AVOIDING TRACEABLE BIT STREAM PATTERNS IN LOGIC BASED STREAM CIPHERS USING INTELLIGENT ALGORITHMS

Researcher:

Syed Irfan Ullah

Reg. No.: 03-FAS/PHDCS/F03

Supervisor:

Prof. Dr. M. Sikandar Hayat Khiyal

Department of Computer Science
Faculty of Basic & Applied Sciences
INTERNATIONAL ISLAMIC UNIVERSITY, ISLAMABAD
PAKISTAN

1. Artificial intelligence
2. Pattern recognition systems

AVOIDING TRACEABLE BIT STREAM PATTERNS IN LOGIC BASED STREAM CIPHERS USING INTELLIGENT ALGORITHMS

Syed Irfan Ullah

Reg. No. 03-FAS/PHDCS/F03

Submitted in partial fulfillment of the requirements for the degree of Doctor of Philosophy in Computer Science at the Faculty of Basic and Applied Sciences International Islamic University, Islamabad.

Prof. Dr. M. Sikandar Hayat Khiyal   October, 2015

بسم الله الرحمن الرحيم

To My Family and Teachers,

Title of Thesis: ***Avoiding Traceable Bit Stream Patterns in Logic Based Stream Ciphers using Intelligent Algorithms***

Name of Student: ***Syed Irfan Ullah***

Registration No.: ***03-FAS/PHDCS/F03***

Accepted by the Department of Computer Science, Faculty of Basic and Applied Sciences, International Islamic University, Islamabad, in partial fulfillment of the requirements for the degree of Doctor of Philosophy in Computer Science

Viva Voce Committee

Dean FBAS (Prof. Dr. Muhammad Sher)--

14-9-15

Chairman DCS (Dr. Husnain Naqvi)--

External Examiner-I (Dr. Mujahid Alam)--

External Examiner-II (Prof. Dr. Sajjad Mohsin)--

Prof. Dr. M. Sikandar Hayat Khiyal (Supervisor) --
Faculty of Computer Sciences,
Preston University, H-8/1, Islamabad

Abu Hamid Muhammad al-Ghazali

Never have I dealt with anything more difficult than my own soul, which sometimes helps me and sometimes opposes me.

Abū Ḥāmid Muhammad ibn Muhammad al-Ghazālī (c. 1058–1111); known as Al-Ghazali or Algazel to the Western medieval world, was a Muslim theologian, jurist, philosopher, and mystic of Persian descent[1]

[1] ^ "Ghazali, al-". *The Columbia Encyclopedia*. Retrieved 17 December 2012.

Abstract

Information security is one of the challenging topics in the modern communication. The stake holders always demands fast and secure communication to share information and communicate over the pubic channels all over the world. They have different requirements and challenges in which they need to communicate. Stream cipher algorithms are best recommended for fast communication in noisy and bursty channels with limited memory and processing powers. These consume least memory, the encryption and decryption is based upon simple XORing the secret key with the given text and the bit wise XORing restrict the error to propagate and the operation is done with extremely high speed.

The entire security of stream ciphers is dependent on secret key, and a weak key may lead to compromise the entire security of a communication channel. The secret key is a random number that is generated by the computer using different Random Number Generator (RNG's) and Pseudorandom Number Generator (PRNG's) algorithms. These algorithms generate random numbers which are used by the stream ciphers for the encryption and decryption process.

The logic based stream ciphers algorithms are unable to generate true random numbers and the cryptanalyst takes the advantage of the weak points in their design and can trace the key stream with less efforts. The key stream may contain a series of patterns that may lead a cryptanalyst to trace the entire key successfully. The designer is always of the opinion to avoid weak patterns in the secret key to let the cryptanalyst unable to guess a part of it.

Stream cipher SNOW 2.0, that was considered to be unbreakable and secure but the later research proved that it has serious security loopholes. It is not due the lapses in the algorithm but the development in cryptanalytic techniques which were used for breaking SNOW 2.0 security. It is proved that the SNOW 2.0 Pseudorandom Number Generator (PRNG) generates weak patterns that let a cryptographer to break its security.

This study presents an intelligent stream cipher model of SNOW 2.0. This model intelligently generates a series of bit streams those can be used confidently as a secret key. The loopholes in the SNOW 2.0 Pseudorandom Number Generator (PRNG) that let a cryptanalyst to break the security of a stream ciphers by guessing a weak key, have been removed by passing it through

Artificial Neural Network (ANN) based system to regenerate a secure key. This model intelligently removes the statistical irregularities from the key stream by replacing the weak patterns with other statistically proved patterns. The main objective of this work is to close all the backdoors those may let a cryptanalyst to break the security of a stream cipher based cryptographic system. Results have proved that proposed model is more secure than SNOW 2.0.

Acknowledgements

I dually express my deep gratitude to my supervisor Dr. M. Sikandar Hayat Khiyal who helped me in the completion of my research work. I do thank him for his continuous encouragement and support for the transfer of knowledge and experience. As an advisor, he taught me practices and skills that I will use in my academic career.

I would also like to thank all my teachers, colleagues, and friends at the department of computer science, International Islamic University, Islamabad for being helpful over all these years of my studies. In particular I am grateful to Dr. Muhammad Sher, Dean Faculty of Basic and Applied Sciences, for his positive comments, valuable discussions and excellent cooperation.

Particular acknowledgements go to Mr. Fareed Ahmad Raza Zakori, Dr. Abd-us-Salam, Dr. Muazzam Ali Khan, Ms. Samina Zohra and Mr. Obaid Ullah for their all-time assistance at critical stages for improving this work.

Let me finally acknowledge the duas, support, encouragement, push and who constantly gave me enthusiasm from my family specially my mother, wife, and children. I hope I will make them proud of my achievements, as I am proud of them.

(Syed Irfan Ullah)

Table of Contents

List of Figures

List of Tables

List of Equations

List of Algorithms

List of Abbreviations

AI	Artificial Intelligence
ANN	Artificial Neural Network
ASCII	American Standard Code for Information Interchange
CA	Cellular Automata
CBC	Cipher Block Chaining
CCA	Chosen Ciphertext Attack
CFB	Cipher Feedback
CIM	Computational Intelligence Model
CPA	Chosen Plaintext Attack
CTR	Counter
CV	Coefficient of Variance
ECB	Electronic Codebook
FSM	Finite State Machine
GA	Genetic Algorithm
GD-Attack	Guess-and-Determine Attack
GRNN	General Regression Neural Networks
GSM	Global System for Mobile Communications
ICM	Intelligent Cryptographic Model
ISO	International Standard Organization
IV	Initial Vector
KB	Knowledge Base
KPA	Known Plaintext Attack
LFSR	Linear Feedback Shift Register

MLP	Multi-Layer Perceptron
NESSIE	New European Schemes for Signatures Integrity and Encryption
NIST	National Institute of Standards and Technology
NLFSR	Non Linear Feedback Shift Register
OFB	Output Feedback
OSI	Open System Interconnect
PCBC	Propagating Cipher Block Chaining
PRNG	Pseudorandom Number Generator
RAM	Random Access Memory
RM	Random Mapping
RNG	Random Number Generator
RSA	Rivest-Shamir-Adelman Algorithm
SN	Semantic Network
TCP/IP	Transmission Control Protocol/ Internet Protocol
WTC	Worst-case Time Complexity
XOR	Exclusive OR

CHAPTER 1
INTRODUCTION

The term Cryptography or cryptology is derived from the Greek word κρυπτός (*kryptós)* means secret or hidden. The word γράφω *(gráfo) means* coding or writing, which is now becoming the branch of information theory in the modern world and is used for secret messaging [1]. Cryptography is the computational and mathematical study of data to transfer information from one location to another location. During this transmission three important things are involved, these are:

i. **Encrypting** the information i.e. to make plaintext unreadable and un-understandable by the computer as well as human beings. This text is known as ciphertext.
ii. **Decrypting** the information i.e. to make ciphertext readable and understandable by the computer and human beings. This text is called plaintext.
iii. The message can sometime be recovered without key by the cryptanalyst or hacker using cryptanalysis. Though the modern cryptographic techniques are considered to be unbreakable.

Data protection is very difficult without proper cryptographic applications and is practically impossible, because opponents are always trying to intercept the communication. The threats (possible attacks) during electronic communication can compromise security. A lot of cryptographic techniques have been proposed to achieve an unbreakable security. These techniques are categorized as:

i. Block ciphers and
ii. Stream ciphers

These methods are considered to be trustworthy, secure and verifiable and are accepted as standard techniques. Over the last few years a series of researches [2] [3] [4] [5] have concluded that the stream ciphers less secure than block ciphers. The loopholes in the design of stream ciphers Pseudorandom Number Generators (PRNGs) and Encryption Algorithms their security has been compromised.

Stream ciphers are comparatively less secure than block ciphers **[2]** **[6]** [7]. The stream ciphers security is based upon the key generated by the Pseudorandom Number Generators (PRNGs). The encryption process is a simple XOR operation that mixes the plaintext with the key. The block cipher has a strong key and complex encryption process that produce confusion in the target ciphertext. Block ciphers consume more memory and involve complex encryptions which are not suitable for data communication in noisy data transfer. Block ciphers are comparatively more secure than the stream ciphers but they are not recommended in noisy and erroneous data transfer [**2**] [**6**] [7].

A stream cipher called SNOW 1.0 and its later versions have been proven to be compromised [**8**] [**9**] [**10**], although these have a strong cryptographic and mathematical background. The reason is that on one side system security is improved whereas on the other side code breaking knowledge has also improved. A good cryptographer not only cares for the secure algorithm but also takes care of current cryptanalytic attacks available and its effect on the design of a secure system. Based on key management cryptography is divided into two major categories:

i. Symmetric key or Single key Cryptography
ii. Asymmetric key or Multi key Cryptography

1.1 Symmetric Key Cryptography

Private Key encryption also called a single key encryption is one of the most familiar cryptographic techniques. In this technique the sender and receiver both use the same key. The sender encrypts the plaintext and the ciphertext is received by the receiver. The receiver uses a decryption algorithm using the same private key to decipher the ciphertext generated by the sender [**11**].

The plaintext is given to the encryption algorithm which converts it into ciphertext using key *K*. The ciphertext is in unreadable and understandable format. The resultant ciphertext is reconverted to plaintext by inputting it into decryption algorithm using the same key K as shown in **Figure 1.1** [12].

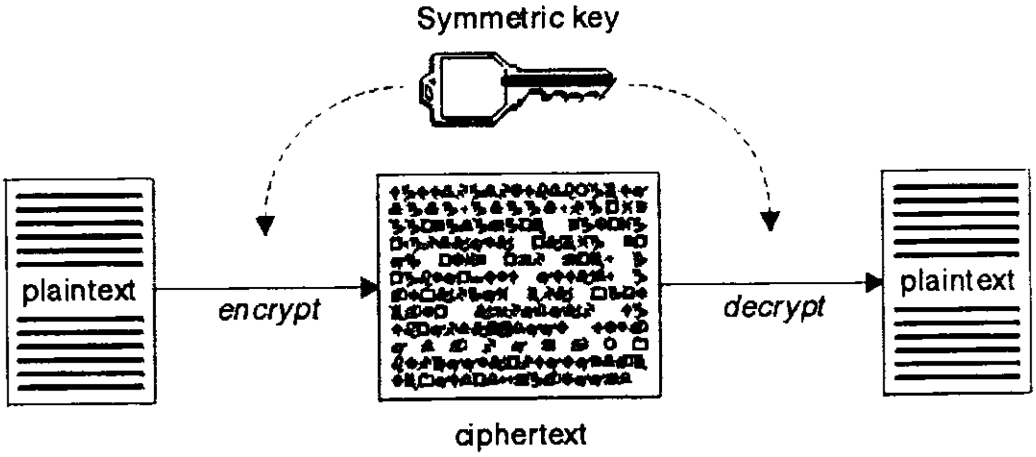


Figure 1.1: Symmetric Key Encryption and Decryption Process

Depending on the requirements different stream cipher algorithms are used in various cryptographic applications. The symmetric key cryptographic applications are used for encrypting a bulk amount of data in a private network. These algorithms can be trusted to secure secret information for a long period of time. The cryptanalyst cannot recover the plaintext from the ciphertext in polynomial time without having a key. The cryptanalyst gets the information about the encryption algorithm that has been used for encrypting the plaintext. The weak keys produced by Pseudorandom Number Generators (PRNGs) may let a cryptanalyst to recover all or a part of the plaintext by guessing the similarity index in the cipher text. This issue arises if a key with weak statistical properties is used as a secret key.

1.2 Asymmetric Key Cryptography

In asymmetric key cryptography separate keys are used for encryption and decryption. Initially pair of keys i.e. public and private key is generated. Then encryption takes place with public key and decryption is made with private key and vice versa. The public key cannot be used to decrypt to get back the plaintext. In this process the private key is kept secret, whereas the public key is shared with the partners. Asymmetric key cryptography is used both for confidentiality and authentication depending on the process of usage of the technique to use it for any of the required purpose **[11]** [13]. Asymmetric key cryptography is used to share secret information over a public channel in which the sender encrypt the plaintext with his private key and the receiver recovers it using the senders public key to verify the authenticity. Depending on the requirements public and private keys can be used in reverse if confidentiality is required.

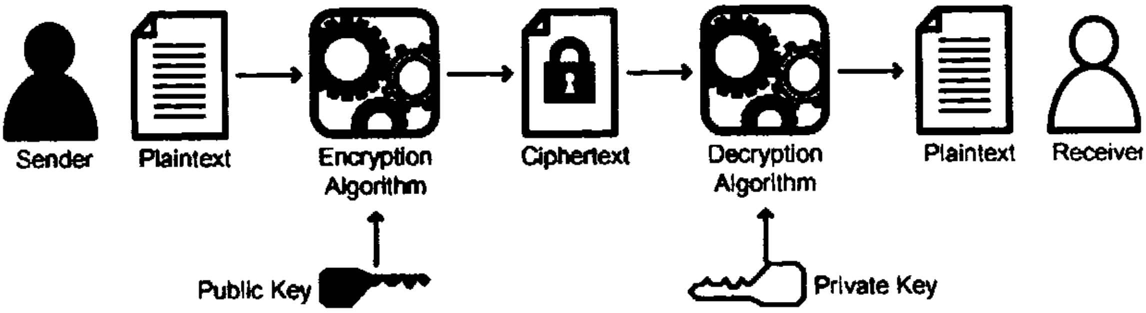


Figure 1.2: Asymmetric Key Encryption and Decryption Process

The plaintext is given to the encryption algorithm which converts it to ciphertext using the public key. The receiver recover the plaintext back by passing the ciphertext through the decryption algorithm using private key as shown in **Figure 1.2** [14]. Various genetic algorithms are used for secure key exchange over public channel. Cryptography is further divided into block ciphers and stream ciphers.

1.3 Block Ciphers

In block cipher cryptography the plaintext is converted into blocks of the same size, which simply may be a group of characters or bits. Considering the block of 8 bytes i.e. 64 characters, when the last block does not contain enough bytes to complete the block, then some padding bits are added to complete the block size [11] [15]. Block ciphers are considered to be more secure and are used by most of the security and defense agencies [16].

