some methods (P. Kocher, 1996) to prevent the timing attack to the RSA cryptosystems, in which the most obvious one is to make all operations take exactly the same amount of time. The second approach is to make timing measurements inaccurate by adding random delay to the processing time so that the attack becomes unfeasible. Another method is to adapt blind signatures so that the attackers do not know the input to the modular exponentiation function.

2.4.3 Power Attack

The power attack of a smartcard is a technique that involves directly interpreting power consumption measurements collected during cryptographic operations to expose the secret key d (P. Kocher et al, 1999).

There are several countermeasures to the power attack (S. Thomas, 2000). The first approach is to reduce signal sizes and choose operations that leak less information on their power consumption. However, making the attack infeasible by aggressive shielding the device will significantly increase the cost and size of a device. The second approach is to introduce noise into power consumption measurements so that the measurements by the attacker are inaccurate.

2.5 Random Number and Random Bit Generators

A random number and random bit generator, RNG and RBG, respectively, is a fundamental tool in many different areas (Andrea Rock, 2005). He stated in his paper that the two main fields of application are stochastic simulation and cryptography. In stochastic simulation, RNGs are used for mimicking the behavior of a random variable with a given probability distribution. In cryptography, these generators are employed to produce secret keys, to encrypt messages or to mask the content of certain protocols by combining the content with a random sequence. Andrea Rock (2005) stated also that a further application of cryptographically secure random numbers is the growing area of internet gambling since these games should imitate very closely the distribution properties of their real equivalents and must not be predictable or influenceable by any adversary.

A random number generator is an algorithm that, based on an initial seed or by means of continuous input, produces a sequence of numbers or respectively bits. We demand that this sequence appears random" to any observer (Wolfram Research Inc., 2008). They stated in these Tutorials that most Researchers will claim that they know what randomness means, but if they are asked to give an exact definition they will have a problem doing so. In most cases terms like unpredictable or uniformly distributed will be used in the attempt to describe the necessary properties of random numbers. However, when can a particular number or output string be called unpredictable or uniformly distributed? In this part we will introduce three different approaches to define randomness or related notions (Andrea Rock, 2005).

In the context of Random Numbers and Random Number Generators (RNGs) the notions of "real" random numbers and True Random Number Generators (TRNGs) appear quite frequently. By real random numbers we mean the independent realizations of a uniformly distributed random variable, by TRNGs we denote generators that output the result of a physical experiment which is considered to be random, like radioactive decay or the noise of a semiconductor diode. In certain circumstances, RNGs employ TRNGs in connection with an additional algorithm to produce a sequence that behaves almost like real random numbers (Fischer, V., et al, 2009).

However, why would we use RNGs instead of TRNGs? First of all, TRNGs are often biased, this means for example that on average their output might contain more ones than zeros and therefore does not correspond to a uniformly distributed random variable. This effect can be balanced by different means, but this post-processing reduces the number of useful bits as well as the efficiency of the generator. Another problem is that some TRNGs are very expensive or need at least an extra hardware

device. In addition, these generators are often too slow for the intended applications. Ordinary RNGs need no additional hardware, are much faster than TRNGs, and their output fulfills the fundamental conditions, like unbiasedness, that are expected from random numbers.

These conditions are required for high quality RNGs but they cannot be generalized to the wide range of available generators. Despite the arguments above, TRNGs have their place in the arsenal. They are used to generate the seed or the continuous input for RNGs. In (Ellison, C. 1995) the author lists several hardware sources that can be applied for such a purpose. Independently of whether a RNG is used for stochastic simulation or for cryptographic applications, it has to satisfy certain conditions. First of all the output should imitate the realization of a sequence of independent uniformly distributed random variables.

Random variables that are not uniformly distributed can be simulated by applying specific transformations on the output of uniformly distributed generators, (L. Devroye, 1996) for some examples or (W. Hormann & J. Leydold, 2000), which provides a program library that allows to produce non-uniform random numbers from uniform RNGs.

Andrea Rock (2005), in his paper limited his discussion on generators that imitate uniformly distributed variables. In a binary sequence that was produced by independent and identically uniform random variables the ones and zeros as well as all binary n -tuples for $n \geq 1$ are uniformly distributed in the n -dimensional space. Furthermore there exists no correlation between individual bits or n -tuples, respectively. From the output of a high quality RNG we expect the same behavior.

For some generators those conditions can be checked by theoretical analysis, but for most RNGs they are checked by means of empirical tests.

Moreover, a good RNG should work efficiently, which means it should be able to produce a large amount of random numbers in a short period of time. For applications like stochastic simulation, stream ciphers, the masking of protocols or online gambling, huge amounts of random numbers are necessary and thus fast RNGs are required.

In addition to the conditions above, RNGs for cryptographic applications must be resistant against attacks (Mike Hamburg, 2012), a scenario which is not relevant in stochastic simulation. This means that an adversary should not be able to guess any current, future, or previous output of the generator, even if he or she has some information about the input, the inner state, or the current or previous output of the RNG.

The topic of cryptographic RNGs concerns both mathematicians and engineers. Most of the time engineers are more interested in the design of specific RNGs or test suites, whereas mathematicians are more concerned with definitions of randomness, theoretical analysis of deterministic RNGs and the interpretation of empirical test results (Andrea Rock, 2005). In his thesis he tried to address both disciplines by giving the description of five real-world cryptographic RNGs as well as the necessary mathematical background. This thesis however comparatively analysis the five real-world cryptographic RNGs to see which of them are more efficient than the other to be used in Cryptography.

2.6 Categories of Random Numbers Generators

There are three types of random numbers, quasi-, pseudo- and true- random numbers. These different types of random numbers have different applications (Mario Rütli, 2004). (It is philosophical question what we can call random or not, but here, we use the following descriptions, its simpler)

True Random Number: The most often used example for “truly” random numbers is the decay of a radioactive material. If a Geiger counter is put in front of such a radioactive source, the intervals between the decay events are truly random. True random numbers are gained from physical processes like radioactive decay or also rolling a dice. But rolling a dice is difficult; perhaps someone could control the dice so well to determine the outcome.

Pseudo Random Number: According to Kaushik Patowary (2009), essentially, PRNGs are algorithms that use mathematical formulae or simply precalculated tables to produce sequences of numbers that appear random. Examples of PRNG are:

- Blum Blum Shub
- Complementary-multiply-with-carry
- Inversive congruential generator
- ISAAC (cipher)
- Fibonacci Random Numbers Generator
- Specific Range Random Numbers Generators
- Gaussian (Normal) Distribution Random Numbers Generators
- Secured Random Numbers Generators
- Linear congruential generator
- Maximal periodic reciprocals

- Mersenne twister
- Multiply-with-carry
- Naor-Reingold Pseudorandom Function
- Park–Miller random number generator

A good deal of research has gone into pseudo-random number theory, and modern algorithms for generating pseudo-random numbers are so good that the numbers look exactly like they were really random. According to F. Rusu and A. Dobra (2007), AMS-sketches were used to determine mathematically the efficiency of the Pseudo-Random Numbers generators. In this paper, they provided a thorough comparison of the various generating schemes with the goal of identifying the efficient ones. To this end, they explain how the schemes can be implemented on modern processors and use such implementations to empirically evaluate them.

Quasi Random Number: A good description quoted from (W. H. Press et al, 1992) stated that sequences of n -tuples that fill n -space more uniformly than uncorrelated random points are called quasi-random sequences. That term is somewhat of a misnomer, since there is nothing random about quasi-random sequences: They are cleverly crafted to be, in fact, sub-random. The sample points in a quasi-random sequence are, in a precise sense, maximally avoiding each other.

Quasi random numbers are not designed to appear random, rather to be uniformly distributed. One aim of such numbers is to reduce and control errors in Monte Carlo simulations. A picture is always a good way to illustrate the difference between this two types. In figure 2.2 and 2.3 we have plots with different numbers of pseudo- and quasi-random numbers. This is a good demonstration to show the structure of quasi-random numbers, but it is also possible to see that quasi-random

numbers fill continuously the hole plane, while pseudorandom numbers may build clusters and holes. If we are talking about random numbers in the following parts, we mean pseudo random numbers.

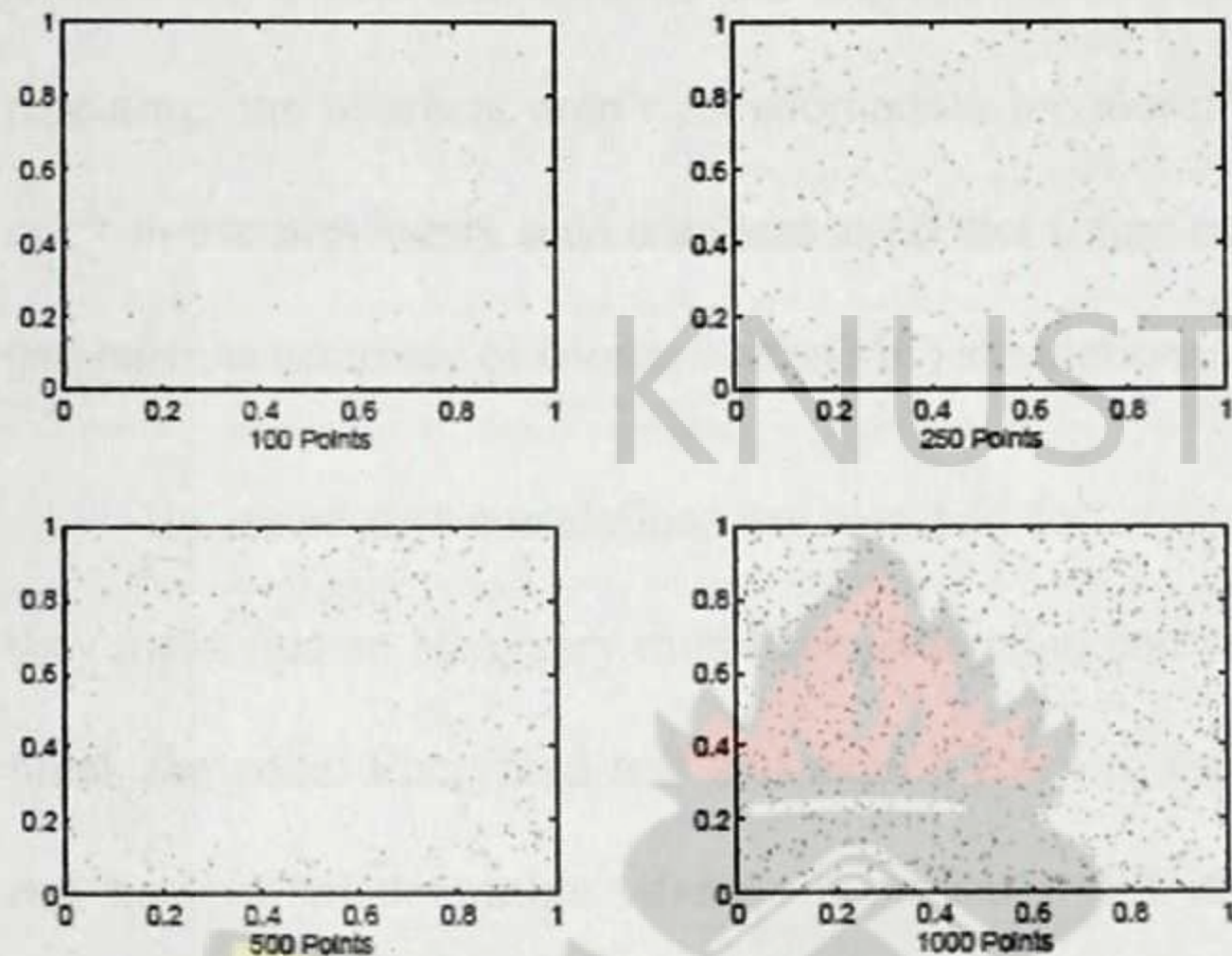


Figure 2.2: Pseudo Random Numbers

(Source: Mario Rütli, 2004)

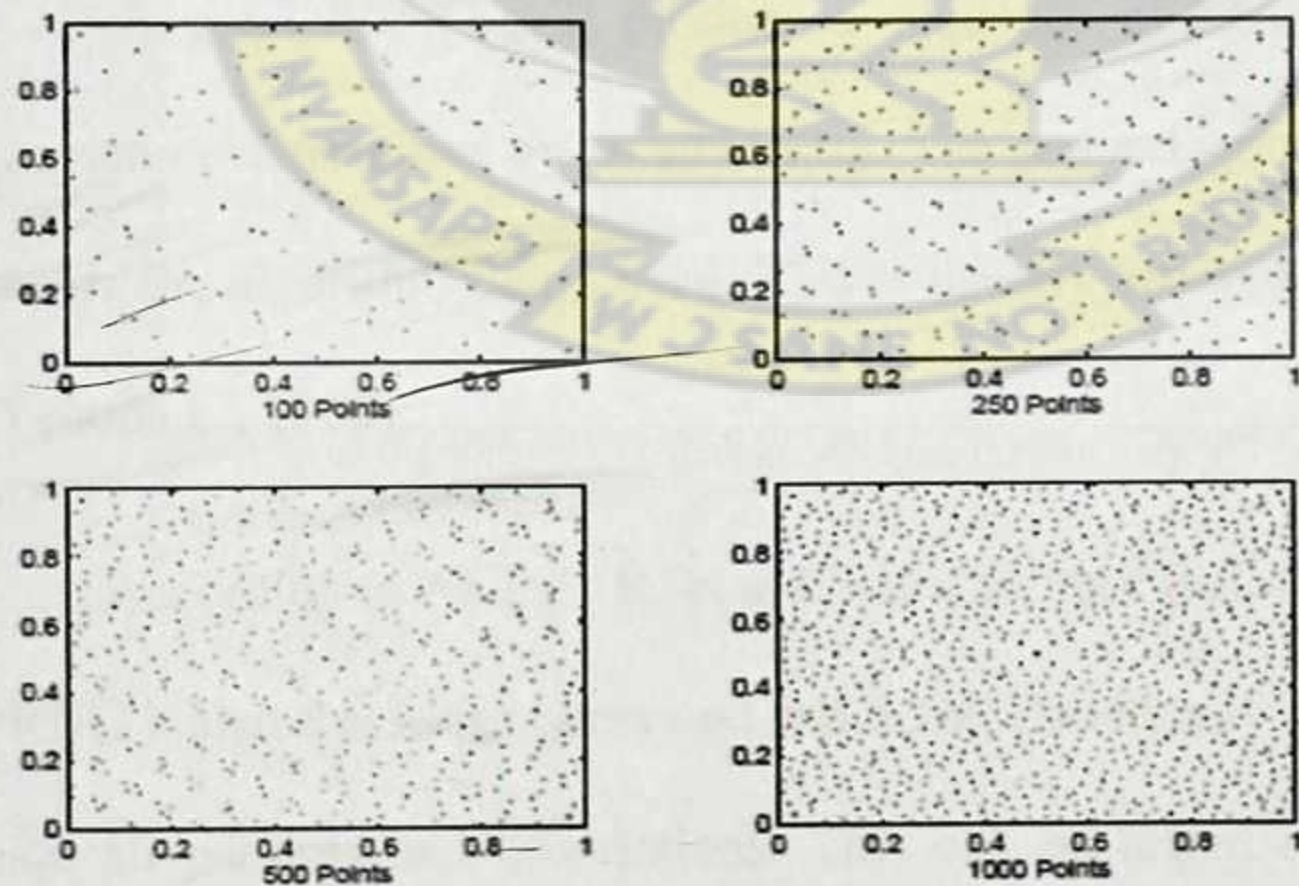


Figure 2.3: Quasi Random Numbers

(Source: Mario Rütli, 2004)

2.7 Properties of Good PRNGs

According to M. Hoemmen (2006), each PRNG defines a single sequence of pseudorandom numbers. Since modular integer arithmetic allows only a finite number of numbers, eventually the sequence must repeat. Once the numbers start repeating, the numbers aren't pseudorandom anymore; they are correlated. That is, one can use previously seen numbers to predict future numbers in the sequence. This can hurt the accuracy of Monte Carlo (MC) simulations.

He stated that correlations are also bad for cryptography algorithms, because they mean that an adversary might see patterns in encrypted text and exploit them to break the code. Encrypted text and exploit them to break the code. Statistical tests can be used to determine whether a particular PRNG produces a sequence with sufficient statistical randomness may break on a certain sensitive application. It may be a good idea to test a MC simulation with multiple PRNG's, to check if it's sensitive to the particular

A good PRNG will produce a sequence of numbers that cannot be easily guessed or determined by an adversary. The general assumption is that the opponent knows the algorithm being used. This is usually referred to as Kerckhoff's principle (Yehuda L., 2006).

According to Andrew R. el at (2010), another factor to be considered for a good PRNG is that the series generated should be statistically random. It should be able to hide all patterns and correlations. This can be tested by some statistical tests. A PRNG should have the statistical property that each value over the interval has an equally likely occurrence. In the case of the interval $[0,1]$ each number would appear with the frequency $1/2$ in a long run. Each pair of numbers would appear with

frequency $1/4$, and each triplet with frequency $1/8$. This would form a basis of observation that the numbers being generated are unbiased and successive outcomes are uncorrelated (James A., et al (2003)). Tests should also include mean and variance checks. Mean should be close to 0.5 and variance $1/12 = 0.08$ for uniformly distributed pseudorandom numbers.

Again, NIST (2010) stated that, Frequency (Monobits) Test is also another test for efficiency. The focus of the test is the proportion of zeroes and ones for the entire sequence. The purpose of this test is to determine whether that number of ones and zeros in a sequence are approximately the same as would be expected for a truly random sequence. The test assesses the closeness of the fraction of ones to $1/2$, that is, the number of ones and zeroes in a sequence should be about the same. Test for Frequency within a Block: The focus of the test is the proportion of zeroes and ones within M-bit blocks. The purpose of this test is to determine whether the frequency of one's in an M-bit block is approximately $M/2$.

According to Justin A. (2012 August 1), Period is another factor that affects the efficiency of PRNG algorithm: Each algorithm has a period that depends on its input parameters. Whenever the algorithm produces a result that is the same as its seed, it starts over generating the same numbers. Note that the period of an algorithm can depend strongly on its seed and/or its parameters. The shorter the period, the more repetitions of numbers, the generator and hence the high probability of predictability.

Uniformity: Good PRNG algorithms will generate numbers uniformly across their ranges. Algorithms with small periods but large ranges will fail this test (Christophe D.& Diethelm W.,2003 September). But even some algorithms with

large periods can give problematic results if there are correlations between the numbers they produce.

Christophe D. and Diethelm W.(2003), stated that it is important to ensure that a given PRNG does not have obvious correlations between successive elements of its sequence. This can mean that even an algorithm with a large period and good (eventual) uniformity can be strongly non-uniform when a limited number of values are generated. The more the correlation, the more numbers generated are dependent on each other, the less secured the generator.

2.8 Conclusion

In this Section, the most widely used Pseudo-Random Numbers Generators (PRNG) in cryptography, more specifically, RSA Cryptography has been described and related literature being reviewed in the quest to determine the efficiency of the PRNG algorithm in a Cryptographic Application.

CHAPTER THREE

RESEARCH METHODOLOGY

3.1 Introduction

This chapter presents different methods of testing Random number Generators (RNGs) for a Cryptographic Applications employed by the researcher. These tests are the test for randomness of the Random Number Generator (RNGs) Algorithm, and the test for the efficiency of the Random Number Generators (RNGs) Algorithm. The following sections present three main methods of testing for Random Number Generators (RNGs) and its parameters for testing for both efficiency and randomness of the individual generator widely used in Cryptographic Applications employed the researcher.

3.2 Test for Random Number Generators

Tests for RNGs can be used in two ways; they are used to determine whether a set of data has a recognizable pattern to it, and the determining how efficient a particular RNGs algorithm is (Wolfram & Stephen, pp. 975-976, (May 2002).).

3.2.1 Tests for Randomness of Random Number Generators (RNGs).

Randomness tests (or tests of randomness), in data evaluation, are used to analyze the distribution pattern of a set of data. In stochastic modeling, as in some computer simulations, the expected random input data can be verified to show that tests were performed using randomized data.

3.2.2 Chi-square Test

The chi-square test is the most commonly used test for the randomness of data, and is extremely sensitive to errors in pseudorandom sequence generators. The chi-square distribution is calculated for the stream of bytes in the file and expressed as an absolute number and a percentage which indicates how frequently a truly random sequence would exceed the value calculated. The procedures used are as follows:

- i. Java Programming codes is written for each of Fibonacci Random Numbers Generators, Gaussian Random Numbers Generators, Secure Random Numbers Generators and Specific Range Random Numbers Generators
- ii. This test involved producing sequences of 100 random integers between 0 and 1000 using the Java codes for each of Fibonacci Random Numbers Generators, Gaussian Random Numbers Generators, Secure Random Numbers Generators and Specific Range Random Numbers Generators. The generated random numbers are shown in *Appendix IV*.
- iii. The generated random numbers in (ii) above is coded in Statistical Package for Social Science (SPSS) software to test of randomness of the numbers in the generators. The procedures include:
 - a. Go to and click on the *Analyse Menu* on the SPSS bar after the data has been coded.
 - b. Choose *Nonparametric Test* on the submenu.
 - c. Go to and then Click on *Chi-Square Test*.
 - d. Select all the four random numbers generators and move them to the *variable list*.
 - e. The results generated for the Chi-Square Test is show in the *Appendix II*.

- f. The results obtained as in *Appendix II* are interpreted.
- iv. If these generator produces uniformly distributed numbers, one would expect to have about 1 occurrences of the integer 1, 1 occurrences of the integer 2 and so on up to 1 occurrences of the integer 100. Therefore, the expected frequency would be 1.1 or 1.0 for each integer 1 to 100. The actual frequency of each integer produced by the generator is the observed frequency for that integer. The difference between the observed and the expected frequency of each integer can be used to compute the chi-square statistic as follows:

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

Where $R = 100$, is the number of different random integers possible, O_i is the observed frequency of occurrence for the random integer i and E_i is the expected frequency of occurrence for the random integer i . Because the distribution of integers is expected to be uniform, the expected frequencies of occurrence for each random integer are equal. If N is the total number of observations, then the following equation can be used to compute E_i . The Chi-Square values for the four random numbers generators obtained are shown below:

Table 3.1

The Chi-Square Test Result for Pseudo-Random Numbers Generators

Test Statistics				
	Specific Range Random Number Generator	Secure Random Number Generator	Gaussian Random Number Generator	Fibonacci Random Number Generator
Chi-Square	4.500 ^a	7.380 ^b	14.400 ^c	.000 ^d
Df	94	90	87	99
Asymp. Sig.	1.000	1.000	1.000	1.000
a. 95 cells (100.0%) have expected frequencies less than 5. The minimum expected cell frequency is 1.1.				
b. 91 cells (100.0%) have expected frequencies less than 5. The minimum expected cell frequency is 1.1.				
c. 88 cells (100.0%) have expected frequencies less than 5. The minimum expected cell frequency is 1.1.				
d. 100 cells (100.0%) have expected frequencies less than 5. The minimum expected cell frequency is 1.0.				

- v. We interpret the percentage as the degree to which the sequence tested is suspected of being non-random. If the percentage is greater than 99% or less than 1%, the sequence is almost certainly not random. If the percentage is between 99% and 95% or between 1% and 5%, the sequence is suspect. Percentages between 90% and 95% and 5% and 10% indicate the sequence is “almost suspect”.

3.2.3 Kolmogorov-Smirnov test (KS-test)

The Kolmogorov-Smirnov test (KS-test) tries to determine if two datasets differ significantly. The KS-test has the advantage of making no assumption about the distribution of data. (Technically speaking it is non-parametric and distribution free. The Kolmogorov-Smirnov test (Chakravart, Laha, and Roy, 1967) is used to decide if a sample comes from a population with a specific distribution. The Kolmogorov-Smirnov (K-S) test is based on the Empirical Distribution Function (ECDF). Given N ordered data points Y_1, Y_2, \dots, Y_N , the ECDF is defined as

$$E_N = n(i)/N$$

where $n(i)$ is the number of points less than Y_i and the Y_i are ordered from smallest to largest value. This is a step function that increases by $1/N$ at the value of each ordered data point.

The procedures used to perform the Kolmogorov-Smirnov test (KS-test) for the four random numbers generators namely: Fibonacci Random Numbers Generators, Gaussian Random Numbers Generators, Secure Random Numbers Generators and Specific Range Random Numbers Generators are as follows:

- i. Java Programming codes is written for each of Fibonacci Random Numbers Generators, Gaussian Random Numbers Generators, Secure Random Numbers Generators and Specific Range Random Numbers Generators.
- ii. This test involved producing sequences of 100 random integers between 0 and 1000 using the Java codes for each of Fibonacci Random Numbers Generators, Gaussian Random Numbers Generators, Secure Random Numbers Generators and Specific Range

Random Numbers Generators. The generated random numbers are shown in *Appendix IV*.

- iii. The generated random numbers in (ii) above is coded in Statistical Package for Social Science (SPSS) software to test of randomness of the numbers in the generators. The procedures include:
 - a. Go to and click on the *Analyse Menu* on the SPSS bar after the data has been coded.
 - b. Choose *Nonparametric Test* on the submenu.
 - c. Go to and then Click on *One Sample KS-Test*.
 - d. Select all the four random numbers generators and move them to the *variable list* and check *Normal*.
 - e. Click on *Exact* tap and select *asymptotic* option and then click OK.
 - f. The results generated for the *One Sample KS-Test* is show below:

Table 3.2

The descriptive statistics of the Random Numbers Generators

Generators	N	Mean	Std. Deviation	Minimum	Maximum
Specific Range Random Number Generator	100	479.07	302.50	2	988
Secure Random Number Generator	100	512.32	284.47	34	958
Gaussian Random Number Generator	100	5.08	68.51	-121	126
Fibonacci Random Number Generator	100	-90142675.73	975930540	-2092787285	2.E9

Table 3.3

The Kolmogorov-Smirnov Test for Uniformity of random numbers

Statistics			Specific Range Random Numbers Generator	Secure Random Numbers Generator	Gaussian Random Numbers Generator	Fibonacci Random Numbers Generator
N			100	100	100	100
Uniform Parameters		Minimum	2	34	-121	-
		Maximum	988	958	126	2092787285
		Absolute	0.103	0.070	0.072	2144908973
Most Differences	Extreme					0.202
		Positive	0.103	0.045	0.072	0.202
		Negative	-0.039	-0.070	-0.060	-0.164
Kolmogorov-Smirnov Z			1.027	0.701	0.725	2.020
Asymp.Sig. (2-tailed)			0.243	0.709	0.670	0.001
Sig.			0.300	0.670	0.640	0.000
Monte carlo.Sig.(2- tailed)	90%Confidence Interval	Lower	0.225	0.593	0.561	0.000
		Bound				
		Upper	0.375	0.747	0.719	0.23
		Bound				

3.3 Frequency Test of Random Numbers Generators

The researcher used four (4) Pseudo-Random Numbers Generators (PRNGs) namely: Fibonacci Random Numbers Generator (FRNG), Secure Random Numbers Generators (SRNG), Specific Range Random Numbers Generator (RRNG) and Gaussian Random Numbers Generator (GRNG). The following procedures are used:

- The first 100 random numbers are generated with Java codes using the above random numbers generators,
- The output of the generation is shown in *Appendix IV*.

- iii. These outputs for the different random numbers generators are coded into SPSS to compare the number of times each numbers in the generator repeats itself. The procedures here are.
- Go to and click on the *Analyse Menu* on the SPSS bar after the data has been coded.
 - Choose *Descriptive Statistics* on the submenu.
 - Go to and then Click on *Frequency*.
 - Select all the four random numbers generators and move them to the *variable* window.
 - Click OK.
 - The results generated for the *Frequency test* is shown in *Appendix*

III:

3.4. Conclusion

In conclusion, the researcher employed three main tests for the efficiency of the PRNGs algorithms namely; the frequency test, Chi-Square test and Kolmogorov-Smirnov test. The output of each of these tests is the determinant factor for whether or not any of the four compared PRNGs is more secured generator than the other.

CHAPTER FOUR

RESULTS AND DISCUSSIONS

4.1 Introduction

This chapter presents the analysis and discussions of the results for four Pseudo-Random Number Generators (PRNGs) namely; Fibonacci Random Number Generator, Secure Random Numbers Generators, Specific Range Random Numbers Generator and the Gaussian Random Numbers Generator. Frequency Test, Chi-Square and Kolmogorov-Smirnov Tests are used to test the first 100 random numbers generated from these four PRNGs to determine its randomness. The sections below present the analysis and the discussion of the results.