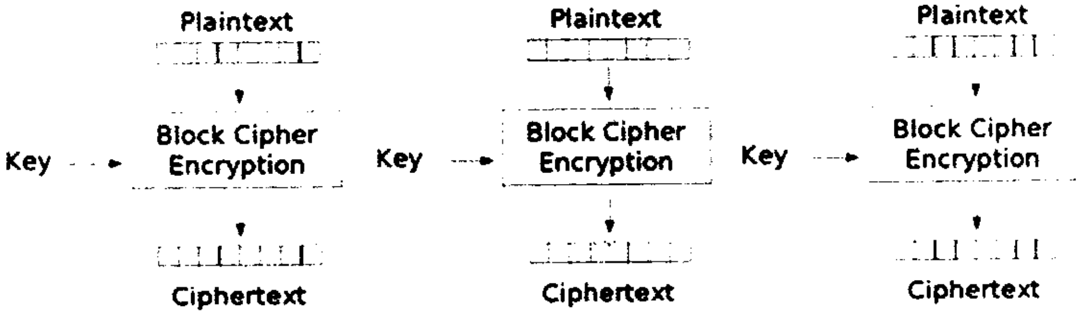


Figure 1.3: Block Cipher Algorithm (Electronic Codebook Mode of Operation)

The data is then passed to the encryption algorithm to convert this plaintext into ciphertext after converting into blocks. During the decryption process the ciphertext is converted into plaintext after dropping the padding bits (if any). Block ciphers are not recommended in noisy and erroneous data transfer modes.

The entire plaintext is divided into predefined block size. The plaintext block is given to the encryption algorithm which converts it into ciphertext block. This process continues till the end of the document. The electronic codebook mode of operation of the block cipher is shown in **Figure 1.3** [17]. Different modes of operation are used to convert a plaintext block into ciphertext block namely [3] [11] [18]:

i. Electronic Codebook (ECB)
ii. Cipher Block Chaining (CBC)
iii. Propagating Cipher Block Chaining (PCBC)
iv. Cipher Feedback (CFB)
v. Output Feedback (OFB)
vi. Counter (CTR)

The major disadvantages of block ciphers is that it consumes more time in the encryption and decryption process as the plaintext is converted into ciphertext after passing through several rounds. The memory is consumed in bulk and is not recommended where memory and computational power are limited. Further, error is propagated in the entire document even if the error occurs in a single bit. The complex nature of the encryption process of block cipher generates ciphertext even if a weak key is used as a secret key.

1.4 Stream Cipher

In stream cipher cryptography the key encryption algorithm takes one bit or one machine word or byte at a time from the plaintext and encrypts it using secret key as delineated in **Figure 1.4**. In this type of algorithms the whole focus is on the key strength and the key stream is generated by Finite State Machine (FSM). If the key is stronger enough then the decipherment without key would not be possible, but if the key is weak, then the cryptanalyst may look for the loopholes in the ciphertext, and apply different algebraic attacks to get the plaintext back [4] [19] [20].

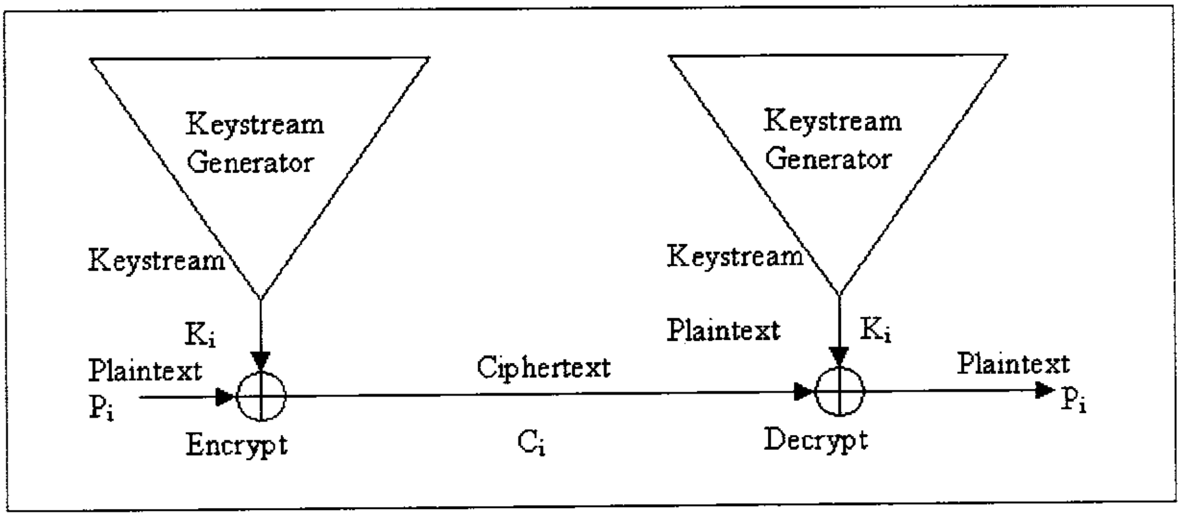


Figure 1.4: Stream Cipher Algorithm

The algorithms apply XOR operation on the key and plaintext bit, word or byte whatever applicable to generate the ciphertext and for decryption process the same key is again XORed with the ciphertext using the same algorithm, to get the plaintext back. It is very important that the key must be strong enough and the cryptanalyst must not get any of the information from the ciphertext, to decipher the code without key.

A single bit plaintext input P_i is given as input to the encryption algorithm which converts it into ciphertext C_i using K_i. The plaintext P_i can be recovered by inputting C_i to the decryption algorithm using the same part of the key K_i as shown in **Figure 1.4** [21]. Stream cipher algorithms have the following important characteristics:

i. Easy to implement and very fast as compared to block ciphers

ii. Minimal hardware and memory requirement

Due to the above mentioned features stream ciphers are best recommended in situations where a very high speed encryption is required, channels are noisy, memory is limited and hardware resources are limited. In military communication, stream ciphers are more popular than block ciphers. These algorithms take less time in encryption and decryption and works better in noisy and even in erroneous communication. These algorithms are best recommended where real time actions are critical. The entire security of these algorithms is dependent on key stream generator, because if the resultant key has weak cryptographic properties then the entire security can be compromised. The key size of most of the stream ciphers is 128 or 256 bits, which is randomly produced by key stream generators. If key stream has the repeated patterns then the encrypted text may let the cryptanalyst of deduce some part of the key, which may let him to recover the entire plaintext. Repeated patterns can be distorted in weak keys using Artificial Neural Networks (ANN) which makes the key more effective by producing a disbursed the ciphertext.

1.5 Artificial Neural Network

The idea of Artificial Neural Network (ANN) has evolved due to inspiration from the human understanding and knowledge processing [22] [23] [24] [25]. The human brain is considered to be using the massive parallel computations. Thousands of neurons are working and processing at the same time while performing different tasks. These are termed as computing elements in the massively parallel system and are envisioned to perform even a very simple task as shown in **Figure 1.5** [26].

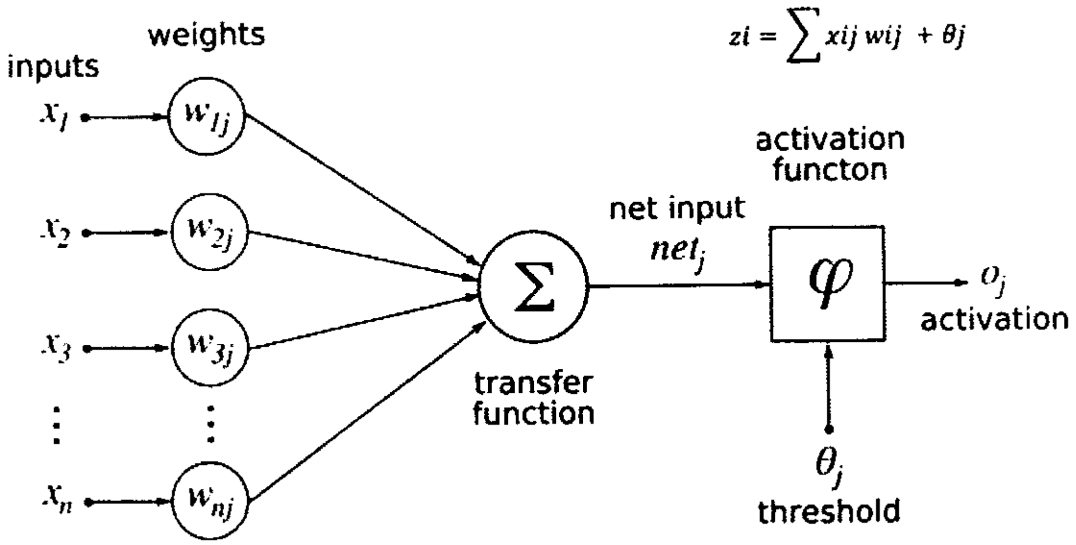


Figure 1.5: Mathematical Representation of Artificial Neural Network

$y_i = f(z_i)$ where z_i is assumed to be real valued input,

y_i is the binary or real values output of the i^{th} neuron, and f is a non-linear function or node function, depending on the application area. Its structure varies and may be a simple equation or a complex mathematical operation, but its general structure is same as shown in the **Figure 1.5**.

In human body a neuron takes input signals from different organs of the body for example eyes to look, ear to hear, body to touch, nose to smell etc. and the output of one neuron may be sent to another neuron or other part of the body for further processing.

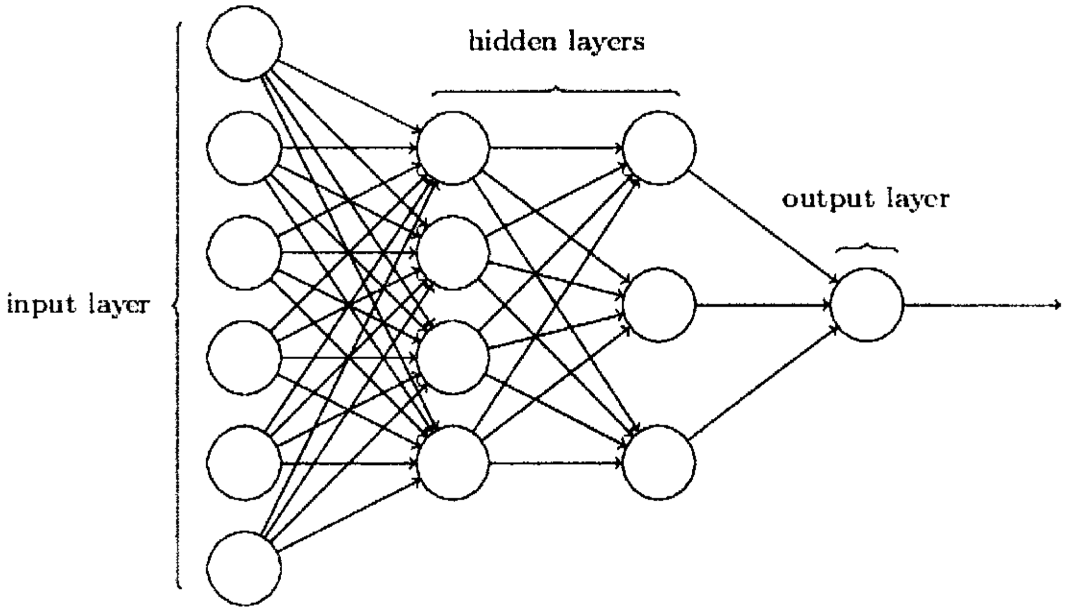


Figure 1.6: Multilayer Architecture of Artificial Neural Network

Artificial Neural Network has input layer, hidden layer and output layers. The input layers takes input from the outer side and pass it to hidden layer. The hidden layer processes the input value and concludes its decision and delivers its processed data to the output layer as shown in **Figure 1.6** [27]. Similarly, in ANN the input to one node function may be any fact required for that processing and the output is given to another node function or main function where the node functions or neurons work collectively as a processing unit. A trained Artificial Neural Networks (ANN) plays an effective role in producing a secret key. This work later on proves that it can be effectively used for a generating a secure key having strong statistical properties by removing the traceable bit stream patterns.

1.6 Stream Cipher Algorithms

Stream ciphers are very important class of ciphering scheme in cryptography based· on symmetric key, i.e. the same key is used for both encryption and decryption using relevant algorithms. A stream ciphering algorithm works in the following two steps:

i. A pseudorandom key is generated which is exclusive-OR (XORed) with the plaintext to get the ciphertext using the encryption algorithm.
ii. In reverse the decryption operation takes place in which the same key is XORed with the ciphertext back to generate the plaintext.

Following are the main features of stream ciphers:

i. They are very fast and recommended in areas with low memory and computational power.
ii. They are very effective in noisy situations, where there are chances of errors during data transfer, because every bit or byte is encrypted independent of the others, therefore error does not propagate during processing.

A pseudorandom key can be classified as strong and weak key depending on various statistical properties. Semantic networks effectively do this job by passing the key through various semantic nodes. Each node checks a specific statistical property and classifies further to pass it through other statistical test. The overall result of the semantic network is a key classified as strong or weak key.

1.6.1 Security Issues in Stream Ciphers

Stream ciphers are comparatively less secure because the encryption process is done by XORing the plaintext with the key to generate ciphertext. The security of the stream ciphers is mainly dependent on the key stream produced by the random number generator that is called key stream generator. The cryptanalyst while planting an attack may check the behavior of the key stream generator of the stream cipher and its approach to the keys those are produced by it.

The improper use of the stream cipher algorithm by the programmer, even if the algorithm is stronger enough may cause serious attacks and the security is being compromised. The cryptanalyst may check the generation algorithm to determine that whether the key produced has some of the loopholes and may implant a series of attacks to determine the key from the ciphertext and recover they plaintext. It is recommended that the key must be used only once and for the next time a new key must be generated and utilized **[5]** **[28]** **[29]** **[30]** **[31]**.

The focus of this work is to find the solutions of the improper work of the stream cipher algorithms. Stream ciphers are inherently weaker **[18]** because of simple encryption and decryption operation. The simplest encryption operation improves the efficiency and the encryption process is performed with a tremendous speed. A weak key may key may let a cryptanalyst to deduce a part or the entire key by using chosen-plaintext attack. A strong secret key is required to encrypt a plaintext so that the cryptanalyst may not be able to recover a plaintext. The stream ciphers do not give better attention to ciphering algorithm, because they mostly XORing the key stream with plaintext. Stream ciphering algorithms respond differently to cryptographic attacks as listed below:

i. All cryptographic attacks (like algebraic, differential attacks) applicable on stream ciphers.
ii. The correlation attacks are applicable on stream ciphers. Algebraic attacks on stream ciphers are more effective.
iii. Minimal chances of data loss in stream ciphers because in erroneous communication only the affected part of the text is lost.
iv. Guess-and-Determine attacks, slide attacks and cube are more effective over the stream ciphers.

A large number of applications **[32]** **[33]** based on stream ciphers are shifting to block ciphers, because it is claimed that stream ciphers do not give ultimate security. Its use in telecommunication and military sector unavoidable because it efficiently work in a noisy and bursty communication. This data is required to be kept secret and no one should be able to break the security of these systems in polynomial time and can only be made readable if someone has the relevant key and encryption algorithm **[28]** **[29]** **[30]**. A weak key having poor statistical properties let a cryptanalyst to guess a part of the key and determine the rest of the key. If a part of the key used for the encryption process is as same as some other part of the key, then the ciphertext will have that reflection. The simple XORing of the encryption process with both the similar patterns will result the same ciphertext.

A cryptographic algorithm is considered that it is known to anyone but they are unable to break its security and only the authorized person having the key would be able to get proper information from the encrypted text. The designer must keep in mind that the key used for encryption must be stronger enough and must not be breakable in polynomial time. The key must not have the patterns those may give a chance to the hacker to get information from ciphertext. The Guess-and-Determine attacks are based on guessing a part of the key and determine the rest of the key. They are more effective against stream ciphers and the key may be compromised to leak out the secret information.

1.6.2 Properties of Stream Ciphers

For the last few decades a lot of work is done in the cryptography to improve the security of stream ciphers [2] [18] [34] because of its advantages over the block ciphers. Every stream ciphering algorithm has the following properties:

i. Stream ciphers are more efficient than block ciphers
ii. Used in area of minimal memory and other resources
iii. Used in noisy and erroneous data transfer
iv. Real time actions and data transfer like in military signaling

Stream cipher algorithms developed have been theoretically and mathematically proven to be stronger, but when they come into real world applications a strange approach from hackers may compromise their security and can breakdown its security in practical [35] [36] [37]. The cryptanalyst needs to know about advanced cryptanalytic techniques those are in practice among the hackers. Special attention is required to design a new cryptosystem and resist all types of possible attacks.

In stream ciphers the main focus of the cryptosystem is key i.e. password therefore it is required to generate such a pseudorandom key of a given size that is proven to be unbreakable in polynomial time by a cryptanalyst. A weak key let a cryptanalyst to break the entire security of a cryptosystem and may lead to dangerous situations. The differential key analysis [3] [5] [38], dictionary attacks [39], Guess-and-Determine attacks [40] and many other known attacks [31] [39] are there to compromise its security. The focus of this research work is on getting such a key stream which is unbreakable by these cryptanalytic techniques used by the cryptographers.

1.7 Random Number and Sequencing

A weak key may let a cryptanalyst to compromise the entire security of a cryptosystem; therefore it is required to produce a secure random sequence of bits. A randomly generated key stream produces more confusion in the ciphertext and it becomes very difficult for a cryptanalyst to guess a part of a key. A key having weak statistical properties is compromised and the plaintext can be recovered over a guessed key. If a part of a key is compromised it may guide an analyst to guess another part of a key. Random numbers can be generated through the following two major types of random number generators [41] [42]:

i. Random Number Generators (RNGs)
ii. Pseudorandom Number Generators (PRNGs)

These generators produce stream of 0's and 1's depending on the size of the key stream which is subdivided into sub streams or blocks of random numbers. The sequence of bits is generated randomly and behaves like fair coin with two sides head or tail where the probability of occurrence of head or tail is same and there is no biasness in their occurrences [41] [42]. The probability of 0's and 1's is exactly ½ and produces a sequence of (0, 1) without any planning. If there is a perfect random bit sequence generator then the sequence will have 0's and 1's which are uniformly distributed. This sequence of un-biasness and fairness has an important role in the field of cryptography. The key produced by this process will not be predictable and the cryptanalyst cannot properly get the knowledge whether the key stream generator is malfunctioning. Thus the key stream produced will not have the uniform distribution of 0's and 1's in the given sequence.

All elements of the sequence are generated independent of each other, and the value of the next element in the sequence cannot be predicted, regardless of how many elements have already been produced. Obviously, the use of unbiased coins for cryptographic purposes is impractical. Nonetheless, the hypothetical output of such an idealized generator of a true random sequence serves as a benchmark for the evaluation of random and pseudorandom number generators.

1.7.1 Random Number Generators (RNGs)

The first type of sequence generator is Random Number Generator (RNG). They use nondeterministic source like noise in the electrical circuit, the processor time taken by some process like key strokes on keyboard or mouse movement etc. This information is given to the entropy distillation process to produces randomness. The entropy distillation is necessary to remove any weakness in the entropy source which may result in non- random numbers which are easy to predict like long sequence of 0's and 1's.

The output produced by RNGs must be unpredictable; however, there are some entropy sources which are quiet predictable such as date and time. The output of these random number generators is reprocessed by some other techniques to regenerate a random numbers to overcome this problem. Still such a random numbers have the deficiency and the cryptanalyst may predict it easily. Thus instead of using such a RNGs it is preferable to use PRNGs to generate large number of random numbers.

1.7.2 Pseudorandom Number Generators (PRNGs)

The pseudorandom number generators (PRNGs) use one or more inputs and produce multiple pseudorandom numbers. The inputs given to the PRNGs are called seeds, which by self are random numbers. The seeds given to the PRNGs must also be unpredictable and the resultant pseudorandom numbers have no correlation with the seed. The pseudorandom numbers is passed to deterministic algorithms which produces deterministic pseudorandom number. The reproduction of these random numbers, eliminate statistical autocorrelation between input and output and produce more randomness. The output of a PRNG may have better statistical properties than RNGs.

1.7.3 Forward Unpredictability

The cryptographic applications those produce the random and pseudorandom numbers should be unpredictable and should not be determined by any cryptanalytic method. If one key is produced by Pseudorandom Number Generator, the other key produced by it has no relation with the previous one. The new key must be independent of the previous keys produced by it. All bits generated by PRNGs are at random and have ½ probabilities. This property is known as forward unpredictability.

The seed used for generating the key must be unknown and could not be guessed from the key generated by the PRNGs i.e. the backward unpredictability is also required. If the key generation algorithm is weak then the key can be determined if the seed is known. As most of the stream ciphers are known then the designer must ensure that the seed must be random and secret otherwise the analyst may get the knowledge about the resultant key therefore, the seed must also be unpredictable.

1.8 Statistical Testing

A randomly generated sequence of bits is required to satisfy various statistical tests. It is not necessary that each randomly generated key is stronger enough and satisfy all parameters defined for verifying randomization properties. A randomly generated sequence of bits may have some series of bit patterns similar to some other part of the key. Moreover if millions of keys are produced there may be a chance that some keys may be repeatedly generated in a given key space. Various statistical tests are applied to ensure the true randomness in a key produced by PRNGs. These tests ensure the likeliness of a key that whether it is acceptable to use as a secret key or not? These tests check the randomness in different angles to ensure whether there are some patterns of bits available in the sequence which may result in guessing the entire sequence? If the sequence does not have weak patterns and lies in the acceptable region then it is accepted as key otherwise it is rejected.

Statistical hypothesis tests can be used as under:

Null Hypothesis, H_0 = the sequence is random

Alternate Hypothesis, H_a = the sequence is not random

This hypothesis is made for a series of statistical tests and the result of each test my lie in the acceptable or rejected region, which conclude that whether the sequence produced is an acceptable sequence of bits or not. The critical region is defined and results is verified whether it exists in it or otherwise. National Institute of Standards and Technology (NIST) [41] [43] [44] has introduced sixteen statistical tests which are necessary for certifying whether a key is truly random. These statistical tests ensure that a randomly generated bit sequence does not have the repeated patterns or a series of bit patterns that may lead to guess the forth coming bit sequences in a given key or a key space. NIST recommended tests for randomness verification are listed below:

i. The Frequency (Monobit) Test
ii. Frequency Test within a Block
iii. The Runs Test
iv. Test for the Longest-Run-of-Ones in a Block
v. The Binary Matrix Rank Test
vi. The Discrete Fourier Transform (Spectral) Test
vii. The Non-overlapping Template Matching Test
viii. The Overlapping Template Matching Test
ix. Maurer's "Universal Statistical" Test
x. The Lempel-Ziv Compression Test
xi. The Linear Complexity Test
xii. The Serial Test
xiii. The Approximate Entropy Test
xiv. The Cumulative Sums (Cusums) Test
xv. The Random Excursions Test
xvi. The Random Excursions Variant Test

1.9 Considerations for Randomness and Unpredictability Testing

The following assumptions are made with respect to random binary sequences to be tested:

i. **Uniformity:** The occurrence of a zero or one is equally likely at any point in the generation of a sequence of random or pseudorandom bits. The expected number of zeros (or ones) is *n/2*, where *n* is the sequence length.

ii. **Scalability:** Any test applicable to a sequence can also be applied to subsequences extracted randomly. If a sequence is random, then any such extracted subsequence should also be random. Hence, any extracted subsequence should pass any test for randomness.

iii. **Consistency:** The behavior of a generator must be consistent across starting values (seeds). It is inadequate to test a PRNG based on the output from a single seed, or an RNG on the basis of an output produced from a single physical output.

1.9.1 Random Mappings

When the data vectors are high-dimensional it is computationally infeasible to use data analysis or pattern recognition algorithms which repeatedly compute similarities or distances in the original data space [45]. It is therefore necessary to reduce the dimensionality before, for example, clustering the data. Random Mapping (RM) is a fast dimensionality reduction method categorized as feature extraction method. The RM consists in generation of a random matrix that is multiplied by each original vector and result in a reduced vector Let F_n denote the collection of all functions (mappings) from a finite domain of size n to a finite co-domain of size n. A random mapping model is that where the random element F_n are considered for processing. If there is a model containing a series of random functions and we pick each and every model equally likely, then randomization may occur and these models are frequently used in number theory and cryptography [18] [45] . Because the cryptography is based upon choosing a true random number if possible and passed through the algorithmic number theory to check the fitness and of its application.

If there is function $|Fn| = n^n$, then the probability of selecting that particular function is $1/n^n$.

To extract the valuable knowledge from the information stored in the knowledge base the functional graphs are produced and to encircle the areas which are associated with each other. A functional graph may have several *components.* Each component consists of a directed *cycle,* and some directed *trees* attached to the cycle as shown in **Figure 1.7** [45].

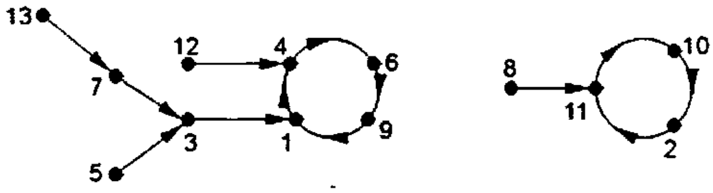


Figure 1.7: A Functional Graph Containing Cycles and Trees

These functional graphs are produced in a way that, if there are a series of keys produced and are kept together then those parts of the key are encircled which have the resemblance with other keys. For example if a key that is having repeated 1's or 0' then it is checked whether some other keys produced by the given key stream generator is also having such a series. This process is applied on the consecutive key streams, and the patterns are encircled so that we can easily find out that the key stream generator is producing the patterns those are also repeated in other keys. The cryptanalyst also determine that after how many keys such pattern is being repeated. The picture shows that if a set of keys is given then there is a probability that the sane sequence is repeated if it is not controlled intelligently. The designer may stuck in the circular part of the Finite State Machine (FSM) and would generate a series of similar patterns.

1.9.2 Complexity Theory

To solve the computational problems we need to classify them according to the resources needed, the complexity theory is used. Classification is based on the complexity and difficulty of the problem and does not rely on a particular computational model. The resources might be of different types for example the number of processors, time, random bits and storage space etc. **[18] [42] [46]**.

The computational procedure which takes inputs in the form of variables and results output is called algorithm. In sense of computational model, i.e. the set of instruction given to the computer in an order to accept input from some source and give resultant value called output. The computer can't be operated without these instructions and they are designed in a view to give fruitful outcomes.

To get a correct output the computational procedures or algorithms are designed by using the formal computational models for example the Boolean Circuits, Turing Machines, random access machines etc. The end user is not being involved in the technicalities of involved while designing these computational procedures, but the scientists must carefully observe and producing more computational models. Looking to the problem, an algorithm is taught and implemented in some programming language for computers and halts on a state that give the required output. Designing a new algorithm for solving a computational problem it is important to keep the following points in mind:

i. It should be efficient and depends on the time taken by the algorithms when it halts
ii. The resultant output must be accurate and
iii. Other algorithms working for solving that problem should be known and this new algorithm must be comparatively better than the previous ones.

For cryptographic problem the input size is the total number of bits, provided to the algorithm for solving a problem and sometimes it is the number of items in the input [**18**] [**43**].

i. The number of bits in the binary representation of a positive integer *n* is 1 + lg (n)bits. For simplicity, the size of *n* will be approximated by lg (n).
ii. If *f* is a polynomial of degree at most *k*, each coefficient being a non-negative integer at most *n*, then the size of *f* is (k + 1) lg (n) bits.
iii. If *A* is a matrix with *r* rows, *s* columns, and with non-negative integer entries each at most *n*, then the size of *A* is (r.s.lg(n)) bits.

The number of steps taken by the algorithm while solving a new problem is called the running time of the algorithm. A step may be a comparison, a machine instruction, a clock cycle or modular multiplication. A step can be any operation those are involved in generating the final output.

i. The total number of "steps" or operations done by algorithm on some input is called the running time complexity.
ii. The *average-case running time* complexity is the average time taken by the algorithm for a particular input, because some inputs may take more time and some may take comparatively less time so the average-case time complexity is required to check the overall performance of the algorithm.
iii. The *worst-case running time* of an algorithm is an upper bound on the running time for any input, expressed as a function of the input size, i.e. the maximum time taken by an algorithm from solving a particular problem.

1.9.3 Asymptotic Notations

A Pseudorandom Number Generator (PRNG) produces a random bit sequence that is a member of a given key space. A cryptographer prefers a key that is the member of a key space but must not lie at their edges. A cryptographer is restricted to design such a security algorithms that lies in a given asymptotic upper bound and lower bound. He tries to increase the time complexity for searching a key space and the cryptanalyst must not be able to find out the key in polynomial time. The exhaustive search attacks and other statistical attacks are infeasible in terms of time and resources that infinite time and resources fails to search a given key.

Due to the difficulty of finding out the exact running time of the algorithm; it is needed to use approximation of the running time. For this purpose asymptotic running time of the algorithm is derived, i.e. so that the complexity of the algorithms can be found easily irrespective of the size of the input is provided, they exist within the given bounds **[18]**.

1.9.4 Complexity Classes

The functions having worst-case runtime complexity is of the form $O(m^n)$, is called a polynomial time algorithm, where m is the input size and n is a constant. If m = 64 and n = 100, then $O(m^n)$ = $O(64^{100})$, lies in polynomial time. The runtime complexity for such an algorithm cannot be bounded and are called exponential time algorithms. The algorithms whose complexities lie in the exponential time are considered to be inefficient but there are certain realities and practical solutions where this distinction in not appropriate. For example, the degree of polynomial is significant while considering the polynomial time complexity, so the algorithm having the running time complexity of $O(n^{\ln(\ln(n))})$, where n is the input size, is slower than the algorithm with a running time of $O(n^{100})$ asymptotically. The first algorithm may be faster in practice for smaller values of n, specially the hidden constant used by the big-O notation are smaller.

The average time complexity in cryptography is more important than that of the worst-case time complexity. A necessary condition, that the cryptographic scheme used for encryption is considered to be secure and the cryptanalysis problem is difficult on average, and not just for some isolated cases. So, the designer of the algorithm must keep this in mind the he should not only find the worst case time complexity but also the average, and the best case time is complexity too.