4.2 Frequency Test of Random Numbers Generators

4.2.1 Specific Range Random Numbers Generator

It is found out that the Specific Range Random Numbers Generator produces bigger random numbers that are clustered together into a range of 0 to 200. The quantum of random numbers however decrease between the range of 200 to 400, and hence then scattered from the range of 400 to 1000. In general random numbers produced as a result of this generator are not uniform as they are scattered throughout from 0 to 100 ranges of input data as shown in *Figure 4.1*. Numbers produced in this generator is therefore pattern-based as seen in *Figure 4.1*. It can easily be found that if the first number produced is low then the second number will be high and vice versa. This therefore makes Specific Range Random Generator less uniform and hence deviate from the normal distribution curve with standard deviation of 302.502.

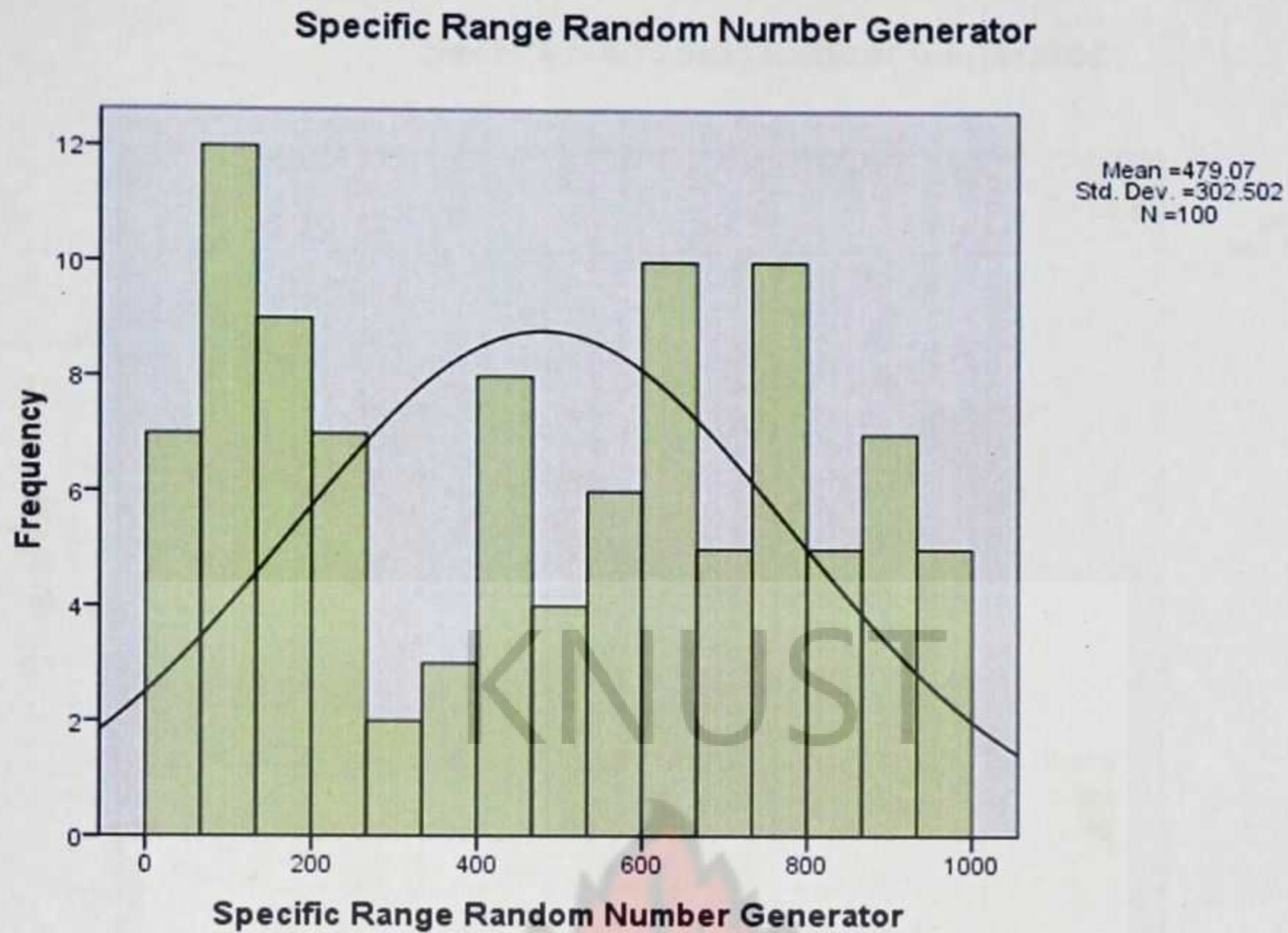


Figure 4.1: The trend of randomness in a Specific Range Random Number Generator

4.2.2 Secure Random Number Generator

The Secure Random Numbers Generator produces bigger random numbers just like the Specific Range Random Numbers, but more skew to the center than its counterpart. This also produces random numbers that are not much uniform and hence does not follow the normal distribution curve. However, numbers at certain ranges in the results are much uniform than at certain points. As shown in the *Figure 4.2*, between the ranges of 0-250, 350-450, 600-700 and 800-900, numbers are more uniform than the ranges 200-400, 400-600, 650-800 and 900-1000. This results deviates from the normal with a standard deviation of 284.471. Numbers produced in this generator is also pattern-based as seen in *Figure 4.2*. It can easily be found also that if the first number produced is low then the second number will be high and vice versa.

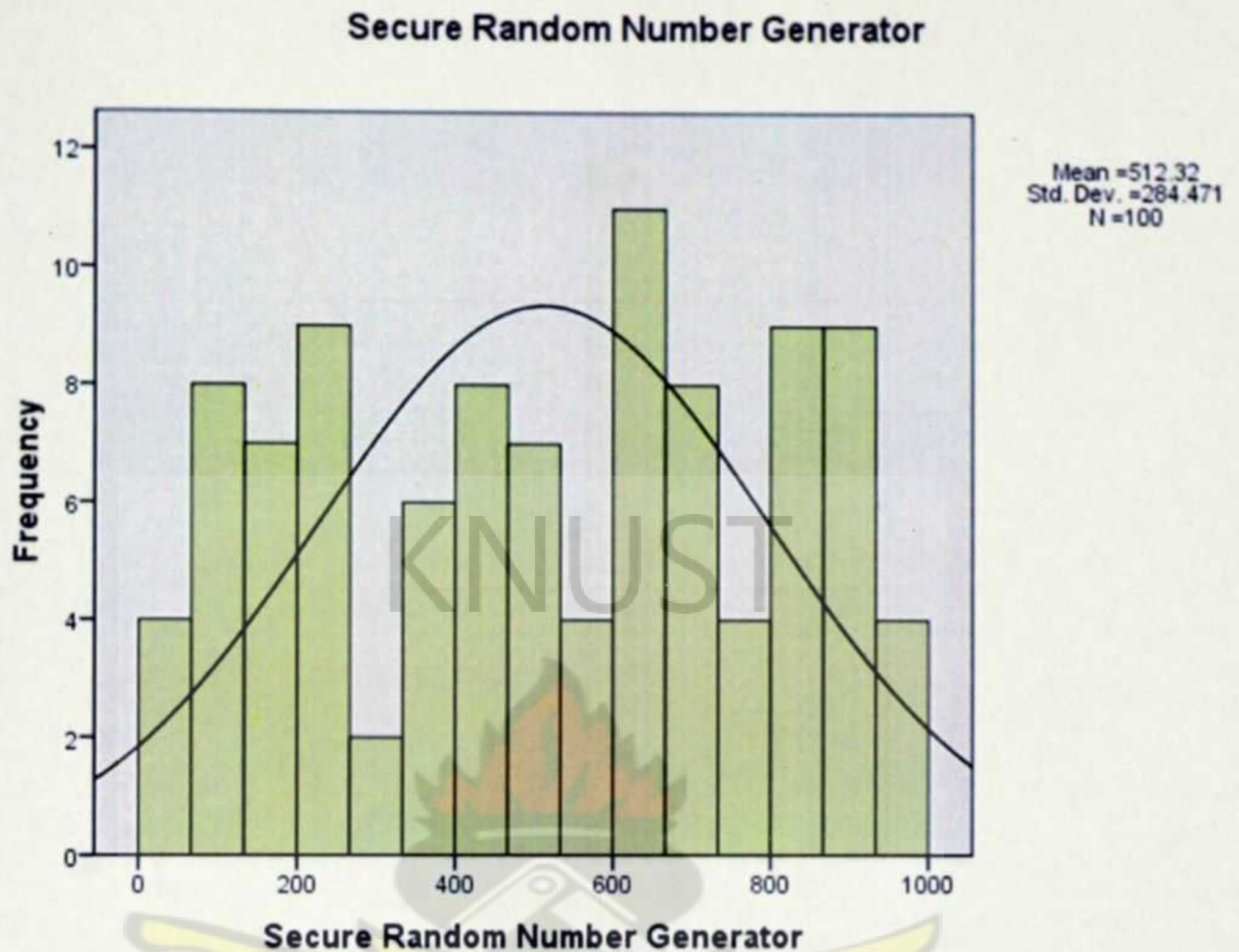


Figure 4.2: The trend of randomness in Secure Random Number Generator

4.2.3 Gaussian Random Number Generator

This generator produces smaller random numbers at the beginning and at the end of the range used, while producing bigger numbers at the middle of the range of the numbers generated as shown in *Figure 4.3*. However abnormal uniformity occurs at the range of -100 to 50. The generator produces random numbers that are much skewed to the center, hence produces a substantially good standard deviation of 68.515. This standard deviation value shows that the generator obeys the normal distribution curve.

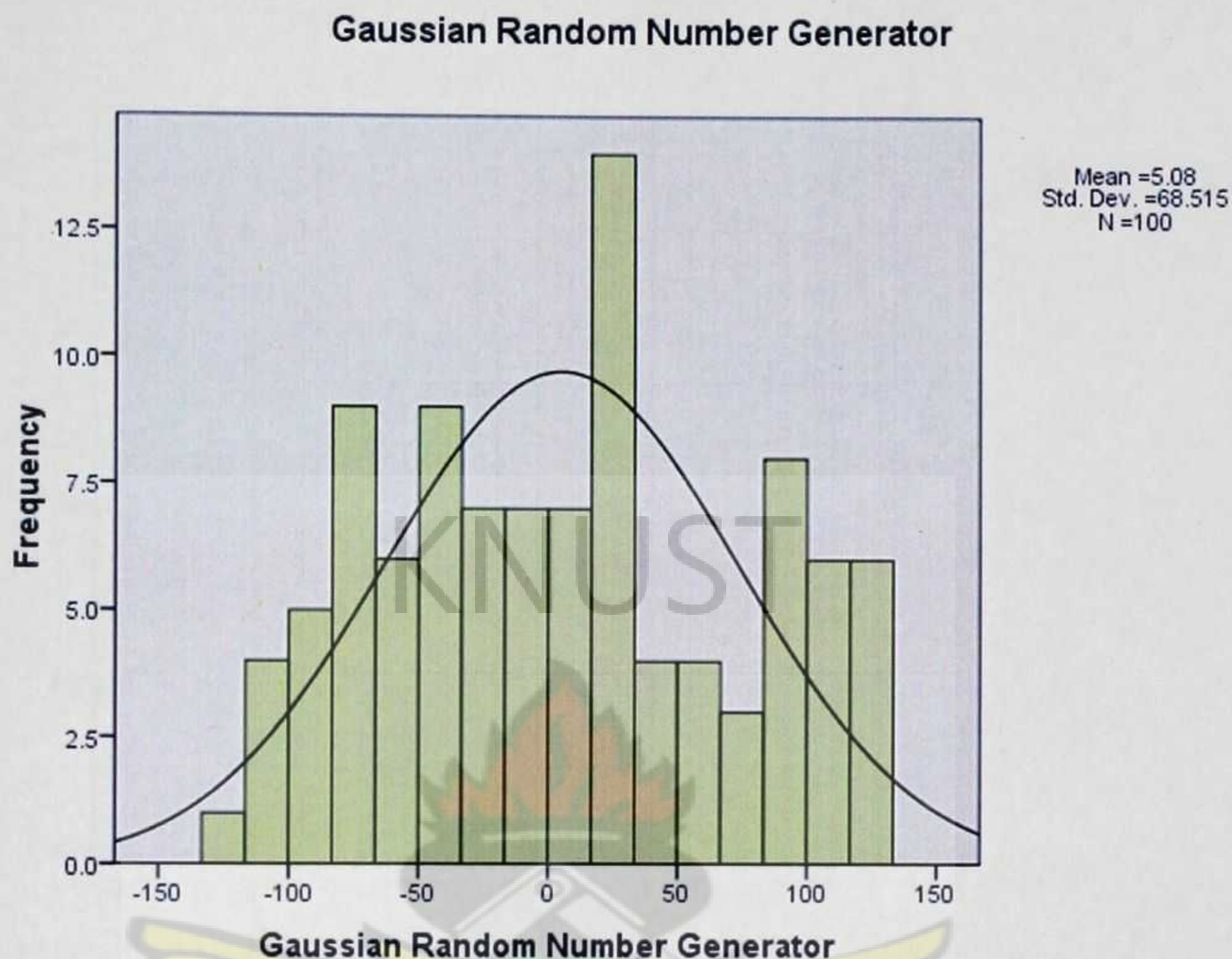


Figure 4.3: The trend of randomness Gaussian Random number Generator

4.2.4 Fibonacci Random Number Generator

From the results generated, it is found out that the Fibonacci Random Numbers Generator started with smaller random numbers at the beginning, elevates gradually, gets to its peak at a value of 0.0E0 and declines gradually to the end of the range as in *Figure 4.4*. This result tends to obey the normal distribution curve more than all the other generators studied, with standard deviation of 9.759E8. This represents the greatest deviation of individual numbers produced by the generator from each other and hence not normally distributed.