An algorithm whose worst-case running time complexity function is of the form $e^{o(n)}$, where n is the input size, then such an algorithm is called *subexponential-time algorithm.*

An algorithm whose running time is fully exponential in the input size, the *subexponential-time algorithm* is faster than that, while the former is asymptotically slower than polynomial time algorithms.

The set of all decision problems those are solvable in polynomial time are kept in the complexity class **P.**

The set of all decisions problems for which a YES answer can be verified in polynomial time given some extra information, called certificate are kept in class **NP**.

The set of all decisions problems for which a NO answer can be verified in polynomial time using appropriate certificate are kept in complexity class **co-NP**.

All cryptographic algorithms are proven to be NP-Hard and their solution is not possible in polynomial time. The cryptanalyst applies transformation or detection strategies for finding the solution by tracing the cryptographic weaknesses in a given security model. If it is proven that the NP-Hard problem is converted into NP-Complete then transformation and detection techniques fail to find out the solution of a problem. This work removes the traceable bit patterns from the key generated by PRNG of SNOW 2.0 security model to vanish the chances of transformation and detection techniques to find out its solution.

1.10 Statistical Hypothesis Testing

A Pseudorandom Number Generator (PRNG) generates key, which may be random or nonrandom depending on their statistical properties. The ultimate goal of this work is to generate a random sequence of bits and a nonrandom sequence is classified as weak key. National Institute of Standards and Technology (NIST) recommends various statistical for this purpose that checks the key structure for various statistical properties. The sequence of bits and their orientation, long run of zeros or ones, pattern repetition, overlapping and non-overlapping properties of the patterns etc. is verified in a given key. The key is verified to be a true random and having higher degree and entropy and distortion.

Statistical hypothesis testing is a conclusion-generation procedure that has two possible outcomes [**41**]:

i. Null Hypothesis, H_0 = the key stream is random
ii. Alternate Hypothesis, H_a = the key stream in not random

The following table shows the situations that whether accept the hypothesis or reject it.

Table 1.1: Statistical Hypothesis Test

True Situation	**Conclusion**	
	Accept H_0	**Accept H_a (reject H_0)**
Data is Random (H_0 is True)	No error	Type I error
Data is not Random (Ha is not True)	Type II error	No error

Two types of errors may occur:

Type I error: reject null hypothesis and accept alternate hypothesis (i.e. the sequence is non-random) whereas the sequence in truth is random.

Type II error: reject alternate hypothesis and accept null hypothesis (i.e. the sequence is random) whereas the sequence in truth is non-random.

The probability of a Type I error is called level of significance of the test, which is set prior to the test and is represented as α, where α is the probability that the test results that the sequence is non-random when it really is random.

The probability of a Type II error is called level of significance of the test, which is set prior to the test and is represented as β, where β is the probability that the test results that the sequence is random when it really is non-random. The calculation of β is difficult than α because of the many types of non-randomness.

These tests are used to minimize the risk of Type II errors. The non-random sequence treatment as random may result in worst cryptographic loopholes and may result in information vulnerability.

1.11 Motivation

i. Humans have an interesting approach to problem solving, and a person cannot specifically says that by what reasons he responds intelligently. The knowledge base is an integral part of his intelligence but it is not the only part that enables a human to respond intelligently because there are the situations in which a person with no knowledge also has a reasonable response [47] [48].

ii. Human beings have the capability to understand and adopt the new environment, whereas the computer does not have this capability. Genetic programming and artificial intelligence algorithm can also let the computers think and to make decisions based on the situation arises [49] [50].

iii. Stream ciphers play an important role in securing data where the communication channels noisy and less computational power and has the following features:

   a. Stream ciphers are more efficient than block ciphers

   b. Used in area of minimal memory and other resources

   c. Used in noisy and erroneous data transfer

   d. Real time actions and data transfer like in military signaling

iv. In military communication stream ciphers are more popular than block ciphers because they take less time in the encryption and decryption and works better in noisy and even in erroneous communication [29] [51].

1.12 Thesis Organization

This dissertation is organized in the form of chapters, in the following each chapter is described briefly so as to make it more understandable:

- In ***Chapter 2***, we review the relevant literature concerning traceable pattern in the key streams of different ciphering algorithms and the work done in this area. We also discuss the artificial intelligence techniques those are suitable for ciphering algorithms; also we present some of the algorithms those are claiming to be intelligent.
- In ***Chapter 3***, we analyze the stream cipher SNOW 2.0 and apply different statistical attacks then present the formal specification of the problem of the traceable bit stream

patterns in logic based stream ciphers, to check the vulnerability of the stream ciphers, and present a formal discussion on the reasons and the realities that why we adopt this new approach.

- In ***Chapter 4***, we present a proposed model based on intelligent algorithms and stream cipher, and the algorithm that extract knowledge from the Knowledge Base while creating the new key and intelligent action in such a situation.
- In ***Chapter 5***, we present our results and the comparative analysis of both the original ciphering Algorithm and our proposed model.
- In ***Chapter 6***, we conclude our work and give recommendations for the future work.

1.13 Summary

A related knowledge is provided in this chapter to understand the rest of the work in this dissertation. Stream cipher algorithms are used for swift communication and are suitable for the noisy communication channels. In case of error only specific part of the data is disrupted, and the error does not propagate in the proceeded bits. The major problem with the stream ciphers is that these are comparatively less secure than block ciphers but required in high speed communication. In stream ciphers the security is based upon the key that is used for the encryption and decryption process. The designers always have concerns about getting a strong random key. If the key generated is weak the cryptanalyst may get a clue regarding the plaintext which may compromise the entire security of the system.

The cryptanalyst tries to break an encryption algorithm and guess a sequence of bits that may be used as a key and tries to find out the weak points in the design of key stream generator and keys produced by RNGs. NIST statistical suite is used for a randomness verification of key that is produced by a given PRNG and before using it as secret key it is required to pass randomness verification test.

CHAPTER 2
RELATED WORK

Classical data encryption algorithms were used for data security and the messages were encrypted using these algorithms. The evolutions of computer change the approach towards modern cryptography. Stream cipher algorithms have a short history and a wide range of contribution in telecommunication sector. This chapter reviews:

i. Modern stream ciphers
ii. Genetic programming based stream ciphers
iii. Cryptanalysis and security attacks on stream ciphers
iv. ANN based stream ciphers

Cryptography is one of the most concentrating areas that attract everyone to trace out the secret of another that ranges from an individual to the organizations and countries in their national interest. The work done in this field is always a trade secret and the organizations and departments keep their security models and algorithms secret. The cryptanalyst not only tries to break an encryption algorithm but also tries to trace the seasons of breaking a security algorithms. Every stream cipher cryptosystem consists of two parts which are:

i. Key stream (Random sequence bit) generator
ii. Mixer (XOR for the binary sequence)

The encryption process is simply XORing the key stream with the plaintext data. If the key stream produced is not strong enough then a cryptanalyst may implant different cryptanalytic attacks to recover the key or produce the plaintext from the given ciphertext which may lead to code breaking.

2.1 LFSR Based Stream Ciphers

A number of cryptographic stream ciphers are available such as SNOW 2.0 [9] [10], ORYX [52], RC4 [53], SEAL [54], Rabbit [55], Turing [56], A5/1 [32] [33], E0 [19] [57], COS [58], Scream [59], HC-256 [60] [61] [62] . The algorithms like Golden Fish [63], Fish Tailed Lion

[64] are claimed to be intelligent but their critical analysis show that these algorithms have been amalgamated with some other cryptographic techniques. These algorithms use different techniques to improve their security. This section covers the design and cryptographic loopholes in modern logic based stream ciphering algorithms.

2.1.1 Golden Fish Algorithm

Golden Fish algorithm claims to be an intelligent stream ciphering algorithm. It is designed by fusing the block cipher operation mode with the memory modules of the stream cipher, where the security is proven by periodic and nonlinear evaluation. The Linear Feedback Shift Register (LFSR) of stream cipher is replaced by the nonlinear characteristics of the block ciphers. The author claims that it is an intelligent algorithm, however, the real structure and working process shows that it is a logic based stream cipher and the LFSR characteristics are converted to non-linear by fusing it with the block cipher. The linear characteristics of the LFSR based stream cipher are fused with the block cipher to get a nonlinear system [63] .

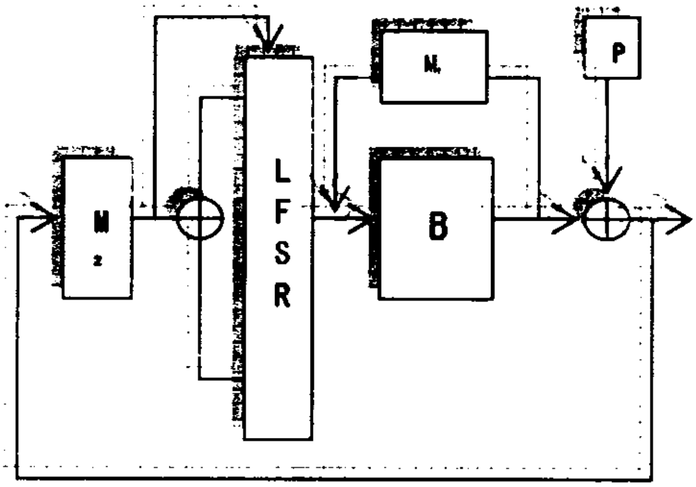


Figure 2.1: Golden Fish: Block Cipher and Stream Ciphers Fused Together

The M_2 output is given as input to LFSR which is adjusted as result of XORing M_2 and LFSR of the current state. The output from LFSR is given to B (block cipher) after mixing it with the output of B. The output from B is again XORed with P that generates a single bit stream at random. The random bit is kept in array and also given again as input to M_2. The process is repeated until the desired key of a given length is generated as shown in **Figure 2.1.**

2.1.2 Fish Tailed Lion Algorithm

It is a mini intelligent stream cipher that includes one stream cipher, block cipher and two HASHs. It is a toy-model design based on combining the ciphers, which are S-quark, Trivium, Skein and Serpent. In this model the resultant Fish Tailed Lion is claimed to be different from the classical ciphering algorithms. It is a simple module that enhances the security of the stream ciphers **[64]**.

This model again the design is partly the stream cipher and partly the block cipher. It is a stream cipher that is combined with block ciphers to improve its security features. It is claimed to be secure but still no literature is available on the cryptanalysis of this algorithm. The major drawback in Fish Tailed Lion Algorithm is that it can neither be used as a replacement of the block cipher nor as a replacement of the stream cipher.

2.1.3 SNOW 2.0

Ekdahl P. et al (2003) **[10]** redesigned SNOW 1.0 that was a New European Schemes for Signatures, Integrity and Encryption (NISSIE) project and has been used in 3G, GSM and military applications. Serious cryptographic loopholes in SNOW 1.0 were addressed to remove them in the new version SNOW 2.0. Most of the part of SNOW 2.0 is similar to SNOW 1.0 but major changes have been made in the Finite State Machine (FSM). It has the following variations over the original SNOW 1.0:

i. The word size in both versions of SNOW is same i.e. (32 bits)

ii. Linear Feedback Shift Register (LFSR) length is 16, but the feedback polynomial is different from SNOW 1.0.

iii. The operation of both versions i.e. SNOW 1.0 and SNOW 2.0 is slightly different. In SNOW 1.0, the first symbol is read out before the cipher is clocked after the key initialization, but in SNOW 2.0, the first symbol is read out after the cipher is clocked once.

iv. SNOW 2.0 involves two different elements in the feedback loop, i.e. α and α^{-1}.

v. The Finite State Machine (FSM) has two input words instead of one, taken from the Linear Feedback Sift Register (LFSR), and the running key is generated by XOR between FSM output and the last entry of the Linear Feedback Shift Register (LFSR).

The Pseudorandom Number Generator (PRNG) of SNOW 2.0 shown in **Figure 2.2** [**10**]

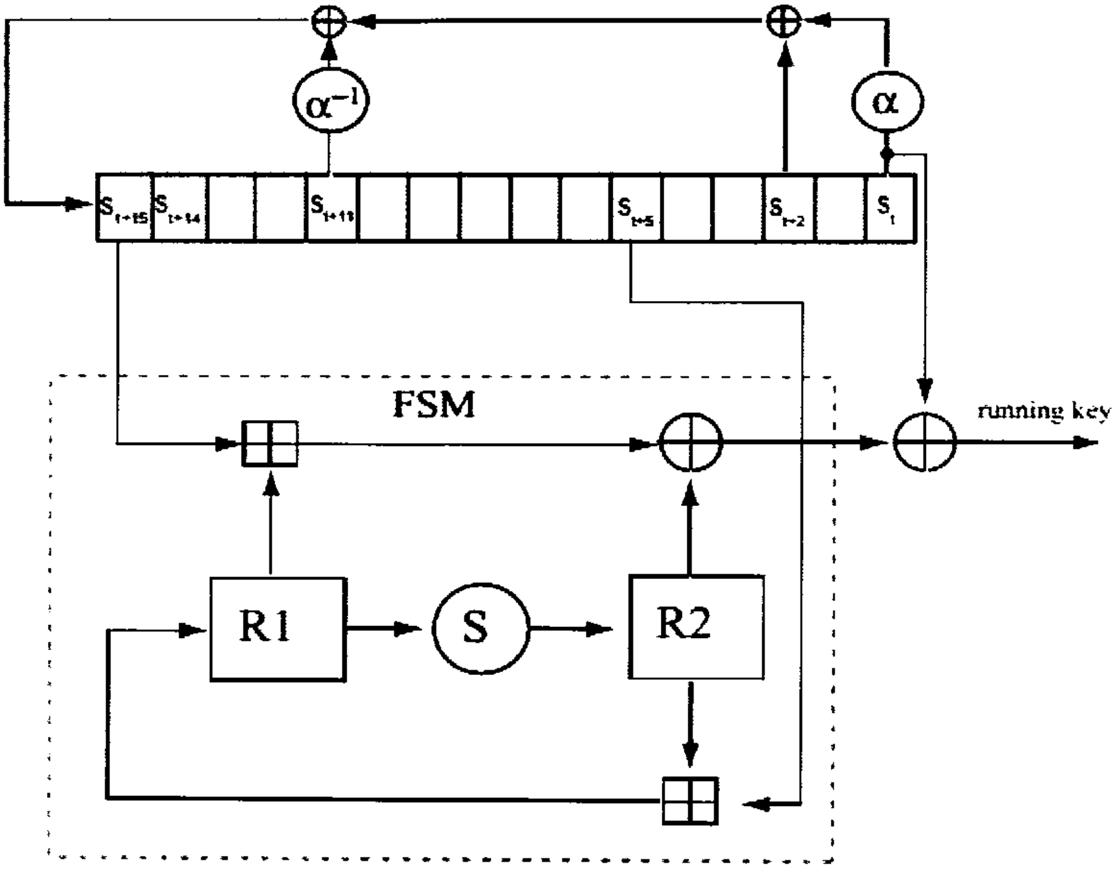


Figure 2.2: Diagrammatic Representation of SNOW 2.0

⊞ Addition mod 2^{32}

⊕ Bitwise XOR

The Pseudorandom Number Generator (PRNG) of SNOW 2.0 initializes the key as follows [**10**]:

i. Initially key initialization is performed. This operation provides starting states to Linear Feedback Sift Register (LFSR) as well as to assign initial values to the internal Finite State Machine (FSM) registers R1 and R2.

ii. SNOW 2.0 takes two parameters as input value, the publicly known 128-bit initialization value IV and a secret key of either 128 or 256 bits.

iii. The IV, i.e. $IV = (IV_3, IV_2, IV_1, IV_0)$ value is considered as a four word input, where IV_3 is the most significant one.

iv. The initial vector IV, possibly ranges from $0......2^{128}-1$, means that, for a given key *K*, SNOW 2.0 implements a pseudorandom length increasing function from the set of IV values to the set of possible output sequences.

v. The secret key denoted by $K = (k_3, k_2, k_1, k_0)$, in 128 bit case, where k_i is a word and k_3 is the most significant word. The first half of the shift register is initialized with *K* and IV as follows:

$$\begin{array}{lll} s_{15} = k_3 \oplus IV_0 & s_{14} = k_2 & s_{13} = k_1 \\ s_{12} = k_0 \oplus IV_1 & s_{11} = k_3 \oplus 1 & s_{10} = k_2 \oplus 1 \oplus IV_2 \\ s_9 = k_1 \oplus 1 \oplus IV_3 & s_8 = k_0 \oplus 1 & \end{array}$$

and the second half of the shift register as follows:

$$\begin{array}{lll} s_7 = k_3 & s_6 = k_2 & s_5 = k_1 \\ k_4 = k_0 & s_3 = k_3 \oplus 1 & s_2 = k_2 \oplus 1 \\ s_1 = k_1 \oplus 1 & s_0 = k_0 \oplus 1 & \end{array}$$

It is easy to implement reaching speed of 3Gbits/Sec on Intel Pentium 4 Computer.

vi. The word size is 32 bits and LFSR length is 16 as it is in the original cipher but the feedback polynomial has changed **[8] [10] [65]**.

vii. The Finite State Machine (FSM) takes both inputs from LFSR and the running key. This key is made up of the exclusive or XOR between FSM output and the last element of the Linear Shift Feedback Register (LFSR).

2.1.3.1 Cryptographic Weaknesses

SNOW 2.0 is an advanced system that estimates the biases more accurately in the linear approximation of Finite State Machine (FSM). As a result most of the successful attacks over SNOW 1.0 are impractical over SNOW 2.0. This improved bias estimate, proves that the linear distinguishers with bias $2^{-86.9}$ is significantly stronger and distinguishes the output key stream of SNOW 2.0 of length 2^{174} words from a truly random sequence with workload 2^{174}, which is more stronger than the distinguishing attack **[8] [37] [66]**. SNOW 2.0 was claimed that it produces a secure, minimal biased bit sequence those were observed in the bit stream of produced by its Pseudorandom Number Generator (PRNG) of SNOW 1.0. A recent fast correlation attack over the extension fields of SNOW 2.0, (an ISO/IEC 18033-4 ISO Standard Stream Cipher) is established by *Zhang B. et al* (2015) **[67]**. It is 2^{49} times faster than best known attack on SNOW 2.0 published in **[68]**. This attack again proves that SNOW 2.0 is not secure and correlation attacks over extension fields are more successful than the other known attacks.

2.1.3.2 Guess-and-Determine (GD) Attack

The Guess-and-Determine attack is a very effective and general attack that is applied on approximately all the stream cipher algorithms **[7] [69] [70]**. This attack initially guesses the contents of some of the cells and obtains the states of all the cells of the ciphering system and comparing the running key sequence with the resulting key. If the guess is proper then an approximate sequence is taken otherwise try another guess to find out the sequence those are similar to that of the original one. The time complexity of the Guess-and-Determine attack is similar to that of the exhaustive search algorithms, but the advantage here is that if the initial guess is more appropriate then the probability of success is improved. Because of these reasons the Guess-and-Determine attack is most recommended and often implemented heuristically **[31] [40] [68]**.

Guess-and-Determine attack and Correlation attacks are still applicable on SNOW 2.0. The Pseudorandom Number Generator (PRNG) of SNOW 2.0 generate a sequence of bits having improper "0" and "1" mixed together those may lead to cryptographic vulnerabilities [7]. The bit streams in Modified SNOW 2.0 are comparatively more organized and highly resistant to Guess-and-Determine attacks and increasing the time complexity to 2^{279} **[34] [37] [68]**. The cryptanalyst require more overhead to compromise the security of modified SNOW 2.0, because the statistical weaknesses cannot be easily determined in modified SNOW 2.0. The initial guess is given to the Guess-and-determine attack which determines the relationship between the internal values of the key and the cipher is broken completely **[40] [71]**. Similarly, the linear masking attack that requires 2^{225} output words (2^{230} bits) and 2^{225} steps of analysis to distinguish the output of SNOW 2.0 from a true random bit sequence **[13]**. In this attack a series of bits are selected to seek the correlation of selected bits with rest of the bit patterns in a given key to determine another part of the key. This attack greatly supports the Guess-and-Determine attacks which focuses only over guessing a part of the key and determine some other sequence of bit patterns.

2.1.4 ORYX

ORYX algorithm is used to encrypt data in digital cellular phones when data are sent from one phone set to another. It is based on three 32-bit Galois, Linear Feedback Shift Registers (LFSR) and is different from the block cipher CMEA that is used to encrypt the cellular data control channel. D. Wagner, J. Kelsey, B. Schneider belong to cryptographic tag-team from Counterpane systems have developed an attack on ORYX. The attack was successful and requiring about 2^{16} guesses on 24 bytes of known plaintext **[31]** **[52]** .

Wagner D. et al (2002) **[52]** presents a very strong attack on the security of ORYX. Using this attack the full 96-bit internal state can be directly recovered irrespective of the key schedule. The North America Cellular system is using ORYX to encrypt data that is transmitted in the channels. All the above discussion proves that a very low level security is provided by the ORYX algorithm, and is strictly discouraged in the second generation and third generation mobile communication.

The ORYX cryptosystem has 96-bit key space to guess the generator's initial state as a whole. As it is not feasible, so the cryptanalyst divide the initial generator state in to smaller parts and then implants the guess on these small parts of the keys and continuously checks whether the guess is correct, hence the entire key is compromised. The following are the two main points in the attack:

i. If the cryptanalyst knows 25 bytes of the key stream, then the probability of success is 99, where the attack requires exhaustive search over 16 bits.

ii. If more than 25 bytes of the key stream are vulnerable to the analyst, the probability of success increases even more.

If the cryptanalyst has access to more than 25 bytes of the key stream, the probability of success is increased. It proves that the cryptographic algorithm ORYX provides very poor security, and most of the second generation mobiles and telephonic devices are insecure and the security may compromise if some secret information is transmitted.

2.1.5 RC4

It is a very important stream cipher algorithm, from RSA Security, Inc. It was a trade secret and the US military but the source code was anonymously published in 1994. This algorithm was performing similar to that of RC4 used in different devices launched by the official RSA products. It is widely used and a large number of products are made secure using RC4. There were no known security arracks on this algorithm as assumed until 1994. It is not developed in RSA laboratories but RSA was using it as trade secret. The 40-bit exportable version of RC4 was broken in Kremlin using brute force attack **[36]** **[53]** [72] [73].

2.1.5.1 Strength

The RC4 algorithm gives theoretical and practical security, and has number of features:

i. It is very difficult to know that where the required values is located in the table.

ii. It is difficult to know that which location in the table is used to pick up the values for the sequence.

iii. The key is generated only once by the RC4 pseudorandom number generator.

iv. The encryption scheme of the RC4 algorithm is 15 times faster than Data Encryption Standard Algorithm (DES) and other block ciphers **[36]** [72].

2.1.5.2 Weakness

Couture N. et al (2004) **[36]** argue that although RC4 is a widely used stream cipher but numbers of effective attacks are available, which have compromised the security of RC4. *Johansson T. et al* (2003) **[29]** and *Vladimor V. C. et al* (2002) [74] present some effective brute force attacks on RC4 and a 40-bit can be recovered in few minutes. He also presents another efficient attack on RC4, in which the author claims that RC4 is fully insecure in the natural mode of operation, and must be rejected for using in widely deployed WEP (Wired Equivalent Privacy Protocols, which is a part of the 802.11b Wi-Fi standard.)

The RC4 algorithm has the following issues in its structure:

i. The keys produced by RC4 are strong enough, but in every 256 keys there may be a weak key. The analyst identifies the keys and checks the correlations of these keys with the others and on finding the correlation the attacks are generated [75]. The reason for this attack is that the state table is vulnerable to analytical attacks.

ii. The cryptanalyst finds the correlation among the keys produced by the key generators, as there are circumstances that the keys are strongly correlated with the subset of the key bytes, or there can be only one key out of every 256 keys generated by the RC4 key generator.

iii. The keys generated by RC4 are weak because, if some of the patterns have been detected in the weak key will compromise the entire key. When a sub part of the key generated by Key Scheduling Algorithm (KSA) of RC4 is exposed, the attacker applies related key analysis attacks to find out the next part of the key.

2.1.6 SEAL

It is one of the fastest and secure encryption algorithm designed by *Coppersmith D. et al* (1998) [54] of IBM Corporation. SEAL requires only five operations per byte to generate the key stream, and need very less memory as compared to other algorithms of the same category. It is best recommended for encrypting large amount of data on hard drive or where the ciphering data is read from the middle of the channel in a ciphering communication.

It is a binary additive stream ciphers proposed in 1993. It is a software optimized encryption algorithm. The cryptanalysts did not give proper attention to SEAL. However, some attacks are proposed which are very effective on the simplified version of SEAL. This stream cipher is specifically designed for 32-bit processors, with the capability of very high speed encryption and decryption process.

The simplified version of SEAL has serious cryptographic vulnerabilities and effective attacks can be launched on it. The attack on simplified version of SEAL is described in the following four steps.

i. Derive the unordered set of values from the T-table of the unknown 32 bit constant D^i

ii. Compute constant α^i is required to find the statistics involving B^{i-1}, D^i, A^i and B^i in the third step.

iii. Use the values of n^i those have been found and constant values to make the relation between input and output values in the T-table.

iv. Compute the input and output values of the T-table.

If the T-table is broken down, then the security is compromised, which is done in about 2^{24} samples of (n, l). The simplified version of SEAL can be broken down, if we find the sets of values (n_1, n_2, n_3, n_4). The security of T-table can be broken by generating randomized R-table and find the relation between T- table and R-table [**76**].

2.1.7 Rabbit

It is one of the fastest encryption algorithms and was presented at the Fast Software Encryption workshop (2003) [**55**]. Its design is based on the complex behavior of the chaotic maps having the exponential sensitivity and generates a map containing random numbers and remained long-time unpredictable.

i. It takes 128 bit long secret key and generates an output of 128 pseudorandom bits from the combination of the internal state.
ii. The encryption and decryption is done by XORing the plaintext and ciphertext with the pseudorandom number respectively.
iii. The internal 513 bits are divided into eight 32-bit counters, eight 32 bit state variables and one carry bit.
iv. The eight state variables are updated by eight coupled non-linear integer valued functions.
v. The lower bound is secured by counters on the period length for the state variables.

The two main features of Rabbit are:

i. **Security:** The key size of the Rabbit is 128 bits and encrypts 264 byte of the plaintext.
ii. **Speed:** It is a faster and commonly used stream ciphers.

A brief and comprehensive technical discussion of different aspects of stream ciphers is presented in [7]. The stream ciphers are fast but less secure than block ciphers thus the focus of designers of the Rabbit was on maintaining the speed and keeping the security of the data at the highest level. One of the drawbacks of entire scheme of stream ciphers is that their main focus is the key only. If someone is unable to get a truly random key, then none of the stream ciphering will guarantee speed. A comprehensive discussion on security and cryptanalytic attacks on

Rabbit is available in [77]. *Golić J.D. et al* (2005) **[30]** proposes the cryptanalysis of the stream ciphers and proposes alternative designs. Various cryptanalytic weaknesses in the design of Rabbit are discussed in [7].

2.1.8 Turing

It is another fast encryption stream cipher method using a key size of 265-bits. It is an efficient algorithm and is designed to be used in areas involving huge computations. It is an LFSR based stream cipher used in another algorithm SOBER along with the key mixing function of a block cipher round **[56]** [78].

2.1.8.1 Security of Turing

In Truing two different aspects are combined together in a manner to achieve ultimate security. It has a long key size and cannot be broken in polynomial time because 2^{256} keys are required in the worst case.

2.1.8.2 Known Plaintext Attacks

The key generated in the Turing is independent of the plaintext, and any misuse of the stream cipher will result in destruction of plaintext. If a key stream is reused or if there are some relevant bit patterns previously used in the key stream may compromise the security and the cryptanalyst may get information about the plaintext from the ciphertext.

2.1.8.3 Statistical Attacks

The biased key stream generator may produce the weak keys which are detectable and the cryptanalyst may find these weaknesses and approach the information. The Turing design is based on the nonlinear transformation and discourages the linearity of the LFSR outputs. The designer is of the opinion that LFSR based stream ciphers are less secured as compared to NLFSR based stream ciphers.

2.1.8.4 Related Key Attacks

In these attacks the hacker gets some output from a black box, and look for the relevance of that text with the ciphertext that has been attacked. Turing key loading mechanism addresses this attack and ensures that a change in single byte will make a significant change in the S-Boxes. It also affects the in term values of the Linear Feedback Shift Register (LFSR). The attackers make a pair of keys and apply to the available plaintext and ciphertext looking for the similarity. During the process of finding such a pair, if some part of the original key is known, the hacker tries to find the key with similar characteristics. However, a key pair with an unknown pair is very difficult to produce.

2.1.8.5 Correlation and Distinguishing Attacks

The nonlinear filter of Turing is designed to perform a strong transformation, and to use significant amount of input state. The correlation among the states exists but it is sufficiently small to support the correlation attack. However, the cryptographer does not rely on this attack because it would not be successful due to security reasons. The correlation and distinguishing attacks are inapplicable on the secret key based S-Boxes. The main advantage of the nonlinear filter function is that the hacker cannot analyze them as in case of LFSR based stream ciphers where some unknown values are generated form a kind of "whitening" those are traceable and predictable in view of the designer of Turing.

2.1.8.6 Guess-and-Determine Attacks

This is an efficient attack and very successful against the weakly designed LFSR based ciphering algorithms. The nonlinear feedback shift register based stream cipher has less vulnerability to these attacks. A strong NLFSR based structure of Turing has no fear of the Guess-and-Determine attacks. In the New European Schemes for Signatures Integrity and Encryption (NESSIE) [78] project it was studied in detail and provided a minimum complexity exceeding the enumeration of the 256-bit keys.