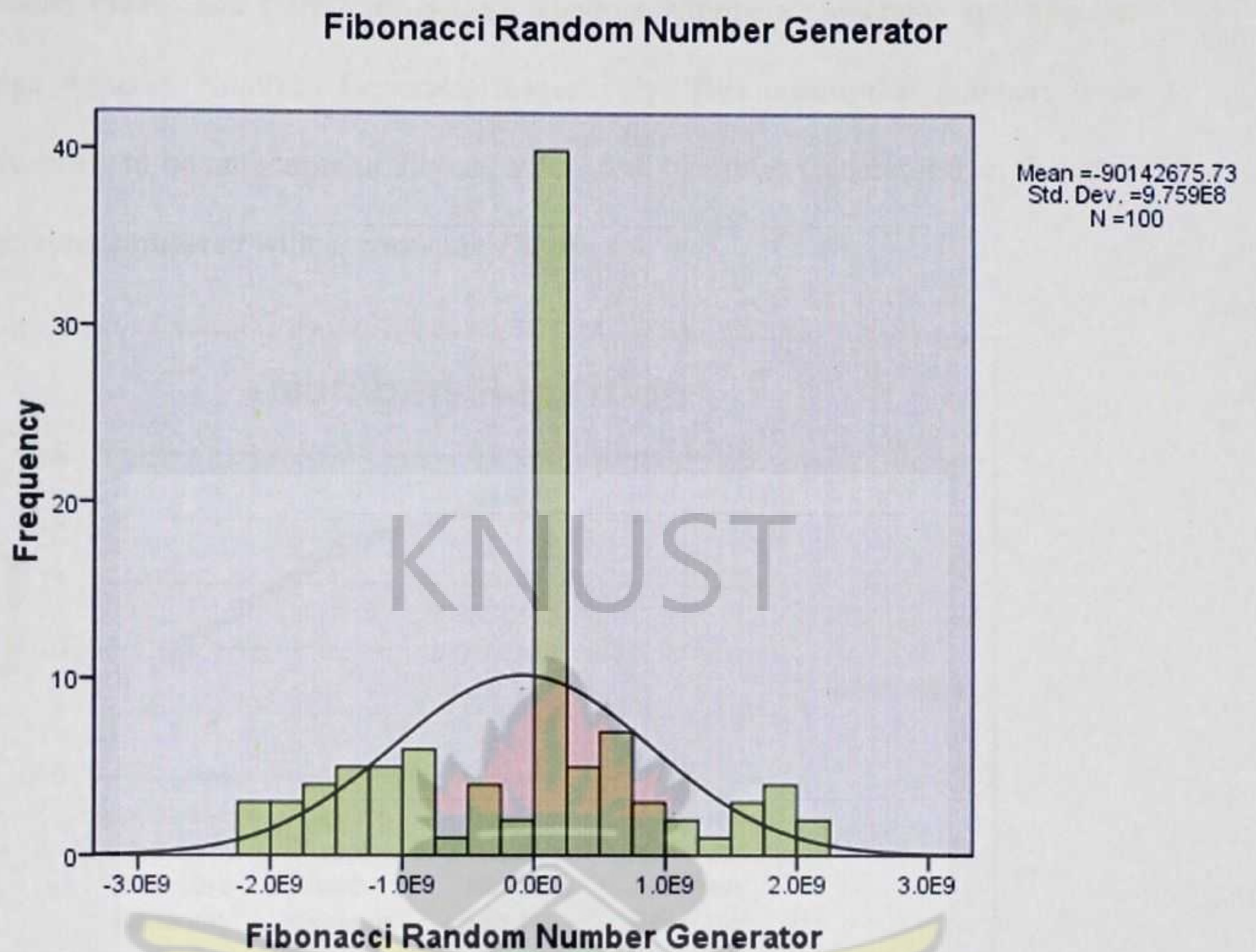


Figure 4.4: The trend of randomness in Fibonacci Random Number Generator

The researcher in comparing the four (4) Pseudo-Random Numbers Generators (PRNGs) studied namely: Fibonacci Random Numbers Generator (FRNG), Secure Random Numbers Generators (SRNG), Specific Range Random Numbers Generator (RRNG) and Gaussian Random Numbers Generator (GRNG).

The results of the uniformity of the four generators are used to draw a line graph as shown in *Figure 4.5*. The figure shows a line graph of frequencies (expressed in %) for the four random numbers generators. It is found out that the random number generator with the highest number of repeated numbers is Gaussian Random Numbers Generator (22%), while Fibonacci Random Numbers Generator had no

repeated numbers in the random numbers. The repetition of the other generators included (18%) and (10%) for Secure Random Numbers Generator and Specific Range Random Numbers Generator respectively. This means that numbers were more likely to be random with Fibonacci Random Numbers Generator than the other generators compared with as shown in *Figure 4.5*.

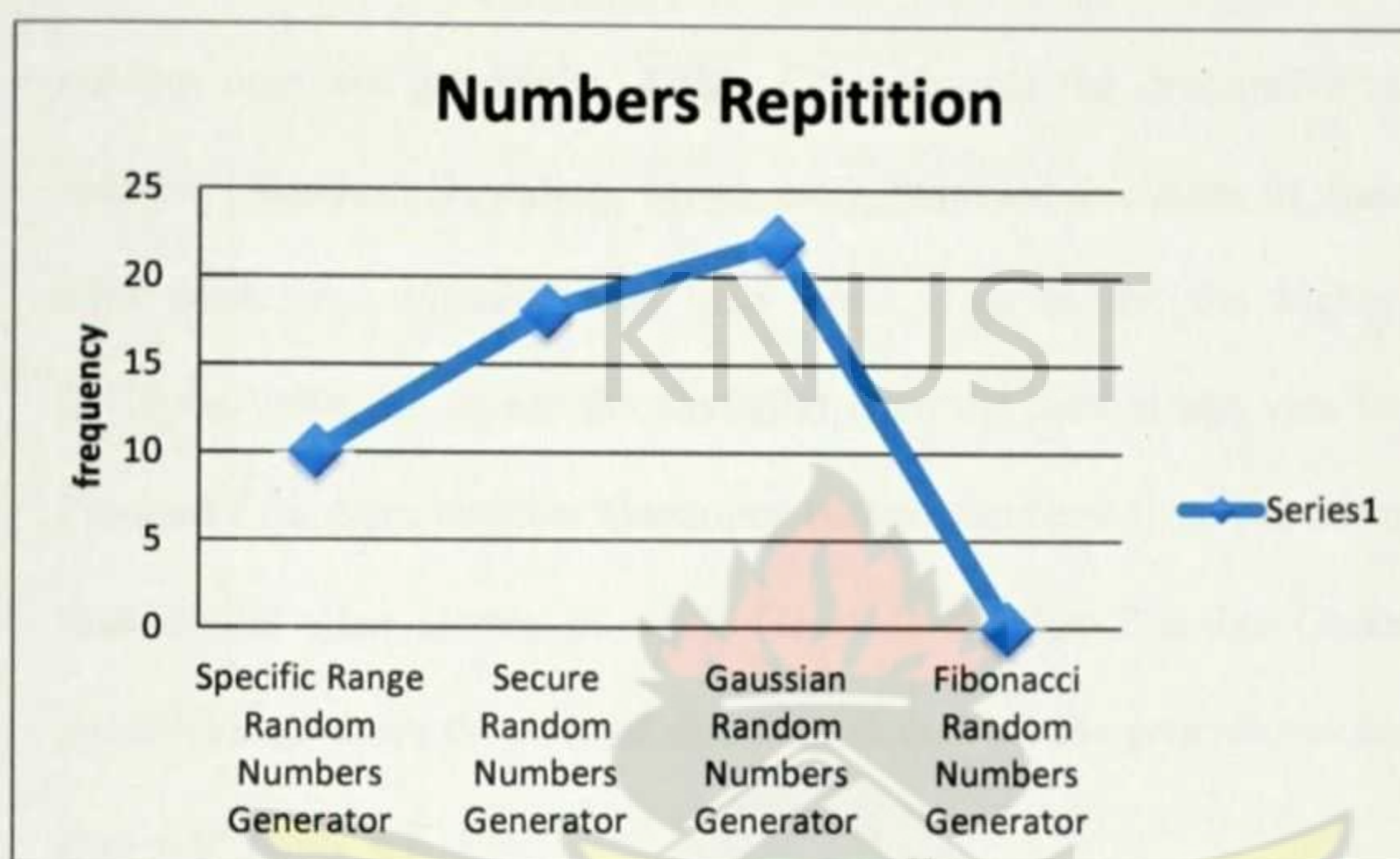


Figure 4.5: The number of repetitions in Pseudo Random Number Generators

4.3 The Chi-Square Test for Uniformity of random numbers

The analysis of the factors responsible for randomness in a random numbers generators showed that factors tend to give much considerations to Fibonacci Random Numbers Generator than the other Generators compared with. This is because the Chi-Square analysis for the independence of numbers in the generators showed that numbers were more independent to each other in the Fibonacci Random Numbers Generator (Chi-Square Value = 0.000) than the other generators under studied. This is because independence of numbers increases with the decrease in Chi-Square Value and vice versa.

However, the worst generator in terms of independence is Gaussian Random Numbers Generator (Chi-Square value = 14.400) as shown in *Table 4.1*. This is because it produces the highest Chi-Square Value from the test. This also means that numbers in the GRNG were more likely to depend on each other's than all the others random numbers generators. *Table 4.2* represents the descriptive statistics of the analysis (Standard Deviation, Mean, etc.). Standard deviation of numbers increases with decreases normality and vice versa. This means the higher the standard deviation value the higher the deviation from the normal and vice versa. Therefore Fibonacci Random Number Generator is more deviated from the normal distribution than all the other generators while Gaussian Random Number Generator produces numbers that obeys the normal distribution than all the generators compared with as shown in *Table 4.2*.

Table 4.1

The Chi-Square Test for Independence of PRNGs

Statistics		Specific Range Random Numbers Generator	Secure Random Numbers Generator	Gaussian Random Numbers Generator	Fibonacci Random Numbers Generator
Chi-Square		4.500	7.380	14.400	0.000
Df		94	90	87	99
Asymp.Sig.		1.000	1.000	1.000	1.000
Monte carlo Sig.		1.000	1.000	1.000	1.000
	Lower Bound	0.977	0.977	0.977	0.977
90% Confidence Interval	Upper Bound	1.000	1.000	1.000	1.000

Table 4.2

The descriptive statistics of the Random Numbers Generators

Generators	N	Mean	Std. Deviation	Minimum	Maximum
Specific Range Random Number Generator	100	479.07	302.50	2	988
Secure Random Number Generator	100	512.32	284.47	34	958
Gaussian Random Number Generator	100	5.08	68.51	-121	126
Fibonacci Random Number Generator	100	-90142675.73	975930540	-2092787285	2.E9

4.4 The Kolmogorov-Smirnov Test for Uniformity of random numbers

In Kolmogorov-Smirnov, if the more the Z-Value falls in between the lower and upper bound of the Monte carlo Significance.(2-tailed) Value, the more uniform the numbers generated by the random number is and vice versa. From *Table 4.3*, the Secure Random Number Generator and Gaussian Random Number Generator produces more uniform numbers than its counterparts Fibonacci and Specific Range Random Number Generators. This is because Secure Random Number Generator and Gaussian Random Number Generator produces Z- Values that fall closer between the lower and upper bounds of Z-Values, while Fibonacci and Specific Range Random Number Generators produces Z-Values that fall outside the range of lower and upper bounds Z-Values as shown in *Table 4.3*. The factors responsible for decision also showed that factors tend to give more considerations to Fibonacci Random Numbers Generators (K-S value = 2.020) than the other generators compared with in this Thesis. This therefore makes Fibonacci random Numbers Generator less uniform than the other generators: Secure Random Numbers Generator (K-S value = 0.701),

Specific Range Random Numbers Generator (K-S value = 1.027), Gaussian Random Numbers Generator (K-S value = 0.725) as shown in *Table 4.3*.

However, the lesser the uniformity of numbers produced, the more independent numbers are and vice versa. This accounts for why Fibonacci Random Number Generator is most independent of the four generators as shown in (*section 4.3*).

Table 4.3
The Kolmogorov-Smirnov Test for Uniformity of random numbers

Statistics			Specific Range Random Numbers Generator	Secure Random Numbers Generator	Gaussian Random Numbers Generator	Fibonacci Random Numbers Generator		
N			100	100	100	100		
Uniform Parameters			Minimum	2	34	-121	-	
			Maximum	988	958	126	2092787285	
			Absolute	0.103	0.070	0.072	2144908973	
Most Differences			Extreme					
			Positive	0.103	0.045	0.072	0.202	
			Negative	-0.039	-0.070	-0.060	-0.164	
Kolmogorov-Smirnov Z				1.027	0.701	0.725	2.020	
Asymp.Sig. (2-tailed)				0.243	0.709	0.670	0.001	
Sig.				0.300	0.670	0.640	0.000	
Monte carlo.Sig.(2-tailed)			90%Confidence Interval —	Lower Bound	0.225	0.593	0.561	0.000
				Upper Bound	0.375	0.747	0.719	0.23

4.5 Summary of Findings and Discussions

According to the analysis of the factors responsible for decision to be taken on the effectiveness of Pseudo Random Numbers Generators, it is found out that the random number generator with the highest number of repeated numbers was Gaussian Random Numbers Generator, followed by Secure Random Numbers Generators and then Specific Range Random Numbers Generators, while Fibonacci Random Numbers Generator has no repeated numbers in the random numbers. The repetition of numbers in the other generators like the Secure Random Numbers Generator and Specific Range Random Numbers Generator were found in between. This means that numbers were more likely to be random using Fibonacci Random Numbers Generator than the other generators compared with.

Again, the analysis of the factors responsible for randomness in a random numbers generators showed that factors tend to give much considerations to Fibonacci Random Numbers Generator than the other generators compared with. This is because the Chi-Square analysis for the independence of numbers in the generators showed that numbers were more independent to each other in the Fibonacci Random Numbers Generator than the other generators under studied.

However, the worst generator in terms of independence was Gaussian Random Numbers Generator (GRNG). This means that numbers in the GRNG were more likely to depend on each other's than all the others random numbers generators.

Furthermore, the test for uniformity also shows that factors tend to give more considerations to Secure Random Number Generator and Gaussian Random Number Generator than the other generators compared with in this Thesis. This therefore

makes Fibonacci random Numbers Generator and Specific Range Random Numbers Generator more than the other generators.

A good PRNG will produce a sequence of numbers that cannot be easily guessed or determined by an adversary. The general assumption is that the opponent knows the algorithm being used. This is usually referred to as Kerckhoff's principle (Yehuda L., 2006). This assertion is evidenced in Fibonacci Random Numbers Generator than the other Generators compared with. This is because the Chi-Square analysis for the independence of numbers in the generators showed that numbers were more independent to each other in the Fibonacci Random Numbers Generator than the other generators under studied.