2.1.9 A5/1

A5/1 is a strong and efficient stream cipher algorithm that provides privacy to over 200 million users in the air link of data and voice communication. It was used as a Global System for Mobile Communications (GSM) standard for security. The best published attacks against A5/1 require steps ranging from 2^{40} to 2^{45}, to compromise its security. It is best recommended in the areas where the data is attacked by software based mechanism but not recommended where the special hardware is used to implant attacks by large organizations. There are some flaws in the tap structure of A5/1, their frequent reset and their non-invertible clocking mechanism which makes the algorithm insecure and the cryptanalysts with better arrangements for reestablishing the plaintext [5]. The attacks on A5/1 can be carried out by using a single PC, which has to be applied only once after a 2^{48} parallelizable data preparation stage.

2.1.9.1 Security Attacks on A5/1

A5/1 is under the strict considerations of the cryptanalysts due to its wide spread use and implementation in the GSM technology. In these attacks the attacker assumes that he knows some part of the secret key that has been generated by the A5/1 pseudorandom number generator. It is a standard assumption in the cryptanalysis of all stream ciphers. The analyst then applies the probability and statistical approaches and tries to guess what will be the next pattern and how much probable it is. When the process completes for all chunks then he combines them together using permutation. The GSM technology sends a new frame after every 4.6 milliseconds, containing 2^8 frames, and thus finding the initial key during conversation and the rest of the conversation is decrypted [32] [33]. *Biryukov A. et al* (2002) [79] proposes two new attacks on A5/1:

i. First attack is a very serious and it computes the key in several minutes, where there is conversation among the parties for only two seconds.

ii. The second attack is even more serious and efficient if there is a conversation between the parties for several minutes. During the first two minutes the key is computed in one second approximately.

Both of these attacks are interrelated to each other, but they use different types of time-memory tradeoff. These attacks are practically applied and verified to be efficient, but if there are special arrangements such that the data is covered by other cryptographic measures too, the attacks cannot be executed completely. These attacks has proved that A5/1 cannot be used any more for achieving better security, and to protect the voice communication channel from data leakage.

2.1.9.2 Informal Description of the New Attacks

The key idea of the two attacks is given below:

i. They use the Golic time memory tradeoff.

ii. They identify the states from the prefixes of their output sequences.

iii. The A5/1 algorithm can be efficiently inverted.

iv. The key can be extracted from the initial state of any of the frame.

v. The Golic attacks cannot be practically applied on A5/1.

vi. Special states are used, which can be efficiently sampled in A5/1.

vii. Biased birthday or Hellman's time-memory tradeoff attacks are used on a sub-graph of special states

viii. A5/1 is very efficient on a PC, using parallelized stages for the software based cryptanalysis.

These attacks are discussed in detail in [32] [33] [79].

2.1.10 E0

The Bluetooth protocol uses E0 algorithm which is a proficient stream cipher algorithm, and is specially designed for Bluetooth supported devices and applications. The internal work E0 is described below:

i. The data is combined with the sequence of pseudorandom numbers generated by the algorithm. A 128 bit long key is used but the size of the key may vary.

ii. E0 generates one bit per iteration using four shift registers of length 39, 33, 31, and 25 bits and two internal states each one is 2 bit long.

iii. At each clock tick, the registers of A5/1 are shifted and its two states i.e. the current state with the previous state and the values in the shift register are updated.

iv. From the shift register four bits are extracted and added together. The 2-bit register and the four bit shift registers are XORed, where the first bit of the result is output for the encoding.

It is used for link encryption in telecommunication based on stream cipher [57] that is used for radio network link called Bluetooth. *Patrik E. et al* (2003) and others [5] [19] [57] analyzes E0 in different ways and prove that it is not secure and they recommend that should not be used for secure communication in telecommunication sector.

Lu Y. et al (2004) [57] proposes different algorithms to attack E0. Different algorithms have been proposed to break its security and it has been shown that E0 is no more secure. The time complexity of the proposed attacks in the worst-case time is $O(2^{38})$.

The effective attacks over E0 are the correlation attacks that break its security. The design lapses in E0 let the analyst to plot an effective attack. The initialization scheme of E0 is insecure as it takes place in single step and does not strengthen the resultant key and the cryptanalyst successfully break down the E0 security. It is not recommended for use in dedicated stream ciphers, and proves that E0 is no more secure, as the known-plaintext attack on Bluetooth encryption.

2.1.11 COS

COS is another fast and secure stream cipher algorithm based on Nonlinear Feedback Shift Register (NLFSR), proposed by *Filiol E. et al* (2001) **[58]**. *Babbage S. H. et al* (2001) **[80]** **[81]** shows that Mode II COS (2, 128) is extremely weak. He has proved that Mode I of COS (2, 128) is also extremely weak, where the secret information could be recovered easily with a known plaintext of size 216-bits. He describes a very efficient attack and claims that the stream cipher of COS family are all extremely weak and states that the described attack requires successive clocking amounts of 64 and 65. In fact it is easy to see that the attack can be generalized to work for any two successive clocking amounts that are not both equal to 64. Given any three consecutive blocks of known key streams, with ¾ probability there is an attack that recovers the entire state of the registers with negligible effort. He claims that it is not a weak cryptosystem but it cannot be trusted any more.

2.1.12 Scream

Scream is another efficient and secure stream cipher algorithm. Its designers were inspired by SEAL. It is more efficient and secure than SEAL **[59]**. The input to the first stage of the Scream is a short random string and the pseudorandom generator of the Scream expands it into a longer string such that the resultant key is still random and the adversaries with limited resources cannot break it. The pseudorandom number generator of Scream accepts a short input string called seed (or key) and generates a longer output string called output stream (Key Stream). The key is used only once for a single session and for the next session it is recommended to generate new key. The key cannot be used for longer session because the cryptographers may get enough time to break the security of Scream [7] **[82]**. The main goals achieved by the Scream are the following:

i. Scream is more secure than SEAL. The same seed can be used for 264 bytes of output. To distinguish a cipher from a truly random number, the attacker would elapse more time than that of the precise time, while looking at the initial 2^{64} bytes of output. It is impossible to break it down, due to 2^{80} key spaces and 2^{96} times, and the security of the Scream is above these values.

ii. Scream is comparatively faster, i.e. on common PC's it is about 5 cycles. It has better performance over the 32-bit and 64-bit architectures.

iii. Full 128-bit input nonce is allowed for Scream and 32-bit nonce in SEAL is allowed.

iv. Scream is more suitable to be used in other implementation where SEAL is failed to maintain the security, as it is more amendable and easily adjustable there too, for example in smartcard etc.

Two distinguishing attacks are proposed on Scream Family:

i. The best one has 2^{80} time complexity, but no specific attack has been proposed for Scream. However, linear distinguishing attack can be applied if a sequence of 2^{80} output bytes is available. Based on the linear approximation of the nonlinear *S*- boxes the distinguishing proves that some of the parts of the S- boxes have weak entries which make the stream cipher vulnerable to cryptanalyst.

ii. The second attack is due to the improper distribution of the values in *S* boxes create problems. The attack is established which uses 2^{105} output words approximately and the complexities of the similar size.

The linear distinguishing attack is faster than the exhaustive search attacks for the Scream of 128 bit keys. A full security cannot be obtained by Scream, means that none of the known attack is applicable in polynomial time, but in case of the Scream Exhaustive key search and Linear distinguishing attacks are the known attacks which may compromise the security of Scream stream cipher, and the world cannot rely on Scream to use in industrial products.

2.1.13 HC-256

HC-256 is an efficient, fast and secure stream cipher algorithm. It requires low memory and resources and 4 to 5 times faster than other stream ciphers of its family. It is very simple to implement and encryption is done by XORing the key stream with the plaintext. It updates one element of a table with non-linear feedback function in the two secret tables with 1024 32-bit elements. It is a software efficient stream cipher having superscalar features of modern and new microprocessors [**60**] [**61**] [**62**].

The linear distinguishing attack on the key stream of HC-256 has been applied with no linear masking. The weak feedback functions are analyzed and was proved that the key generator generates a random numbers of 2^{128} bits. It has the characteristics of truly random sequence. None of the known attack is computationally feasible as the designer of HC-256 claims. The cache-timing attacks are later on proven to be feasible on HC-256 and guess the key by taking the random seed required for key initialization. Hyper threading attacks and process-interruption attacks also have significance to break it security [**31**] [**38**] [**83**].

2.2 Security Issues in Stream Ciphers

Stream cipher algorithms security is mostly dependent on key stream and must not be discovered by any cryptanalytic attack. The key space must contain enough keys that randomly generated keys could not be reproduced in polynomial time. The cryptographers also require that not only the entire key but the subpart of the key would also not have biases that would let the attacker to distinguish a stream from any type of random noise. The ciphertext must not have any relationship with the plaintext that could trace a secret key as a result the resultant ciphertext is generated. The cryptographer should not get any clue to find the relationship among the chosen plaintext and resultant ciphertext. A stream cipher can be guaranteed that it is unbreakable for some of the cryptographic attacks and there may not have practical ways that it will be broken down but might have other weaknesses which may lead it to break down in future.

The sequence of bits produced by PRNG is treated as secret and is used only once for a communication channel and should not be used twice. The key is used for short term communication and should not be used for long period of times as the prolonged period lead to security leaks. The application designer must also recognize that the stream ciphers are mostly used for confidentiality and not for authenticity. The encrypted message has the threat to be modified in transit, and the designer need to take special actions to overcome such a situation. Short period stream ciphers have practical concerns:

i. If the key size is very short then Brute Force and other Algebraic attacks may compromise their security.

ii. If encryption is being performed at a rate of 8 megabytes per second, a stream of period 2^{32} blocks will repeat after about a half an hour.

iii. The weak implementation of a stream cipher may let the analyst to guess the key by implanting chosen plaintext or ciphertext attack.

The designer must also confuse the relationship among the key used and resultant ciphertext. The stream cipher RC4 that was used as a trade secret by US department of defense till 1994 is attackable because of weaknesses in its key setup routines and new applications strictly discourage the use of RC4 algorithm to be used for security purposes [**75**].

2.3 Genetic Programming in Cryptography

Cryptography is one of the most significant techniques in the field of information security. The use of evolutionary computation to analyze the cryptology is the product of evolutionary computation and cryptography. Over the last few decades many researchers have made lot of achievements. Evolutionary computation techniques include genetic algorithms and artificial neural networks and their role cannot be ignored in cryptography. A number of stream cipher algorithms are designed based on Artificial Intelligence. **Section 2.3.1** discusses the work done in this area so far.

2.3.1 Genetic Programming Based Stream Cipher

Spillman R. et al (1993) [**84**] used Genetic Algorithm (GA) and he used simulated annealing method for the sheet substitution cipher of analysis. *Clark J. A. et al* (1998) [**85**], *Van V. J. H. et al* (2003) [**86**] added their contributions in designing and developing more clever genetic techniques. They developed tabu search algorithm and particle swarm optimization algorithm for sheet substitution cipher analysis. *Servos W. et al* (2004) [**87**] used genetic algorithm (GA) for the analysis of multi-Chart substitution Vigenere cipher. They used genetic algorithm to identify and determine the parameters of the multi-chart substitution ciphers. *Peleg S.et al* (1979) [**88**] decoded substitution cipher using a technique called relaxation algorithm.

Albassal M. B. et al (2003) **[89]** used genetic algorithms (GA) for deformation Feistel type cipher analysis, in addition to this, he also used the same method and cryptography parameters for the deformation of SPN cryptanalysis. *Al-Salami M. et al* (2004) **[90]** introduces a genetic algorithm (GA) and timing attacks (time attack) on RSA cryptosystem. *Nalini N. et al* (2005) **[91]** used genetic algorithms for the cryptanalysis of the S-Boxes. Combining cryptanalysis with evolutionary computation methods is still a very challenging and innovative research direction; however a lot of questions need to be answered **[85]**. *Clark J. A. et al* (2004) **[92]** pointed out that the analysis of the evolution of modern cryptography is a very difficult and time consuming task.

Khaled M. G. et al (2005) **[93]** presented a cryptographic model based on general regression neural networks (GRNN). The model is designed in three layers; the encryption process is performed by one set of neurons which consists of 3 nodes representing n-bit blocks, the pattern layer has 8 nodes and the output layer too has 8 nodes which are used to identify the decrypted output message. The simulation results have shown a very good result, with relatively better performance than the traditional encryption methods, but without any knowledge about the security of the proposed model.

Shihab K. et al (2006) **[94]** presented a new asymmetric encryption mechanism based on artificial neural networks.

Necla O. et al (2007) **[95]** presented a new data security approach over electronic communication based on artificial neural networks (ANNs). *Dalkiran I. et al* (2010) **[96]** proposed a chaotic cryptosystems based on artificial neural network and chaotic generator synchronization.

Karam M. Z. O. et al (2011) **[97]** presented a stream cipher system based on pseudo Random Number Generator (PRNG) using artificial Neural Networks (ANN). *Volna E. et al* (2005) **[98]** presents their work on multi-layer neural networks in cryptography. The multilayer neural networks modified by back-propagation. The planned model converted the input message into ASCII code then gets the sequence of bit for each code which divided into 6 bit blocks are used as input for the encryption process. The cipher key is the neural network structure contained input layer, hidden layer, output layer, and updated weights.

2.3.2 Issues in GP-Based Stream Ciphers

Designing an automated ciphering system is a complex problem and a number of researchers are working on it. Genetic algorithm (GA) has been used to find a set of rules of Cellular Automata (CA) suitable for cryptographic purposes [99] **[100]**. Also, GA has been used for the construction of Boolean functions for ciphering systems **[101]**. The design of Boolean functions with properties of cryptographic significance is a hard task because the cryptanalyst while finding a sequence that how the next bit is generated will guess the next coming bit in the sequence. This problem has attracted a number of researchers **[102]** and they propose a GA based method for finding Boolean functions which have mostly high degree of nonlinearity. A general automated method for designing a stream cipher is not known and is reviewed in detail by *Ratten R. et al* (2014) **[103]**.

The stochastic optimization and search algorithms can be easily simulated due to genetic algorithms. The genetic programming is suitable for cryptographic algorithms because no analytic expression exists for the analysis of the cryptographic algorithms. The genetic programming can greatly shorten the time of cryptanalysis and improves cryptanalysis efficiency. The genetic algorithms are used in many more applications because of its advantages but it has some prominent drawbacks too. *Zhang J. et al* (2013) **[99]** proves that it performs poorly if it searches for the global optimal solution near stagnation or for forward compatibility and performs slowly and very inefficiently and describes its observations as follows:

i. The neural network with chaotic logistic map is used for cryptography by which both partners use the neural network as input for the logistic map that generates the output bits to be learned.

ii. The General Regression Neural Network (GRNN) is used for encryption and decryption process based on three layers, where the input data divided into 3 bits and 8 bits as output.

iii. The training back propagation neural network act as public key, while Boolean algebra act as private key.

iv. The chaotic hope field neural network with time varying delay was used to generate binary sequence for making plaintext, which considered as a random switching function for chaotic map.

v. The neural network based on chaotic generator is used to generate chaotic dynamic that acts as a shard key.

vi. The initial weight value of the neural network is used after training as symmetric key. The chaotic neural network is used to generate chaotic sequence act as a triple key (combined of initial condition and control parameters) for cryptography.

2.4 Neural Cryptography

Artificial Neural Networks is field of artificial intelligence developed due to the inspiration from the biological neural model of human brain, where the operations are taken and understood like human brain and neuron system is called artificial neural network. An Artificial Neural Network is an interconnected group of nodes, similar to the vast network of neurons in a brain [**104**]. The circular node represents an artificial neuron and an arrow represents a connection from the output of one neuron to the input of another as shown in **Figure 1.6.**

Artificial Intelligence and Neural Networks are used in almost every sphere of the computer applications but limited literature is available on its application in cryptography in general and stream ciphers in particular. It is a powerful tool and can be used effectively due to its following advantages:

i. The processes that cannot be performed by linear programming are done by neural network.

ii. If one of the neuron fails to perform the other neurons are not affected and they work properly.

iii. No reprogramming is required and the advancements are adjusted due to learning.

Artificial Neural Networks (ANN) on the other hand has the following disadvantages:

i. A very boring and long training session is required, but the problem does not rest only in ANN but remains in all other AI models too.

ii. In some applications it requires an indefinite period of time, to make a system trained, because making rules identical to human sometimes become impossible.

iii. Large neural networks it requires high processing time.

Beavers A. F. et al (2013) [**105**] presents that Alan Turing in early 1950's for the first time conceived the idea of Artificial Intelligence that computer processes can behave and process like human brain. Turing's this hypothesis remains a vision and many researchers were inspired and worked in this area. It is the designer choice to choose the performance function, network topology, the learning rule and the criteria that when the system should stop training phase. It is difficult to design a priori algorithms and if the system malfunctioning then the readjustment is very hard.

For computational intelligence there is no standard definition that could be adapted and accepted by the researchers for the computers system to make able to behave intelligently in complex and changing environments [**49**]. The biggest advantage of the computational intelligence is that, it has the ability to learn, adapt new situations, discovery, association and arbitration, automatic modification and quick response in hazardous environments. The Computational Intelligence Model (CIM) systems are based on the human biological system, where there is a central decision making organ along with a series of local and central transmitters i.e. neurons and Knowledge Base and the Semantic Net is processing all these information by taking new suitable decisions. The evolutionary system were supplemented with the elements of reasoning so as to get the computational intelligence also later on the fuzzy logics and artificial neural networks were merged together and the modern intelligent computing system were developed [**106**].

Some stream ciphering algorithms claim to be intelligent such as Golden Fish [**63**] and Fish Tailed Lion [**64**]. These are based on merging the stream ciphers with block ciphers so as to improve the security. However, they do not come under the domain of Artificial Intelligence. An intelligent system contains some memory and logic, and the decision is based on the past knowledge, where logic interprets new decision. Golden Fish and Fish Tailed Lion are designed to combine stream ciphers and block cipher characteristics and present a new model. The designers claim that these are more secure than ordinary stream ciphers. Some other algorithms claim to be intelligent such as A5 [**32**]. It does not use Artificial Intelligence techniques and only its internal structure has been modified.

The emphasis of this research work is to use more sophisticated and advanced AI techniques to design security algorithms. Stream cipher algorithms those are comparatively weak but fast and best recommended in low memory and noisy communication are a big challenge for the modern world to improve its security. This is achieved to avoid bit patterns which are vulnerable to the cryptanalyst.

Neural cryptography is the branch of cryptography dedicated to analyze the security of encryption algorithms used in the field of cryptography and cryptanalysis. This approach uses ciphering algorithms based on functions those could be reproduced. Neural Cryptography is used in the key exchange protocol to increase the synaptic depth the analyst pay more cost by generating a successful attack [**107**] [**108**].

Different applications based on Stream ciphers for security are now trying to shift to block ciphers due to the inherent weaknesses in the stream ciphers. However, direct conversion to block ciphers creates more problems [**35**] [**65**] [**109**]. The information security conversion in block ciphers is a direction that researchers utilize the artificial intelligent techniques to optimize its application according to the requirement. Expert systems and machine learning enforce the security of Internet and wireless networks, which become scalable and flexible in their application structures. So that intelligent information security is a new field that combines artificial intelligent methods in cryptography. Furthermore, the traditional block cipher algorithms adopt some intelligent technologies to build trusted computing platform. Besides the security factor there are other factors such as speed and quality of service in the application of block ciphers. To design more secure system as compared to existing block ciphers is little costly to get a wider block cipher mode. There are still many other related fields that must pay more attention to the application view, such as the size of network platform and type of networks. The Internet security component structure is based on TCP/IP which is simpler than OSI model of ISO and is scandalized by some big organizations [**109**]. If the key size is increased a stream ciphering algorithm may results in the following [**8**] [**110**]:

i. Increase the complexity of the linear attack

ii. The depth of the key space increase exponentially

The differential key attacks Guess-and-Determine attacks and other clever attacks may be more effective if the key characteristics remain the same.

2.5 Problem Statement

The stream cipher algorithms are comparatively less secure than block ciphers. The block cipher algorithms are used where we have enough memory and computational power. A long sequence of bits can be used as a key and the encryption and decryption process can be made more complicated by confusing the given input *K* and plaintext *P*. The Encryption process is a time consuming job and the ciphertext is produced after passing the plaintext and key from several rounds. There are very less chances of successful attacks on modern block ciphers.

The steam ciphers by nature have limited key size and the ciphertext is produced by simply XORing the plaintext *P* and key *K* because they are used in limited memory and low computational power environment. The cryptanalyst by applying clever attacks may have comparatively more chances of success to recover the key or the plaintext from the ciphertext without key.

Stream ciphers are mostly used in real-time communication systems where the cryptanalyst has limited time for a successful attack. A weak key can let a cryptanalyst to become successful in breaking the stream cipher algorithm very quickly. The cryptanalyst looks for the loopholes in the implementation process of the algorithm and tries to apply different cryptographic attacks by getting the information about the used stream cipher algorithm. By just changing the key size or the key space is not enough to enhance the security of an algorithm, because if we ignore other parameters, then the entropy may reduce and a statistical attack may become successful. So, it is not the appropriate solution to achieve an unbreakable system. The key stream generated may have statistical weaknesses that may lead a cryptanalyst to apply a successful attack.

The reason for compromising a stream ciphers is the weak key. The cryptanalyst may break the security of a stream cipher algorithm by finding the expected key that has been used for the encryption process. Most of the cryptanalytic attacks discussed so far are targeting weak keys. This work focus on weak keys those may lead a cryptanalyst to break the security of SNOW 2.0, a stream cipher algorithm.

2.6 Problem Definition

One of the most important classes of symmetric-key encryption scheme is the stream ciphers. They are termed to be simple block ciphers in one sense having block length equal to one bit, computer-word, or byte. The plaintext to encrypt using the stream ciphers is very simple than a single bit, word or character is encrypted at a time. It is useful in situation where the noise or distortion may affect the transformed data and the transmission errors are highly probable. In stream ciphers each bit is encrypted independent of another bit by simple XORing the key stream with it, so there is no chances of error propagation in this scheme. It is used in the environments where the data must be processed by one symbol at a time, for example the devices having less processing power and no memory or limited data buffering.

By just making changes in the design of the algorithm does not give us the ultimate security while changing the logic may for the time being make your model secure however, it may lead to some more sophisticated and dangerous situations as it has been observed in cases of DES and SNOW family of algorithms **[65]** [**69**] [**70**]. Stream cipher algorithms operate in two major steps:

i. Pseudorandom Number is generated by Pseudorandom Number Generators(PRNGs)
ii. The resultant sequence is treated as secret key and the plaintext and the secret key is XORed together to generate ciphertext.

The whole security of stream cipher algorithms is dependent on the key stream because the ciphertext is generated by simply XORing the plaintext with the secret key. As discussed earlier that none of the stream ciphers fully satisfy the randomness verification tests recommended by National Institute of Standards and Technology (NIST). That's why these ciphering algorithms are susceptible to different cryptographic attacks and cannot be trusted to use them for secure communication. Cryptographers suggest different methods to secure stream cipher algorithms, for example they suggest that the PRNG must have a sound mathematical foundation and the key size must also be large enough that it should not be traceable in polynomial time.

A strong key is required to achieve unconditional security. The security of a stream cipher is compromised if there is a loophole in the key stream that has been used in the encryption process. The cryptanalysts are always in search of these loopholes which may support their attempt for a successful attack. The PRNGs ensure that the key sequence of bits produced is random. This work considers SNOW 2.0 for research and implant cryptographic attacks on key stream produced by the SNOW 2.0 to check its soundness towards these attacks. The basic purpose of this study is to highlight:

i. The security of cryptographic schemes is improved then on the other hand we may result the cryptosystems with weaker operational efficiency. We need to be careful about all such improvements and we need to find ways to improve the security with minimal operational cost.

ii. The Pseudorandom Number Generator (PRNG) of SNOW 2.0 produces a series of keys that can be traced by the cryptanalyst. Once the cryptanalyst gets knowledge, he plans for the attack to recover the plaintext without having the original key.

iii. In Pakistan E-commerce industry, banking sector and national defense systems cannot completely trust on existing methods developed by other countries for securing their databases and communications. It is dire need to develop our own encryption algorithms and techniques to be used in different areas to secure data of national interests from criminals and enemies.

iv. Encryption models discussed in this chapter have mathematical weaknesses as described. Some well-reputed algorithms claiming to be very strong were found to be very weak like Rabbit, SNOW 1.0, SNOW 2.0 and HC-256. These threats results in that these stream cipher models are not suitable for unconditional security.

v. The loopholes and application backdoors in the cryptographic models make the stream cipher models susceptible to strong cryptographic attacks and if not used properly in a controlled environment then it may lead to complete breakdown of these models. The uncontrolled repeating bits in key generation and the ciphertext may compromise their security [65] [70]. The loopholes are sometimes left by the designers due to ignorance or innocence that may give advantage to cryptologists later to break its security.

2.7 Summary

Most of the modern stream cipher algorithms those claim to be secure have been compromised after some period. These algorithms are based on logics and the cryptanalysts tries to find out the loopholes in their design to break their security. They apply different cryptanalytic techniques to recover the plaintext without having a key and try to guess a part of the key and recover the remaining part of it. The designing weaknesses open doors to the hackers to get secret information from a secure system. A number of logic based stream ciphering algorithms failed due to their design weaknesses. It is also not always the case that the designers have designed a weak system but the other reason is that the advancement in the cryptanalytic research has also made the life of the designers of the cryptosystems very difficult. Everyday new advancements in the field of cryptanalysis threaten their so called secure security systems.

Genetic programming based stream cipher algorithms discussed are used either for cryptanalysis or generating new stream cipher algorithms. If a cryptographically weak key is produced these algorithms does not have the capability to check it prior to deliver it. Concrete models that may cover all security aspects without losing the main features still need more concentration. These algorithms too have the security drawbacks as discussed in this chapter. Golden Fish algorithm and Fish Tailed Lion algorithms claim that these are intelligent but actually they fuse block cipher operation modes with stream ciphers to improve their security. *Stankovski P. et al* (2013) [**111**] and *Wu H. et al* (2008) [**60**] proved that the clever cryptographic attacks break the security of weak stream cipher algorithms. A number of researchers have shown their interest in the field of neural cryptography and a numbers of publication are discussed here in this chapter, but no significant work is done in designing a secure stream cipher algorithm.

CHAPTER 3
BIT PATTERNS IN STREAM CIPHERS

A true random key is the combination of a sequence of balanced zeros and ones that is the randomly generated key has equal number of zeros and ones. The occurrence of both zeros and ones is n/2, where n is the total number of bits in the key. The keys generated by the Pseudorandom Number Generator (PRNG) of the stream cipher SNOW 2.0 require passing the randomization tests. A randomly generated key does not have necessarily all characteristics required in a true random number and if such a key is used for encryption of a secret message may have to be compromised.

National Institute of Standards and Technology (NIST) recommends 16 different statistical tests for the verification of random number. These tests ensure whether a key is random. These NIST recommended tests mostly check the existence of zeros and ones and the interrelationship of one part of the key with another part of the key. The frequency test (Monobit) test checks the pattern of length 1 bit in key and its occurrence in a given key stream. The frequency test within a block checks the proportion in ones within M-bit block. The purpose of this test is to determine the frequency of 1 in a given pattern where a pattern is M-bit long and the frequency of ones is M/2. These NIST recommended tests ensure that the occurrence of zeros and ones in a given sequence is uniform, which will be discussed later in this chapter. The cryptanalysts checks the behavior of these random number generators and implant multiple statistical tests to check whether the key is traceable in polynomial time. The cryptanalysts are always in search of breaking the cryptographic security even though it has been proven to be NP-Hard. The emphasis of this work is security issues of SNOW 2.0 algorithm. Their designers have claimed that its security is unbreakable [9] [10]. This work shows that there are some weaknesses in the design of SNOW 2.0 algorithm that let the clever attacks to trace the key generated by its PRNGs [5] [40] [69]. This chapter focuses on the security issues in the design of stream ciphers that may let the cryptanalyst to trace a sub part of key stream, which may result in compromising the entire key.

3.1 The key space

A key space in a cryptographic system is the set of keys available for encryption or decryption process. The key space contains all possible keys in a given range. It needs to be larger enough that is by removing all possible weak keys from a given key space the rest of the strong keys are untraceable using various cryptographic attacks. In symmetric key cryptographic a single keys is selected from a given key space for the encryption and decryption process.

3.1.1 Issues in Key Space

SNOW 2.0 is available in with a key size of 128-bits and 256-bits, which produce 2^{128} and2^{256} possible combination of keys. A key space with 128-bit long includes keys ranging from 000...00 to 111...11, where each key is equally probable. The probability of each key is $1/2^{128}$ and by running a fair PRNG may result in producing any possible combination of zeros and ones in the given range. The focus of our work is whether every key randomly generated is secure or not. The answer is certainly not, because a randomly generated key ranging in the early key space is vulnerable to brute-force and other algebraic attacks. Similarly the key stream contains unbalanced sequence of zeros and ones, which may lead to Guess-and-Determine attacks and related key attacks. A key lies in the extreme end of the key space has a long run of ones with an unbalanced sequence of zeros and once also let a cryptanalyst to apply successful attacks and recover the plaintext.

The randomly generated key may also contain keys with the balanced zero's and one's bits bit the patterns are traceable by its characteristics. For example a key 01010101...........010101, 0011001100...........110011 has balanced zeros and ones but their occurrence is harmonious i.e. with alternate single zeros and ones and alternate double zeros and ones. A randomly generated key with such characteristics cannot be used as a secret key. By removing all such patterns and weak keys the key space again required to have enough key sets those could not be determined in polynomial time otherwise the ciphertext produced with such keys may lead to serious vulnerability.

A randomly generated key with a long run of zeros 00000...1101 or long run of one's 11111...010 cannot be used as a secret key. The repeated bit patterns may cause a ciphering algorithm to produce the same ciphertext for a given plaintext. Guess-and-Determine attacks, correlation attacks, related attacks, distinguishing attack and frequency analysis may let the secret information vulnerable to cryptanalyst to deduce the entire key from a given ciphertext [**8**] [**31**] [**39**] [**67**]. Randomly generated bit stream can be used as a secret key if it has the randomization properties in their bit sequences and does not have the aforementioned issues. A wrong selection of a secret key may let serious cryptographic attacks.