An additional reason for randomness of a pseudo random numbers generator is the basis for good pseudo random number generators (Andrew R. et al (2010)). Every correlations and trends in the generators should be hidden. Statistically these can be proven. The statistical component of a pseudo random numbers generators shows that every value or results over a given range has equal chance of incidence. Every number has a probability of $\frac{1}{2}$ in the range of $[0, 1]$. Every couple of numbers used will have an occurrence of $\frac{1}{4}$ and a frequency of $\frac{1}{8}$ for every triplet. That is then basis for the numbers generated are not biased and consecutive output are not correlated (James A., et al (2003)). The mean and variance computation must also be included in the statistical randomness test. If the output for the Mean is closer to 0.5 and that of the variance is closer to 0.08, then the pseudo random numbers are homogeneously.

Again, NIST (2010) stated that, Frequency (Monobits) Test is also another test for efficiency. The focus of the test is the proportion of zeroes and ones for the

entire sequence. The purpose of this test is to determine whether that number of ones and zeros in a sequence are approximately the same as would be expected for a truly random sequence. The test assesses the closeness of the fraction of ones to $\frac{1}{2}$, that is, the number of ones and zeroes in a sequence should be about the same. Test for Frequency within a Block: The focus of the test is the proportion of zeroes and ones within M-bit blocks. This test also determines whether the frequency of one's is an M-bit block is approximately $M/2$. In the analysis of the factors responsible for decision to be taken on the random number generators revealed that Fibonacci Random Numbers Generator produces random numbers that have less number of ones and zeroes to be about the same than all the others random numbers generators and hence makes numbers generated by this generator more independent of each other.

4.6 Conclusion

The discussions of the findings indicate that Gaussian Random Numbers Generator and Secure Random Numbers Generators Algorithms are the more uniform, and then in terms of independence of numbers, it is the Fibonacci Random Numbers Generator followed by Specific Range Random Number Generator that numbers more independent than their counterpart generators compared.

CHAPTER FIVE

CONCLUSIONS AND RECOMMENDATIONS

5.1 Conclusions

As stated earlier, this thesis is intended to compare the efficiency of Pseudo-Random Numbers Generators (PRNGs) by testing for the repetition of numbers in Generators, testing for the uniformity of numbers in the Generators algorithms, and testing for independence of numbers generated by the Pseudo-Random Numbers Generators algorithms.

In the test for uniformity of PRNGs, it showed that numbers generated using Fibonacci Random Numbers Generators (K-S value = 2.020) were more uniform than the other generators compared with: Secure Random Numbers Generator (K-S value = 0.701), Specific Range Random Numbers Generator (K-S value = 1.027), Gaussian Random Numbers Generator (K-S value = 0.725) (section 4.3).

The test for independence of PRNGs also revealed that numbers in the Fibonacci Random Numbers Generator were more independent to each other than the other Generators compared with. This is because the Chi-Square analysis for the independence in Fibonacci Random Numbers Generator (Chi-Square Value = 0.000) was independence than the other generators under studied.

However, the worst generator in terms of independence was Gaussian Random Numbers Generator (Chi-Square value = 14.400) (section 4.3). This means that numbers in the GRNG were more likely to depend on each other's than all the others random numbers generators.

5.2 Recommendations

According to Elizabeth A. Smith (2001) a highly valued asset is loss when companies lose their important and confidential data in enterprise applications software. Information security is also necessary to ensure that the data, transactions, communications or documents (electronic or physical) are genuine, data cannot be modified undetectably, and prevent the disclosure of information to unauthorized individuals or systems (Akintunde M. Yinka , 2011). It is also important for authenticity to validate that both parties involved are who they claim they are. This makes data available to users according to their access right and ensuring that it cannot be modified by unauthorized users. As such the security of the Cryptographic System is needed to be transparent for Companies for key exchange, digital signatures, or encryption of small blocks of data.

How random the keys are, provide information security in cryptographic applications, which protects data access by unauthorized users, and that depends on the independence of numbers in the PRNGs algorithms used (M. Jayakumar &T. Christopher, 2012). This will increase the security of enterprise software and company's revenue and the overall output performance of the organisation. The following are recommended:

- **Software developers:** Software developers are made aware of most effective PRNG to be used for keys generations, keys handling and other practices that are prescribed by the efficient PRNGs use in the product or application that they develop. Furthermore, the standard needs to keep up with the latest advances in cryptography and in cryptanalysis in order to phase out weaker or broken algorithms.

- Process: The software development process must provide the framework for making the right decisions about implementing encryption, and this should be done using the right random numbers generator algorithm for its security.
- Again, during software development and testing, cryptographic security implementation needs to be reviewed and fully tested. This research ensures that every PRNG algorithm used in cryptographic security will pass certain statistical test.
- It is also recommended that by choosing the right PRNGs algorithm to be used for encryption algorithms in cryptographic systems will ensure maximum security of software data.
- Researchers: Researchers who are keen in the Cryptographic applications, to improve upon the weaknesses in security of cryptographic algorithms. Researchers can also research into comparative analysis of the efficiency of True Random Numbers Generators (TRNGs) algorithms.

References

- Akintunde, M. Y. (2011). *Data and Information Security*, Electrical/Electronic Engineering Department, Tower Polytechnic, Ibadan, Oyo State, Nigeria.
([http://www .hrmars.com/admin/pics/272.pdf](http://www.hrmars.com/admin/pics/272.pdf)), (accessed 2012-August 20).
- Andrea, R. & Salzburg, A. (2005). *Pseudorandom Number Generators For Cryptographic Applications*. Paris, London, pp-1-3.
- Andrew, R., Juan, S., James, N., Miles, S., Elaine, B., Stefan, L., et al. (2010).
“ *A Statistical Test Suite for Random and Pseudorandom Number Generators for Cryptographic Application*, Revision 1a, USA, Special Publication 800-22 (<http://csrc.nist.gov/publications/nistpubs/800-22-rev1a/SP800-22rev1a.pdf>),” (accessed 2012 September 1).
- AuthenTec Embedded Security Solutions (2010). “*The importance of True Randomness in Cryptography*,” (http://www.authentec.com/Portals/5/Documents/TRNG%20Whitepaper_91411.pdf), (accessed 2012 January 14).
- Bell Communications research (1996, September). “New threat model breaks crypto codes,” *Belcore press release*, Morristown,.
- Boneh, D., DeMillo, R., & Lipton, R.(2001). “On the importance of checking cryptographic protocols for faults,” *Journal of Cryptology*, vol. 14, no. 2, pp. 101-119.

- Carey-Smith, Mark T. and Nelson, Karen J. and May, Lauren J. (2007). Improving Information Security Management in Nonprofit Organisations with Action Research. In Valli, Craig and Woodward, Andrew, Eds. *Proceedings The 5th Australian Information Security Management Conference*, pages pp. 38-46, Perth, Western Australia.
- Chaitin, G. J.(2001). Exploring RANDOMNESS, *IBM Research*, Published by Springer-Verlag London, x + 164 pages, hardcover, ISBN 1-85233-4177.
- Chapelle, O., Vapnik, V., Bousquet, O., & Mukherjee, S.(2002). Choosing multiple parameters for support vector machines. *Machine Learning*, 46(1):131–159.
- Christophe, D., & Diethelm, W. (2003, September 20). “A note on random Number generation,” (<http://cran.r-project.org/web/packages/randtoolbox/vignettes/fullpres.pdf>), (accessed 2012 September 7, 2012).
- Dan, B.(2000). “Twenty years of attacks on the RSA cryptosystem,” 2000. (<http://crypto.stanford.edu/~dabo/papers/RSA-survey.pdf>), (accessed 2012 May 10).
- Diffie, W. & Hellman, M.E.(1976). New directions in cryptography, *IEEE transactions on Information theory*, vol. 22, issue. 6, pp: 644-654.
- Digital Signature Standard (DSS) (1994, May), *Federal Information Processing Standards Publication 186*.
- DOI:Mishra, S. & Das, M. (2010). *FPGA Based Random Number Generation for Cryptographic Applications*. National Institute of Technology, Rourkela, India.
- Eastlake, D., Schiller, J. & Crocker, S. (2005) “RFC 4086,” (<http://www.ietf.org/rfc/rfc4086.txt>), (accessed 2012 June 10).

- El Gmal, T.(1985). "*A public cryptosystem and a signature scheme based on discrete logarithms*," Proceedings of CRYPTO 84 on Advances in cryptology, Santa Barbara, California, United States, pp. 10-18.
- Elaine, B., & Allen, R., (2011). *Transitions: Recommendation for Transitioning the Use of Cryptographic Algorithms and Key .NIST Special Publication 800-131A*.
- Elizabeth, A. S.(2001).*The role of tacit and explicit knowledge in the workplace*, Journal Knowledge Management, Volume 5 , Number 4, 311-321 (http://www.uky.edu/~gmswan3/575/KM_roles.pdf), (accessed 2012 January 5).
- English, E. & Hamilton, S.(1996). "Network security under siege: the timing attack," *IEEE Computer*, vol. 29, pp. 95-97.
- Feistel, W. T., Don, C, Alan, K., Carl, M., Mike, M., Roy, A.,et al.(1993). "Federal information processing standards publication 46-2: *Data encryption standard (DES)*," (<http://www.itl.nist.gov/fipspubs/fip46-2.htm>.), (accessed 2012 August 20).
- Fischer, V., Aubert, A., Bernard, F., Valtchanov, B., Danger, J.-L. & Bochard, N. (2009), *True Random Number Generators in Configurable Logic Devices* University of Saint Etienne, *Laboratory Hubert Curien* CNRS UMR 5516
- Fisnik, H. (2011). "*Safe Internet Banking*," (<http://www.itknowledge24.com/blog/safe-internet-banking/>), (accessed 2012 June 10).
- Florin, R., & Alin, D. (2007).*Pseudo-Random Number Generation for Sketch-Based Estimations*, USA, University of Florida Press.
- Gary, C. K. (2012). "*An Overview of Cryptography*," (<http://www.garykessler.net/library/crypto.html>), (accessed 2012 August 10).

- Hani, M. K., Lin, T. S., & Nasir, S-H.(2000). "*FPGA implementation of RSA public-key cryptographic coprocessor*," Proceedings on TENCON 2000, vol. 3, pp.6-11.
- IEEE standard 1363-2000 (2000, January). *Standard specifications for public key cryptography: additional techniques*.
- James, A. H., Abdissa N., Michael D. , Edwardes, B., & Janet E. F., (2003). *Statistical Analysis of Correlated Data Using Generalized Estimating Equations: An Orientation*, American Journal of Epidemiology, USA, Volume 157, Issue 4 , pp. 364-375.
- Jayakumar, M., & Christopher,T. (2012). *Symmetric Algorithms Key Generate Using Static Ip Address*, International Journal of Advanced Research in Computer Science and Engineering, volume 2, issue 2,(http://www.ijarcsse.com/docs/papers/July2012/Volume_2_issue_7/V2I700121.pdf), (accessed 2012 August 12).
- John, H. (2006),Trinity College DUBLIN, *Annual Report 2006 – 2007*.
- Joye, M., Lenstra,A.K.,& Quisquater,J.-J.(1999). "Chinese Remaindering based cryptosystems in the presence of faults," *Journal of Cryptology*, vol. 12, no. 4, pp 241-245.
- Justin, A. (2012). *Progressing past poor Pseudo-randomness*, Cosmos Cluster 4,(<http://cosmos.ucdavis.edu/archives/2012/cluster4/Adsuaara,%20Justin.pdf>), (accessed 2012 August 14).
- Koc, C. K. (1994). "High-speed RSA implementations," *Technical notes TR 201*, RSA Security Inc.
- Koc, C.K.(1995). "RSA hardware implementation," *Technical notes TR 801*, RSA Security Inc..

- Kocher, P. (1996). "Timing attacks on implementations of Diffie-Hellman, RSA, DSS, and other systems," CRYPTO' 96, Springer-Verlag, pp. 104-113.
- Kocher, P., Jaffe, J., & Jun, B. (1999) "Differential power analysis," *Proceedings Of CRYPTO'99*, pp. 388-397, Santa Barbara, CA, USA.
- Korneru, P. (1994) "A systolic, linear-array multiplier for a class of right-shift algorithms" *IEEE Trans. Computer Arithmetic*, vol. 43, pp. 892-898.
- Krishnamurthy, A., Tang, Y., Xu, C., & Wang, Y. (2003, April). "An efficient implementation of multi-prime RSA on dsp processor," *IEEE Int. Con. on Acoustics, Speech, & Signal Processing*, vol. 2, pp 413-416, Hongkong, China.
- Lai, X., & Massey, J. (1991). *A proposal for a new block encryption standard, Proceedings of Eurocrypt advances in Cryptology'90*, Springer-Verlag vol. 473, Berlin.
- Mark, H. (2006). "Generating random numbers in parallel," (<http://www.cs.berkeley.edu/~mhoemmen/cs194/Tutorials/prng.pdf>), (accessed 2012-May 10).
- Marsaglia, G. (2011, October 4). "Random Number CDROM including the Diehard Battery of Tests of Randomness," (<http://www.stat.fsu.edu/pub/diehard/>), (accessed 2012 August 27).
- Menezes, A., Van, P. O., & Vanstone, S. (Update 2011), *Handbook of applied cryptography*, CRC press.
- Mike, H., Paul, K., Mark, E. M. (2012). *Analysis of Intel's Ivy Bridge Digital Random Number Generator*, (http://www.cryptography.com/public/pdf/Intel_TRNG_Report_20120312.pdf), (accessed 2012 August 20.)

Mohammed, A. ,& Annapurna, P., P.(2012,February).*Implementing a secure key issuing scheme for communication in p2p networks*. M.S.Ramiah Institute of Technology,Department of Computer Science and Engineering, Bangalore- 560054, India.

Montgomery, P.L. (1998). "*Modular multiplication without trial division*," Mathematics of Computation, vol. 44, pp. 519-521.

Peter, G. N. (1997), Principal Scientist, *Computer Science Laboratory*, SRI International,MenloParkCA94025-3493,1-650-859-2375 (<http://www.csl.sri.com/neumann.html>), (accessed 2010 November 1).

Quisquater,J.-J.& Couvreur, C.(1982) "Fast decipherment algorithm for RSA public-key cryptosystem," *Electronic Letters*, vol. 18, no. 21, pp 905-907.

Rivest, R. L., Shamir, A., & L. Adleman, L.(1978). *A method for obtaining digital signatures and public-key cryptosystem*, Communications of the ACM, vol. 21, no. 2, pp.120-126.

Robert, S., & Corey, C. (2009). "Preserving identities: protecting personal identifying information through enhanced privacy policies and laws, lengths ,*Information Technology Laboratory* ,"(http://www.albanylaw-journal.org/files/19.1/sprague_format_dpl_ys.pdf), (accessed 2012 July 18).

RSA laboratory bulletin number 13. (2000, April). "A cost-based security analysis of symmetric and asymmetric key lengths," (<http://www.rsasecurity.com/rsalabs/node.asp?id=2088>.), (accessed 2012 July 12)

RSA Security Inc.(2000). *Crypto FAQ*.(4th ed, pp 8-45), Cambridge Center,USA.

Claude E. S. (2012, January 31). Communication Theory of Secrecy Systems. *Bell System Technical Journal*, vol. 28(4), pp. 656-715, 1949.

RSA Security Inc.,(2000).*Crypto FAQ*: Chapter 6: Laws concerning Cryptography, 6.3. Patents on cryptography.(4th ed, pp 174-197). Cambridge Center, USA.

RSA Security Inc.(2000, July). "PKCS #1 v2.0 amendment 1: Multi-prime RSA,". (<ftp://ftp.rsasecurity.com/pub/pkcs/pkcs-1/pkcs-1v2-0a1.pdf>.), (accessed 2012 June 14).

San, F. (2004). *Diploma Thesis A Random Number Generator Test Suite for the C++ Standard*.

Schindler, W.(2000). "A timing attack against RSA with the Chinese Remainder Theorem," *Proceedings of Cryptographic Hardware and Embedded Systems*, pp. 109-124.

Shamir, A. (1997, May). "How to check modular exponentiation," Eurocrypt 97.

Shamir, A.(1999, November). "Method and apparatus for protecting public key schemes from timing and fault attacks," US patent 5991415.

Thomas, B. (2006). "Analysis of a strong Pseudo Random Number Generator," (<http://www.suse.de/~thomas/papers/random-analysis.pdf>), (accessed February 2012).

Thomas, S.(2000). Messages, "Power analysis attack countermeasures and their weaknesses," *Security Technology Research Laboratory*.

Vander, A.J. S., Jack D. (2008). "Overview of Recent Supercomputers," (https://computing.llnl.gov/tutorials/parallel_comp/), (accessed- 2012 September 2).

- Wiener, M.J. (1990) "Cryptanalysis of short RSA secret exponents," *IEEE Transactions on Information Theory*, vol: 36, Issue: 3, pp: 553-558.
- Wolfram Research, Inc.(2008), *Random Numbers Generation*, United States of America.
- Wolfram, S.(2002). *A New Kind of Science*. Wolfram Media.pp. 975–976. ISBN 1-57955-008-8.
- wordiq.com.(2004,December 9). "History of cryptography," (http://www.wordiq.com/definition/History_of_cryptography.), (accessed 2011 January 2)
- Wu, C.-H. , Hong, J.-H., & Wu, C.-W.(2001). *RSA cryptosystem design based the Chinese Remainder Theorem*," Proceedings of the ASP-DAC, pp: 391–395.
- Yang, C.-C., Chang, T.-S., & Jen C.-W.(1998). "A new RSA cryptosystem hardware design based on Montgomery's algorithm," *IEEE Transactions on Circuits and Systems II: Analog and Digital Signal Processing*, vol:45, Issue: 7, pp: 908-913.
- Yehuda, L. (2006). *Introduction to Cryptography*, 89-656, (<http://u.cs.biu.ac.il/~lindell/89-656/Intro-to-crypto-89-656.pdf>), (accessed 2012 August 1).
- Yen,S., Kim,S., Lim,S., & Moon, S.(2003). "RSA speedup with Chinese Remainder Theorem immune against hardware fault attack," *IEEE Transactions on computers*, vol. 52, pp. 461-472.
- YU,L.R.(2002) "The generalization of the Chinese Remainder Theorem," *Acta Mathematica Sinica, English Series*, vol. 18, pp. 532-538.
- Yuval,I.,(2011). *Theory of Cryptography*, 8th Theory of Cryptography Conference,TCC 2011 Providence,RI,USA.

Appendices

Appendix I

Java Codes for Pseudo-Random Number Generators (PRNGs) Algorithms

Fibonacci Random Numbers Generator Algorithm

```
/**
 * This program prints out the first 20 numbers in the Fibonacci sequence. Each
 * term is formed by adding together the previous two terms in the sequence,
 * starting with the terms 1 and 1.
 *
 * By Christopher A Abilimi
 */
public class Fibonacci {
    public static void main(String[] args) {
        int n0 = 1, n1 = 1, n2; // Initialize variables
        System.out.print(n0 + " " + // Print first and second terms
            n1 + " "); // of the series
        for (int i = 0; i < 18; i++) { // Loop for the next 18 terms
            n2 = n1 + n0; // Next term is sum of previous two
            System.out.print(n2 + " "); // Print it out
            n0 = n1; // First previous becomes 2nd previous
            n1 = n2; // And current number becomes previous
        }
        System.out.println(); // Terminate the line
    }
}
```

LIBRARY
KWAME NKRUMAH
UNIVERSITY OF SCIENCE & TECHNOLOGY
KUMASI

Secure Random Generator Algorithm

```
/*
 * A Program to generate random numbers using Secure Random Generator
 */
package generator;
import java.util.Random;
```



```

/**
 *
 * @author Christopher A Abilimi
 */
public class Main {

    /**
     * @param args the command line arguments
     */
    public static void main(String[] args) {
        // TODO code application logic here
    }

    /*
     * To change this template, choose Tools | Templates
     * and open the template in the editor.
     */

    log("Generating 10 random integers in range 0..99.");

    //note a single Random object is reused here
    Random randomGenerator = new Random();
    for (int idx = 1; idx <= 100; ++idx){
        int randomInt = randomGenerator.nextInt(1000);
        log("Generated : " + randomInt);
    }

    log("Done.");
}

private static void log(String aMessage){
    System.out.println(aMessage);
}
}

```

Specific Range Generator

```

*
* A program to generate random numbers using the Specific Range Generator
*
*/
package specificrange;
import java.util.Random;
/**
 *
 * @author Christopher A Abilimi
 */
public class Main {
    /**
     * @param args the command line arguments
     */
}

```



```

    public static void main(String[] args) {
        // Specific Range Generator.
        /** Generate random integers in a certain range. */
        log("Generating random integers in the range 1..1000.");
        int START = 1;
        int END = 1000;
        Random random = new Random();
        for (int idx = 1; idx <= 1000; ++idx){
            showRandomInteger(START, END, random);
        }
        log("Done.");
    }
    private static void showRandomInteger(int aStart, int aEnd, Random aRandom){
        if ( aStart > aEnd ) {
            throw new IllegalArgumentException("Start cannot exceed End.");
        }
        //get the range, casting to long to avoid overflow problems
        long range = (long)aEnd - (long)aStart + 1;
        // compute a fraction of the range, 0 <= frac < range
        long fraction = (long)(range * aRandom.nextDouble());
        int randomNumber = (int)(fraction + aStart);
        log("Generated : " + randomNumber);
    }
    private static void log(String aMessage){
        System.out.println(aMessage);
    }
}

// TODO code application logic here

```

Gaussian (Normal) Distribution Generator

```

/**
 * A Program to generate random numbers using Gaussian Distribution Generator
 *
 */
package Gaussian;
import java.util.Random;
/**
 *
 * @author Christopher A Abilimi
 */
public class Main {
    /**
     * @param args the command line arguments
     */
    public static void main(String[] args) {
        // TODO code application logic here
    }
}

```



```

// Create a new instance or object of class Random
Random util_Random = new Random();
// Generate a pseudorandom number between 0.0 and 1.0
// (Note: number will in the range of 0.0 to 1.0, but never 1.0)
double util_Random_Double = util_Random.nextDouble();
// Returns the next pseudorandom, Gaussian ("normally") distributed
// double value with mean 0.0 and standard deviation 1.0
double util_Random_Gaussian = util_Random.nextGaussian();

// Generate a number between 1.40129846432481707e-45
// to 3.40282346638528860e+38 (positive or negative)
float util_Random_Float = util_Random.nextFloat();

// Generate a number between -9,223,372,036,854,775,808
// to +9,223,372,036,854,775,807
long util_Random_Long = util_Random.nextLong();

// Generate a number between -2,147,483,648 to 2,147,483,647
int util_Random_Integer = util_Random.nextInt();

// Generate an integer from a specified range (if your pass
// in n as a parameter you numbers will range from 0 to n-1)
int util_Random_Integer_from_n = util_Random.nextInt(1000);
// Generate either true or false pseudorandomly
boolean util_Random_Bool = util_Random.nextBoolean();
// Create and initialize an array of 5 bytes
byte[] bytes = new byte[1000];
// Populate the array with 5 randomly generated bytes
// Note: byte is a value that ranges from -128 to 127
util_Random.nextBytes( bytes );
// Creates a java.util.Random object and calls
// the nextDouble() method
double math_Random = Math.random();

// Display all the results from above methods
System.out.println( "Math.random() -> " + math_Random );
System.out.println( "nextDouble() -> " + util_Random_Double );
System.out.println( "nextGaussian() -> " + util_Random_Gaussian );
System.out.println( "nextFloat() -> " + util_Random_Float );
System.out.println( "nextLong() -> " + util_Random_Long );
System.out.println( "nextInt() -> " + util_Random_Integer );
System.out.println( "nextInt(n) -> " + util_Random_Integer_from_n );
System.out.println( "nextBoolean() -> " + util_Random_Bool );
System.out.println( "nextBytes(bytes) -> " );
for( int i = 0; i < bytes.length; i++ )
    System.out.println( "Byte " + (i+1) + ": " + bytes[i] );
}
}
}
}

```


Appendix II

Chi-Square Test

Frequencies

Specific Range Random Number Generator			
	Observed N	Expected N	Residual
2	1	1.1	.0
9	1	1.1	.0
19	1	1.1	.0
22	1	1.1	.0
41	2	1.1	.9
56	1	1.1	.0
76	1	1.1	.0
78	1	1.1	.0
80	1	1.1	.0
86	1	1.1	.0
92	1	1.1	.0
96	1	1.1	.0
100	1	1.1	.0
111	1	1.1	.0
116	1	1.1	.0
123	1	1.1	.0

125	1	1.1	.0
128	1	1.1	.0
134	1	1.1	.0
144	1	1.1	.0
151	1	1.1	.0
153	1	1.1	.0
163	1	1.1	.0
166	2	1.1	.9
167	1	1.1	.0
177	1	1.1	.0
217	1	1.1	.0
220	1	1.1	.0
227	1	1.1	.0
248	1	1.1	.0
249	1	1.1	.0
259	1	1.1	.0
262	1	1.1	.0
268	1	1.1	.0
282	1	1.1	.0
336	1	1.1	.0
361	1	1.1	.0
365	1	1.1	.0
416	1	1.1	.0

430	1	1.1	.0
455	1	1.1	.0
456	2	1.1	.9
458	1	1.1	.0
461	1	1.1	.0
463	1	1.1	.0
495	1	1.1	.0
512	1	1.1	.0
524	1	1.1	.0
526	1	1.1	.0
549	1	1.1	.0
554	1	1.1	.0
564	1	1.1	.0
567	1	1.1	.0
587	1	1.1	.0
589	1	1.1	.0
601	1	1.1	.0
603	1	1.1	.0
604	1	1.1	.0
605	1	1.1	.0
618	1	1.1	.0
620	1	1.1	.0
634	1	1.1	.0

655	1	1.1	.0
656	1	1.1	.0
663	1	1.1	.0
679	1	1.1	.0
702	1	1.1	.0
705	1	1.1	.0
714	1	1.1	.0
719	1	1.1	.0
739	2	1.1	.9
742	1	1.1	.0
751	2	1.1	.9
756	1	1.1	.0
762	1	1.1	.0
766	1	1.1	.0
769	1	1.1	.0
782	1	1.1	.0
827	1	1.1	.0
834	1	1.1	.0
849	1	1.1	.0
852	1	1.1	.0
856	1	1.1	.0
873	1	1.1	.0
877	1	1.1	.0

895	1	1.1	.0
901	1	1.1	.0
902	1	1.1	.0
917	1	1.1	.0
930	1	1.1	.0
952	1	1.1	.0
974	1	1.1	.0
983	1	1.1	.0
984	1	1.1	.0
988	1	1.1	.0
Total	100		

Secure Random Number Generator			
	Observed N	Expected N	Residual
34	1	1.1	.0
45	1	1.1	.0
46	1	1.1	.0
55	1	1.1	.0
72	1	1.1	.0
76	2	1.1	.9
80	1	1.1	.0
86	1	1.1	.0

101	1	1.1	.0
123	2	1.1	.9
136	2	1.1	.9
159	1	1.1	.0
168	1	1.1	.0
171	1	1.1	.0
178	1	1.1	.0
194	1	1.1	.0
203	1	1.1	.0
205	1	1.1	.0
206	1	1.1	.0
218	1	1.1	.0
222	1	1.1	.0
230	1	1.1	.0
236	1	1.1	.0
244	1	1.1	.0
251	1	1.1	.0
294	1	1.1	.0
299	1	1.1	.0
334	1	1.1	.0
359	1	1.1	.0
362	1	1.1	.0
396	1	1.1	.0

397	2	1.1	.9
405	1	1.1	.0
427	1	1.1	.0
436	1	1.1	.0
444	1	1.1	.0
447	1	1.1	.0
450	1	1.1	.0
459	1	1.1	.0
461	1	1.1	.0
469	1	1.1	.0
479	1	1.1	.0
488	1	1.1	.0
500	1	1.1	.0
503	1	1.1	.0
512	1	1.1	.0
524	1	1.1	.0
552	2	1.1	.9
555	1	1.1	.0
597	1	1.1	.0
607	1	1.1	.0
609	2	1.1	.9
610	1	1.1	.0
611	1	1.1	.0

613	1	1.1	.0
632	1	1.1	.0
638	1	1.1	.0
648	1	1.1	.0
663	1	1.1	.0
665	1	1.1	.0
670	1	1.1	.0
672	1	1.1	.0
675	1	1.1	.0
676	1	1.1	.0
686	1	1.1	.0
700	1	1.1	.0
715	1	1.1	.0
717	1	1.1	.0
751	1	1.1	.0
766	2	1.1	.9
784	1	1.1	.0
810	1	1.1	.0
813	1	1.1	.0
825	1	1.1	.0
844	1	1.1	.0
849	1	1.1	.0
859	1	1.1	.0

860	1	1.1	.0
864	1	1.1	.0
866	1	1.1	.0
881	1	1.1	.0
893	1	1.1	.0
895	1	1.1	.0
897	1	1.1	.0
900	1	1.1	.0
919	1	1.1	.0
923	1	1.1	.0
933	2	1.1	.9
945	1	1.1	.0
952	1	1.1	.0
958	2	1.1	.9
Total	100		

Gaussian Random Number Generator			
	Observed N	Expected N	Residual
-121	1	1.1	-.1
-115	1	1.1	-.1
-106	1	1.1	-.1
-104	1	1.1	-.1

-102	1	1.1	-.1
-100	1	1.1	-.1
-95	1	1.1	-.1
-89	2	1.1	.9
-87	1	1.1	-.1
-82	1	1.1	-.1
-79	1	1.1	-.1
-77	2	1.1	.9
-76	1	1.1	-.1
-73	1	1.1	-.1
-69	1	1.1	-.1
-68	1	1.1	-.1
-67	1	1.1	-.1
-66	1	1.1	-.1
-64	1	1.1	-.1
-57	1	1.1	-.1
-52	1	1.1	-.1
-51	2	1.1	.9
-49	1	1.1	-.1
-47	1	1.1	-.1
-44	1	1.1	-.1
-43	1	1.1	-.1
-42	1	1.1	-.1

-41	1	1.1	-.1
-40	1	1.1	-.1
-38	1	1.1	-.1
-36	1	1.1	-.1
-33	1	1.1	-.1
-29	1	1.1	-.1
-27	2	1.1	.9
-25	1	1.1	-.1
-19	1	1.1	-.1
-18	1	1.1	-.1
-15	1	1.1	-.1
-12	1	1.1	-.1
-11	1	1.1	-.1
-9	1	1.1	-.1
-8	1	1.1	-.1
-3	2	1.1	.9
2	1	1.1	-.1
5	1	1.1	-.1
8	1	1.1	-.1
11	1	1.1	-.1
12	1	1.1	-.1
14	1	1.1	-.1
15	1	1.1	-.1

18	1	1.1	-.1
19	2	1.1	.9
20	1	1.1	-.1
21	2	1.1	.9
23	1	1.1	-.1
25	1	1.1	-.1
29	1	1.1	-.1
30	4	1.1	2.9
33	1	1.1	-.1
34	1	1.1	-.1
42	1	1.1	-.1
43	1	1.1	-.1
49	1	1.1	-.1
53	1	1.1	-.1
54	1	1.1	-.1
58	1	1.1	-.1
63	1	1.1	-.1
70	1	1.1	-.1
74	1	1.1	-.1
76	1	1.1	-.1
85	1	1.1	-.1
86	1	1.1	-.1
89	1	1.1	-.1

90	1	1.1	-.1
91	1	1.1	-.1
92	1	1.1	-.1
94	1	1.1	-.1
99	1	1.1	-.1
101	1	1.1	-.1
102	1	1.1	-.1
104	1	1.1	-.1
107	1	1.1	-.1
112	1	1.1	-.1
114	1	1.1	-.1
123	2	1.1	.9
124	1	1.1	-.1
125	1	1.1	-.1
126	2	1.1	.9
Total	100		

Fibonacci Random Number Generator			
	Observed N	Expected N	Residual
-2092787285	1	1.0	.0
-2079590721	1	1.0	.0
-2015728079	1	1.0	.0

-1958435240	1	1.0	.0
-1869596475	1	1.0	.0
-1781832971	1	1.0	.0
-1709589543	1	1.0	.0
-1691007710	1	1.0	.0
-1581614984	1	1.0	.0
-1507123775	1	1.0	.0
-1418756969	1	1.0	.0
-1408458269	1	1.0	.0
-1323752223	1	1.0	.0
-1289228135	1	1.0	.0
-1262539787	1	1.0	.0
-1230842041	1	1.0	.0
-1109825406	1	1.0	.0
-1070442683	1	1.0	.0
-1055680967	1	1.0	.0
-1036647147	1	1.0	.0
-980107325	1	1.0	.0
-945834654	1	1.0	.0
-944741150	1	1.0	.0
-889489150	1	1.0	.0
-811192543	1	1.0	.0
-798870975	1	1.0	.0

-660827267	1	1.0	.0
-433386095	1	1.0	.0
-401779575	1	1.0	.0
-298632863	1	1.0	.0
-285007387	1	1.0	.0
-188547518	1	1.0	.0
-90618175	1	1.0	.0
1	1	1.0	.0
2	1	1.0	.0
3	1	1.0	.0
5	1	1.0	.0
8	1	1.0	.0
13	1	1.0	.0
21	1	1.0	.0
34	1	1.0	.0
55	1	1.0	.0
89	1	1.0	.0
144	1	1.0	.0
233	1	1.0	.0
377	1	1.0	.0
610	1	1.0	.0
987	1	1.0	.0
1597	1	1.0	.0

2584	1	1.0	.0
4181	1	1.0	.0
6765	1	1.0	.0
10946	1	1.0	.0
17711	1	1.0	.0
28657	1	1.0	.0
46368	1	1.0	.0
75025	1	1.0	.0
121393	1	1.0	.0
196418	1	1.0	.0
317811	1	1.0	.0
514229	1	1.0	.0
832040	1	1.0	.0
1346269	1	1.0	.0
2178309	1	1.0	.0
3524578	1	1.0	.0
5702887	1	1.0	.0
9227465	1	1.0	.0
14930352	1	1.0	.0
24157817	1	1.0	.0
39088169	1	1.0	.0
63245986	1	1.0	.0
102334155	1	1.0	.0

165580141	1	1.0	.0
267914296	1	1.0	.0
363076002	1	1.0	.0
368225352	1	1.0	.0
375819880	1	1.0	.0
433494437	1	1.0	.0
511172301	1	1.0	.0
512559680	1	1.0	.0
572466946	1	1.0	.0
695895453	1	1.0	.0
696897233	1	1.0	.0
701408733	1	1.0	.0
708252800	1	1.0	.0
764848393	1	1.0	.0
885444751	1	1.0	.0
887448560	1	1.0	.0
1073992269	1	1.0	.0
1134903170	1	1.0	.0
1412467027	1	1.0	.0
1582341984	1	1.0	.0
1640636603	1	1.0	.0
1642909629	1	1.0	.0
1776683621	1	1.0	.0

1820529360	1	1.0	.0
1836311903	1	1.0	.0
1845853122	1	1.0	.0
2118290601	1	1.0	.0
2144908973	1	1.0	.0
Total	100		

Test Statistics				
	Specific Range Random Number Generator	Secure Random Number Generator	Gaussian Random Number Generator	Fibonacci Random Number Generator
Chi-Square	4.500 ^a	7.380 ^b	14.400 ^c	.000 ^d
Df	94	90	87	99
Asymp. Sig.	1.000	1.000	1.000	1.000
a. 95 cells (100.0%) have expected frequencies less than 5. The minimum expected cell frequency is 1.1.				
b. 91 cells (100.0%) have expected frequencies less than 5. The minimum expected cell frequency is 1.1.				
c. 88 cells (100.0%) have expected frequencies less than 5. The minimum expected cell frequency is 1.1.				
d. 100 cells (100.0%) have expected frequencies less than 5. The minimum expected cell frequency is 1.0.				

Appendix III

Frequency Test for Random Numbers Generators

Statistics					
		Specific Range Random Number Generator	Secure Random Number Generator	Gaussian Random Number Generator	Fibonacci Random Number Generator
N	Valid	100	100	100	100
	Missing	0	0	0	0

Frequency Tables

Specific Range Random Number Generator					
		Frequency	Percent	Valid Percent	Cumulative Percent
Valid	2	1	1.0	1.0	1.0
	9	1	1.0	1.0	2.0
	19	1	1.0	1.0	3.0
	22	1	1.0	1.0	4.0
	41	2	2.0	2.0	6.0
	56	1	1.0	1.0	7.0

76	1	1.0	1.0	8.0
78	1	1.0	1.0	9.0
80	1	1.0	1.0	10.0
86	1	1.0	1.0	11.0
92	1	1.0	1.0	12.0
96	1	1.0	1.0	13.0
100	1	1.0	1.0	14.0
111	1	1.0	1.0	15.0
116	1	1.0	1.0	16.0
123	1	1.0	1.0	17.0
125	1	1.0	1.0	18.0
128	1	1.0	1.0	19.0
134	1	1.0	1.0	20.0
144	1	1.0	1.0	21.0
151	1	1.0	1.0	22.0
153	1	1.0	1.0	23.0
163	1	1.0	1.0	24.0
166	2	2.0	2.0	26.0
167	1	1.0	1.0	27.0
177	1	1.0	1.0	28.0
217	1	1.0	1.0	29.0
220	1	1.0	1.0	30.0

227	1	1.0	1.0	31.0
248	1	1.0	1.0	32.0
249	1	1.0	1.0	33.0
259	1	1.0	1.0	34.0
262	1	1.0	1.0	35.0
268	1	1.0	1.0	36.0
282	1	1.0	1.0	37.0
336	1	1.0	1.0	38.0
361	1	1.0	1.0	39.0
365	1	1.0	1.0	40.0
416	1	1.0	1.0	41.0
430	1	1.0	1.0	42.0
455	1	1.0	1.0	43.0
456	2	2.0	2.0	45.0
458	1	1.0	1.0	46.0
461	1	1.0	1.0	47.0
463	1	1.0	1.0	48.0
495	1	1.0	1.0	49.0
512	1	1.0	1.0	50.0
524	1	1.0	1.0	51.0
526	1	1.0	1.0	52.0
549	1	1.0	1.0	53.0

554	1	1.0	1.0	54.0
564	1	1.0	1.0	55.0
567	1	1.0	1.0	56.0
587	1	1.0	1.0	57.0
589	1	1.0	1.0	58.0
601	1	1.0	1.0	59.0
603	1	1.0	1.0	60.0
604	1	1.0	1.0	61.0
605	1	1.0	1.0	62.0
618	1	1.0	1.0	63.0
620	1	1.0	1.0	64.0
634	1	1.0	1.0	65.0
655	1	1.0	1.0	66.0
656	1	1.0	1.0	67.0
663	1	1.0	1.0	68.0
679	1	1.0	1.0	69.0
702	1	1.0	1.0	70.0
705	1	1.0	1.0	71.0
714	1	1.0	1.0	72.0
719	1	1.0	1.0	73.0
739	2	2.0	2.0	75.0
742	1	1.0	1.0	76.0

751	2	2.0	2.0	78.0
756	1	1.0	1.0	79.0
762	1	1.0	1.0	80.0
766	1	1.0	1.0	81.0
769	1	1.0	1.0	82.0
782	1	1.0	1.0	83.0
827	1	1.0	1.0	84.0
834	1	1.0	1.0	85.0
849	1	1.0	1.0	86.0
852	1	1.0	1.0	87.0
856	1	1.0	1.0	88.0
873	1	1.0	1.0	89.0
877	1	1.0	1.0	90.0
895	1	1.0	1.0	91.0
901	1	1.0	1.0	92.0
902	1	1.0	1.0	93.0
917	1	1.0	1.0	94.0
930	1	1.0	1.0	95.0
952	1	1.0	1.0	96.0
974	1	1.0	1.0	97.0
983	1	1.0	1.0	98.0
984	1	1.0	1.0	99.0

	988	1	1.0	1.0	100.0
	Total	100	100.0	100.0	

Secure Random Number Generator					
		Frequency	Percent	Valid Percent	Cumulative Percent
Valid	34	1	1.0	1.0	1.0
	45	1	1.0	1.0	2.0
	46	1	1.0	1.0	3.0
	55	1	1.0	1.0	4.0
	72	1	1.0	1.0	5.0
	76	2	2.0	2.0	7.0
	80	1	1.0	1.0	8.0
	86	1	1.0	1.0	9.0
	101	1	1.0	1.0	10.0
	123	2	2.0	2.0	12.0
	136	2	2.0	2.0	14.0
	159	1	1.0	1.0	15.0
	168	1	1.0	1.0	16.0
	171	1	1.0	1.0	17.0
	178	1	1.0	1.0	18.0

194	1	1.0	1.0	19.0
203	1	1.0	1.0	20.0
205	1	1.0	1.0	21.0
206	1	1.0	1.0	22.0
218	1	1.0	1.0	23.0
222	1	1.0	1.0	24.0
230	1	1.0	1.0	25.0
236	1	1.0	1.0	26.0
244	1	1.0	1.0	27.0
251	1	1.0	1.0	28.0
294	1	1.0	1.0	29.0
299	1	1.0	1.0	30.0
334	1	1.0	1.0	31.0
359	1	1.0	1.0	32.0
362	1	1.0	1.0	33.0
396	1	1.0	1.0	34.0
397	2	2.0	2.0	36.0
405	1	1.0	1.0	37.0
427	1	1.0	1.0	38.0
436	1	1.0	1.0	39.0
444	1	1.0	1.0	40.0
447	1	1.0	1.0	41.0

450	1	1.0	1.0	42.0
459	1	1.0	1.0	43.0
461	1	1.0	1.0	44.0
469	1	1.0	1.0	45.0
479	1	1.0	1.0	46.0
488	1	1.0	1.0	47.0
500	1	1.0	1.0	48.0
503	1	1.0	1.0	49.0
512	1	1.0	1.0	50.0
524	1	1.0	1.0	51.0
552	2	2.0	2.0	53.0
555	1	1.0	1.0	54.0
597	1	1.0	1.0	55.0
607	1	1.0	1.0	56.0
609	2	2.0	2.0	58.0
610	1	1.0	1.0	59.0
611	1	1.0	1.0	60.0
613	1	1.0	1.0	61.0
632	1	1.0	1.0	62.0
638	1	1.0	1.0	63.0
648	1	1.0	1.0	64.0
663	1	1.0	1.0	65.0

665	1	1.0	1.0	66.0
670	1	1.0	1.0	67.0
672	1	1.0	1.0	68.0
675	1	1.0	1.0	69.0
676	1	1.0	1.0	70.0
686	1	1.0	1.0	71.0
700	1	1.0	1.0	72.0
715	1	1.0	1.0	73.0
717	1	1.0	1.0	74.0
751	1	1.0	1.0	75.0
766	2	2.0	2.0	77.0
784	1	1.0	1.0	78.0
810	1	1.0	1.0	79.0
813	1	1.0	1.0	80.0
825	1	1.0	1.0	81.0
844	1	1.0	1.0	82.0
849	1	1.0	1.0	83.0
859	1	1.0	1.0	84.0
860	1	1.0	1.0	85.0
864	1	1.0	1.0	86.0
866	1	1.0	1.0	87.0
881	1	1.0	1.0	88.0

893	1	1.0	1.0	89.0
895	1	1.0	1.0	90.0
897	1	1.0	1.0	91.0
900	1	1.0	1.0	92.0
919	1	1.0	1.0	93.0
923	1	1.0	1.0	94.0
933	2	2.0	2.0	96.0
945	1	1.0	1.0	97.0
952	1	1.0	1.0	98.0
958	2	2.0	2.0	100.0
Total	100	100.0	100.0	

Gaussian Random Number Generator					
		Frequency	Percent	Valid Percent	Cumulative Percent
Valid	-121	1	1.0	1.0	1.0
	-115	1	1.0	1.0	2.0
	-106	1	1.0	1.0	3.0
	-104	1	1.0	1.0	4.0
	-102	1	1.0	1.0	5.0

-100	1	1.0	1.0	6.0
-95	1	1.0	1.0	7.0
-89	2	2.0	2.0	9.0
-87	1	1.0	1.0	10.0
-82	1	1.0	1.0	11.0
-79	1	1.0	1.0	12.0
-77	2	2.0	2.0	14.0
-76	1	1.0	1.0	15.0
-73	1	1.0	1.0	16.0
-69	1	1.0	1.0	17.0
-68	1	1.0	1.0	18.0
-67	1	1.0	1.0	19.0
-66	1	1.0	1.0	20.0
-64	1	1.0	1.0	21.0
-57	1	1.0	1.0	22.0
-52	1	1.0	1.0	23.0
-51	2	2.0	2.0	25.0
-49	1	1.0	1.0	26.0
-47	1	1.0	1.0	27.0
-44	1	1.0	1.0	28.0
-43	1	1.0	1.0	29.0
-42	1	1.0	1.0	30.0

-41	1	1.0	1.0	31.0
-40	1	1.0	1.0	32.0
-38	1	1.0	1.0	33.0
-36	1	1.0	1.0	34.0
-33	1	1.0	1.0	35.0
-29	1	1.0	1.0	36.0
-27	2	2.0	2.0	38.0
-25	1	1.0	1.0	39.0
-19	1	1.0	1.0	40.0
-18	1	1.0	1.0	41.0
-15	1	1.0	1.0	42.0
-12	1	1.0	1.0	43.0
-11	1	1.0	1.0	44.0
-9	1	1.0	1.0	45.0
-8	1	1.0	1.0	46.0
-3	2	2.0	2.0	48.0
2	1	1.0	1.0	49.0
5	1	1.0	1.0	50.0
8	1	1.0	1.0	51.0
11	1	1.0	1.0	52.0
12	1	1.0	1.0	53.0
14	1	1.0	1.0	54.0

15	1	1.0	1.0	55.0
18	1	1.0	1.0	56.0
19	2	2.0	2.0	58.0
20	1	1.0	1.0	59.0
21	2	2.0	2.0	61.0
23	1	1.0	1.0	62.0
25	1	1.0	1.0	63.0
29	1	1.0	1.0	64.0
30	4	4.0	4.0	68.0
33	1	1.0	1.0	69.0
34	1	1.0	1.0	70.0
42	1	1.0	1.0	71.0
43	1	1.0	1.0	72.0
49	1	1.0	1.0	73.0
53	1	1.0	1.0	74.0
54	1	1.0	1.0	75.0
58	1	1.0	1.0	76.0
63	1	1.0	1.0	77.0
70	1	1.0	1.0	78.0
74	1	1.0	1.0	79.0
76	1	1.0	1.0	80.0
85	1	1.0	1.0	81.0

86	1	1.0	1.0	82.0
89	1	1.0	1.0	83.0
90	1	1.0	1.0	84.0
91	1	1.0	1.0	85.0
92	1	1.0	1.0	86.0
94	1	1.0	1.0	87.0
99	1	1.0	1.0	88.0
101	1	1.0	1.0	89.0
102	1	1.0	1.0	90.0
104	1	1.0	1.0	91.0
107	1	1.0	1.0	92.0
112	1	1.0	1.0	93.0
114	1	1.0	1.0	94.0
123	2	2.0	2.0	96.0
124	1	1.0	1.0	97.0
125	1	1.0	1.0	98.0
126	2	2.0	2.0	100.0
Total	100	100.0	100.0	

Fibonacci Random Number Generator					
		Frequency	Percent	Valid Percent	Cumulative Percent
Valid	-2092787285	1	1.0	1.0	1.0
	-2079590721	1	1.0	1.0	2.0
	-2015728079	1	1.0	1.0	3.0
	-1958435240	1	1.0	1.0	4.0
	-1869596475	1	1.0	1.0	5.0
	-1781832971	1	1.0	1.0	6.0
	-1709589543	1	1.0	1.0	7.0
	-1691007710	1	1.0	1.0	8.0
	-1581614984	1	1.0	1.0	9.0
	-1507123775	1	1.0	1.0	10.0
	-1418756969	1	1.0	1.0	11.0
	-1408458269	1	1.0	1.0	12.0
	-1323752223	1	1.0	1.0	13.0
	-1289228135	1	1.0	1.0	14.0
	-1262539787	1	1.0	1.0	15.0
	-1230842041	1	1.0	1.0	16.0
	-1109825406	1	1.0	1.0	17.0

-1070442683	1	1.0	1.0	18.0
-1055680967	1	1.0	1.0	19.0
-1036647147	1	1.0	1.0	20.0
-980107325	1	1.0	1.0	21.0
-945834654	1	1.0	1.0	22.0
-944741150	1	1.0	1.0	23.0
-889489150	1	1.0	1.0	24.0
-811192543	1	1.0	1.0	25.0
-798870975	1	1.0	1.0	26.0
-660827267	1	1.0	1.0	27.0
-433386095	1	1.0	1.0	28.0
-401779575	1	1.0	1.0	29.0
-298632863	1	1.0	1.0	30.0
-285007387	1	1.0	1.0	31.0
-188547518	1	1.0	1.0	32.0
-90618175	1	1.0	1.0	33.0
1	1	1.0	1.0	34.0
2	1	1.0	1.0	35.0
3	1	1.0	1.0	36.0
5	1	1.0	1.0	37.0
8	1	1.0	1.0	38.0
13	1	1.0	1.0	39.0

21	1	1.0	1.0	40.0
34	1	1.0	1.0	41.0
55	1	1.0	1.0	42.0
89	1	1.0	1.0	43.0
144	1	1.0	1.0	44.0
233	1	1.0	1.0	45.0
377	1	1.0	1.0	46.0
610	1	1.0	1.0	47.0
987	1	1.0	1.0	48.0
1597	1	1.0	1.0	49.0
2584	1	1.0	1.0	50.0
4181	1	1.0	1.0	51.0
6765	1	1.0	1.0	52.0
10946	1	1.0	1.0	53.0
17711	1	1.0	1.0	54.0
28657	1	1.0	1.0	55.0
46368	1	1.0	1.0	56.0
75025	1	1.0	1.0	57.0
121393	1	1.0	1.0	58.0
196418	1	1.0	1.0	59.0
317811	1	1.0	1.0	60.0
514229	1	1.0	1.0	61.0

832040	1	1.0	1.0	62.0
1346269	1	1.0	1.0	63.0
2178309	1	1.0	1.0	64.0
3524578	1	1.0	1.0	65.0
5702887	1	1.0	1.0	66.0
9227465	1	1.0	1.0	67.0
14930352	1	1.0	1.0	68.0
24157817	1	1.0	1.0	69.0
39088169	1	1.0	1.0	70.0
63245986	1	1.0	1.0	71.0
102334155	1	1.0	1.0	72.0
165580141	1	1.0	1.0	73.0
267914296	1	1.0	1.0	74.0
363076002	1	1.0	1.0	75.0
368225352	1	1.0	1.0	76.0
375819880	1	1.0	1.0	77.0
433494437	1	1.0	1.0	78.0
511172301	1	1.0	1.0	79.0
512559680	1	1.0	1.0	80.0
572466946	1	1.0	1.0	81.0
695895453	1	1.0	1.0	82.0
696897233	1	1.0	1.0	83.0

701408733	1	1.0	1.0	84.0
708252800	1	1.0	1.0	85.0
764848393	1	1.0	1.0	86.0
885444751	1	1.0	1.0	87.0
887448560	1	1.0	1.0	88.0
1073992269	1	1.0	1.0	89.0
1134903170	1	1.0	1.0	90.0
1412467027	1	1.0	1.0	91.0
1582341984	1	1.0	1.0	92.0
1640636603	1	1.0	1.0	93.0
1642909629	1	1.0	1.0	94.0
1776683621	1	1.0	1.0	95.0
1820529360	1	1.0	1.0	96.0
1836311903	1	1.0	1.0	97.0
1845853122	1	1.0	1.0	98.0
2118290601	1	1.0	1.0	99.0
2144908973	1	1.0	1.0	100.0
Total	100	100.0	100.0	

Appendix IV

Random Numbers generated by Various Generators

- a. Specific Range Random Number Generator: 782.0, 134.0, 78.0, 849.0, 702.0, 217.0, 163.0, 456.0, 128.0, 19.0, 220.0, 80.0, 144.0, 259.0, 766.0, 589.0, 877.0, 512.0, 762.0, 262.0, 739.0, 719.0, 852.0, 461.0, 663.0, 282.0, 22.0, 917.0, 714.0, 655.0, 455.0, 116.0, 248.0, 41.0, 739.0, 656.0, 984.0, 111.0, 895.0, 2.0, 988.0, 705.0, 834.0, 751.0, 587.0, 86.0, 9.0
- b. Secured Random Number Generator: 461.0, 958.0, 396.0, 844.0, 945.0, 933.0, 436.0, 359.0, 555.0, 923.0, 236.0, 503.0, 610.0, 159.0, 488.0, 597.0, 766.0, 919.0, 251.0, 136.0, 552.0, 717.0, 244.0, 632.0, 459.0, 168.0, 86.0, 136.0, 676.0, 866.0, 334.0, 686.0, 397.0, 849.0, 101.0, 864.0, 76.0, 469.0, 45.0, 76.0, 611.0, 933.0, 897.0, 123.0, 881.0, 670.0, 893.0, 900.0, 46.0, 512.0, 860.0, 825.0, 123.0, 607.0, 672.0, 80.0, 784.0, 203.0, 859.0, 648.0, 895.0, 524.0, 205.0, 638.0, 397.0, 751.0, 479.0, 171.0, 609.0, 613.0, 299.0, 715.0, 405.0, 230.0, 663.0, 813.0, 810.0, 609.0, 206.0, 362.0, 952.0, 766.0, 222.0, 700.0, 194.0, 675.0, 665.0, 500.0, 72.0, 447.0, 427.0, 552.0, 294.0, 958.0, 178.0, 55.0, 218.0, 34.0, 444.0, 450.0
- c. Gaussian Random Numbers Generator: 30.0, -95.0, 125.0, 5.0, -19.0, 124.0, -43.0, 76.0, 101.0, 70.0, 42.0, 107.0, -33.0, 2.0, 94.0, 30.0, -27.0, -73.0, -51.0, -87.0, 99.0, -69.0, 29.0, 126.0, -82.0, 25.0, -29.0, -52.0, -51.0, 19.0, 20.0, -42.0, -67.0, 30.0, 112.0, 86.0, -41.0, -115.0, -3.0, 11.0, 33.0, -79.0, -49.0, -77.0, -64.0, -104.0, 53.0, 92.0, -18.0, -102.0, 14.0, 114.0, 74.0, -68.0, 15.0, 126.0, -8.0, 49.0, 18.0, -121.0, -3.0, 34.0, -76.0, 123.0, -89.0, 43.0, -106.0, 54.0, 102.0, 19.0, -9.0, 91.0, 104.0, -77.0, -38.0, 21.0, 30.0, -44.0, -47.0, -57.0, 123.0, -11.0, 23.0, -25.0, -27.0, -36.0, -100.0, -40.0, 58.0, 89.0, 90.0, 85.0, 21.0, -89.0, -12.0, -66.0, 12.0, -15.0, 8.0, 63.0.

d. Fibonacci Random Numbers Generators:

1.0,2.0,3.0,5.0,8.0,13.0,21.0,34.0,55.0,89.0,144.0,233.0,377.0,610.0,987.0,1597.0,2584.0,4181.0,6765.0,10946.0,17711.0,28657.0,46368.0,75025.0,121393.0,196418.0,317811.0,514229.0,832040.0,1346269.0,2178309.0,3524578.0,5702887.0,9227465.0,1.4930352E7,2.4157817E7,3.9088169E7,6.3245986E7,1.02334155E8,1.65580141E8,2.67914296E8,4.33494437E8,7.01408733E8,1.13490317E9,1.836311903E9,-1.323752223E9,5.1255968E8,-8.11192543E8,2.98632863E8,-1.109825406E9,1.408458269E9,1.776683621E9,3.68225352E8,2.144908973E9,-1.781832971E9,3.63076002E8,-1.418756969E9,-1.055680967E9,1.82052936E9,7.64848393E8,-1.709589543E9,-9.4474115E8,1.640636603E9,6.95895453E8,-1.95843524E9,-1.262539787E9,1.073992269E9,-1.88547518E8,8.85444751E8,6.96897233E8,1.582341984E9,-2.015728079E9,4.33386095E8,1.845853122E9,1.412467027E9,-1.036647147E9,3.7581988E8,-6.60827267E8,-2.85007387E8,-9.45834654E8,-1.230842041E9,2.118290601E9,8.8744856E8,-1.289228135E9,-4.01779575E8,-1.69100771E9,-2.092787285E9,5.11172301E8,-1.581614984E9,-1.070442683E9,1.642909629E9,5.72466946E8,-2.079590721E9,-1.507123775E9,7.082528E8-7.98870975E8,-9.0618175E7,-8.8948915E8,-9.80107325E8,-1.869596475E9