Guess-and-Determine attack discussed in **Chapter 2** is applied over SNOW 2.0 and the guessed keys are stored in a separate file. It is computationally impossible to store all guessed keys in a data file because of the limited storage and resources. Millions of keys guessed and the determinations over the given guesses are not possible to represent as a whole. In our work we base of results and experiments of the selected samples and compare the keys generated through Guess-and-Determine attack with the keys produced through SNOW 2.0. It is necessary to find out the maximum number of traceable bits in a given key to verify the key strength. A secure key contains minimum number of traceable bit patterns because a keys compromised over a specific threshold cannot be trusted. Moreover if the keys generated through Guess-and-Determine attack are used for decrypting a ciphertext the bit patterns similar to the original key will result the same plaintext which may result in understanding the context of the secure message and the cryptanalyst reduce the rest of the undiscovered key.

3.1.2 Key Comparison Algorithm

Key comparison algorithm plays an important role in determining the traceable bit patterns in a given secret key. This algorithm compares the original key produced by Pseudorandom Number Generator (PRNG) of SNOW 2.0 with key set of keys produced through Guess-and-Determine attack. The results of the key comparison algorithm are critically analyzed in this work to determine that the key generated using SNOW 2.0 insecure to a given extent and the risk of collective vulnerability.

The key stream produced by the Guess-and-Determine attack is kept in a separate file to evaluate each individual key against the key generated by the SNOW 2.0 key stream Generator. An algorithm is designed to compare key stream produced through SNOW 2.0 with key keys generated through Guess-and-Determine attack. The key size is restricted due to computational technicalities. A random key *K* is *n* bits long is divided into sub keys k_1, k_2, k_3, ..., .k_i, then each sub key has the random characteristics. It is necessary to understand some basic concepts before describing the comparison algorithm [**8**].

i. Each key stream comprises of 8 alphanumeric.

ii. Key stream generated by Original SNOW 2.0, and attack have been stored in separate files naming *attack_key_file.*

iii. Key stream generated by Original SNOW 2.0 is stored in a file named *original_key_file.*

iv. The first key stream of the *original_key_file* is compared with all attacking key stream in the *attack_key_file,* the next value available in the key stream file is taken and the process continues till the last value in file.

```
Algorithm 3.1: Algorithm Comparing the SNOW 2.0 Generated key with Keys in attack_key_file
Input: Initial Vector //Any value randomly entered by cryptographer ,
         Set IV ← read input through keyboard
        original_key_file=SNOW2(IV) File containing n keys generated by SNOW 2.0
        key[ ] = Key selected from original_key_file Containing keys generated by
                   SNOW2.0
        attack_key[i, j] = Current key in attack_key_file containing n keys responding as 2D array
 Output: Output file containing 0 and 1 // 1 in case of true and 0 in case of false comparison
             Key_comparison
             {
                 vcount = 0
                 j = 0
                 do{    hcount = 0
                        for i = 0 to n
                        {    if (key[i] = = attack_key[i, j])
                             {  put 1 to file
                                  hcount++
                                 vcount++
                             }
                         else{
                              put 0 to file
                             }
                     }
             j++ }
             while (!eof)
             }
```

Algorithm 3.1: Algorithm Comparing the SNOW 2.0 Generated key with Keys in *attack_key_file*

This algorithm compares the original key with each and every key stream of generated by Guess –and-Determine Pseudorandom Number Generator (GD-PRNG) stores the resultant keys in an *attack_key_file* to determines that which of the elements in a key stream are traceable. The algorithm works as follows:

Step I

A series of key streams produced by SNOW 2.0 Pseudorandom Number Generator (PRNG) are stored in the file, called "*original_key_file*". The algorithm loads the first key from this file and store it in array 1, this algorithm stores this key in an array variable key[ ]. Each key in "*original_key_file*" is 64 bit long making 8 ASCII characters as shown in **Figure 3.1.** (Note: the key size of SNOW 2.0 stream cipher is 128 bit and 256 bit long, in this work we take a sample of 64 bits that is 8 characters at the time of generating *original_key_file, where each character is treated as a pattern of 8 bits.*)

Original File
ABCDEFHT
EFDHVFGT
JKLMASDE

Array 1

A	B	C	D	E	F	H	T

Figure 3.1: Keys Selected from *the original_key_file* containing n keys produced by SNOW 2.0 PRNG

Step II

A series of key streams produced by Guess-and-Determine Pseudorandom Number Generator (GD-PRNG) is stored in the file, called "*attack_key_file*". The algorithm loads the keys from *attack_key_file* and store it in array 2, the algorithm store in in array variable attack_key[i, j]. Each key in "attack key file" is 64 bit long making 8 ASCII characters as shown in **Figure 3.2.**

Attack File
ZYXDLRGT
KGJTEADW
BNXCFSRT

Array 2

Z	Y	X	D	L	R	G	T

Figure 3.2: Keys Selected from *attack_key_file* containing n keys generated by GD-RNG

Step III

The original key stream key [ ] are compared with the key values of the attack_key[i, j].

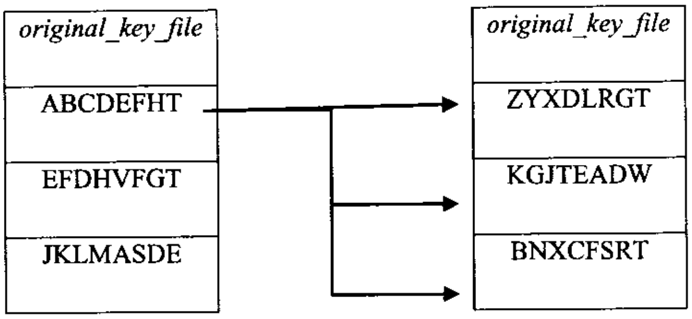


Figure 3.3: Comparison between *original_key_file* and *attack_key_file*

The key values at the respective index in one array are compared with the key values of another array one by one and produce *0* as output in case of mismatch and *1* in case of success as shown in **Table 3.1**. The algorithm stores these values in a separate file that would be used for further analysis.

Table 3.1: Comparison Between *original_key_file* and *attack_key_file*

Index	Value at index of array 1	Matching	Value at index of array 2	Result
		(Equal/not equal)		
0	A	!=	Z	0
1	B	!=	Y	0
2	C	!=	X	0
3	D	= =	D	1
4	E	!=	L	0
5	F	!=	R	0
6	H	!=	G	0
7	T	= =	T	1

The data collected in the experimental process is passed through several statistical tests. Various graphs are drawn for the results produced to find the correlation among the key stream generated by the original Pseudorandom Number Generator (PRNG) and Guess-and-Determine Pseudorandom Number Generator (GD-PRNG), is discussed in **Chapter 5** of this work.

3.2 Effective Attacks on Stream Ciphers

It is important to understand that the purpose of all attacks is to discover a key that is used in the encryption and decryption process. There are a number of methods by which we can attack a secure system to break its security. The designer must have the knowledge of all the known attacks those may possibly be implanted over the designed security system. The effective attacks on stream ciphers are listed below [39]:

i. Exhaustive Search Attack [112]
ii. Algebraic Attack [113]
iii. Correlation Attack [114]
iv. Fault Attack [115]
v. Distinguishing Attack [38] [40] [68] [82]
vi. Chosen-IV Attack [78]
vii. Slide Attack [116] [117] [118] [119]
viii. Cube Attack [120] [121]
ix. Time-Memory Trade-off Attack [122] [123]

3.3 Pseudorandom Numbers and Sequences

In stream cipher algorithms the random number generation is one of the most important primitives, and the entire security is based on this key that is generated by these pseudorandom number generators. For example, if the sender and receiver are at different locations and they want to share some information by transformation then the adversary may get secret information about the key. So the measures are taken that it should be unpredictable [41] [42] [124]. The random numbers are bit sequences involved in generating the random numbers, which may present challenging issues [43] [125]. Whenever a new algorithm is designed, the designers claim that it is secure both in case of stream ciphers and block ciphers, and they claim the efficiency of fast running stream ciphers.

The efficiency of the ciphering scheme can be compromised but the security cannot be compromised. The purpose of the stream ciphers is to generate fast speed encryption and decryption process, which has been achieved by simply XORing the key with the plaintext; key generated by the Pseudorandom Number Generator (PRNG) requires more attention. From the above discussion it becomes clear that in case of generating weak keys the whole stream cipher is breakable. Repeating patterns in key stream may result a series of security issues due to which these keys cannot be used for secure communication. Removing these weak patterns improves the key strength which ultimately improves the security of the parent algorithm and the cryptanalyst may not be able to determine the key in the polynomial time.

Using Linear Feedback Shift Registers (LFSR), Non Linear Feedback Shift Registers NLFSR based stream ciphers, no one can guarantee that a randomly generated key is secure, which is the major drawback in the logic based key generation. To overcome this problem the stream ciphers require special attentions on the design of Pseudorandom Number Generators (PRNGs). This dissertation proves that by removing traceable bit patterns from a key stream enables it to satisfy most of the NIST recommended statistical tests and resist most of the known security attacks. In any of the cryptographic application the following steps must be performed:

i. If there is a finite set of *n* elements, i.e. {1, 2, 3... n}, then select an element at random, such that selection of each element is equally likely.

ii. Randomly select a sequence from the set of all sequences (strings) of length *m* over some finite alphabet *A* of *n* symbols.

iii. Generate a sequence having symbols of length move over a set on *n* symbols at random.

The exact meaning of the term generate at random or select at random is not clear, but calling a number random without a context makes little sense **[18] [41] [125]**.

Key Stream: *K* is called a key stream, such that *K* is key space of set of encryption transformations, where *K* is the sequence of symbols $\{e_1,e_2,e_3,.....e_i\}$

Encryption Scheme: Let *A* be an alphabet of *q* symbols and let E_e be a simple substitution cipher with block length 1 where e ∈ K.

Let $\{m_1\ m_2\ m_3......\}$ be a plaintext string and

Let $\{e_1\ e_2\ e_3\ ......\}$ be a key stream from *K*.

A *stream cipher* takes the plaintext string and produces a ciphertext string $\{c_1\ c_2\ c_3.....\}$, where

$c_i = E(e_i, m_i)$. , then

$D(e_i, c_i) = m_i$ decrypts the ciphertext string.

Encryption Scheme Security: For any of the encryption scheme to be secure, the key space must be larger enough to preclude exhaustive search, it is a necessary but not a sufficient condition.

3.4 Random Number Verification Tests

National Institute of Standards and Technology (NIST) recommends 16 statistical tests to ensure whether a key generated is random or not. None of the stream ciphers designed so far satisfy all 16 statistical tests. The stream cipher that satisfies maximum number of tests is used to be recommended for security purposes. These tests focus on a variety of different types of non-randomness that could exist in a sequence. These tests ensure that whether a key generated by a PRNG is secure if it is used as secret key.

In a given key space all keys are not treated as random even if these keys are generated randomly. Every randomly generated key is not secure and it cannot be used as a secret key for message encryption. National Institute of Standards and Technology (NIST) recommended statistical suite defines the acceptance regions in terms of *Standard Normal* and the *chi-square* χ^2*as r*eference distributions. These distributions define the regions for acceptance and non-acceptance for a given key space. If a bit sequence under test is in fact non-random, the calculated test statistic will fall in extreme regions of the reference distribution. The test results for a given bit stream will fall outside the acceptance region. for example a key sequence fails to satisfy the Monobit test will lie in the extreme region or away from the acceptance region, which means that zeros and ones non proportional.

The standard normal distribution (i.e., the bell-shaped curve) is used to compare the value of the test statistic obtained from the PRNG with the expected value of the statistic under the assumption of randomness. A cryptanalyst expects that a PRNG generates a random key satisfying all NIST recommended tests, and a generated key is tests under the assumption that it is random and satisfies all randomness tests in NIST statistical suite. The test results classify the randomly generated key locate its location under the bell-shaped curve of the Normal distribution. The test statistic for the standard normal distribution is of the form $z = (x - \mu)/\sigma^2$, where x is the sample test statistic value, and μ and σ^2 are the expected value and the variance of the test statistic. The χ^2 distribution (i.e., a left skewed curve) is used to compare the goodness-of-fit of the observed frequencies of a sample measure to the corresponding expected frequencies of the hypothesized distribution. A 128-bit or 256-bit randomly generated bit sequence is tested for the patterns frequency within a block of 8 bits. A key selected require having unique untraceable patterns. A Guess-and-Determine attack over the random key determines the frequency of the original key with the guessed keys. A single pattern is more frequent in a given key space but two consecutive patterns similar to the original key is comparatively less frequent and so forth. Plotting a χ^2 graph is left skewed curve and classify the key as strong with minimal frequent patterns. Rest of NIST statistical tests are discussed in detail in **Chapter 5**.

3.5 Summary

The stream cipher algorithms works in two major steps i.e. key generation phase and data encryption phase. In the first step a random key is generated and forwarded to use for data encryption. The data encryption in stream ciphers is a simple XOR operation of the random key and the available data. If the key generated is not strong enough then the plaintext can be recovered using clever attacks discussed in **Chapter 2**. The whole burden of security lies upon the designer to generate a random key. It should not contain such a bit patterns those may let the cryptanalyst to decipher the encrypted text without having key. We consider stream cipher SNOW 2.0 and generate a random sequence of bits, and pass through Random Key Generator to check whether the key is stronger enough to be used as a secure key for this purpose we check the key produced whether they have the pattern those may be traced by the cryptanalyst. Multiple experiments have been done to check whether the key is strong, which shows that the key is stronger enough if we pass it through limited series of attacks. The same key is attacked for a huge number of times as the cryptanalyst does during key analysis, it is observed that the probability of some bit patterns does not occur under the normal curve and some patterns occurs more frequently. The cryptanalyst while applying the frequency test guesses these patterns which occur more frequently to determine the rest of the key. The cryptographic attacks discussed in **Section 3.1.2** and **Section 3.2** is efficiently used to break the security of stream ciphers and it is shown that the logic based stream ciphers are no more secure. The security algorithm designers therefore have to change the security models those may have strong design structure and also they need to introduce new techniques those may improve the security of stream ciphering models. The stream ciphers based on genetic algorithms discussed in **Chapter 2** have design deficiency and the cryptanalytic attacks compromise their security. The key stream generator of SNOW 2.0 is cryptographically proven to be secure but the practical attacks on SNOW 2.0 has broken its security. These cryptographic attacks not only focus on the mathematical strength but also on the resultant ciphertext produced by these stream ciphers, and there are dozens of examples in which the cryptographically proven NP-Hard problems have been compromised. This work therefore discourages the conventional stream cipher design and present Intelligent Cryptographic Model (ICM) for stream ciphers.

CHAPTER 4
INTELLIGENT ALGORITHM DESIGN

Cryptographers are always in search of designing secure systems for keeping their information secret. The sender and receiver require confidentiality during communication. They want to keep their conversation secret. Different stream cipher algorithms can be used for this purpose. These algorithms are best suited for short term communication having limited memory and processing power. The sender and receiver require secret key to encrypt and decrypt the information. The Random Number Generators (RNGs) and Pseudorandom Number Generators (PRNGs) generate random numbers. Their output is dependent on the Initial Vector (IV), which can be feed by different means. The end user relies upon the bit sequence that is produced by RNGs or PRNGs. If the key produced has statistical weaknesses then that may compromise the security of entire system.

Artificial Intelligence enables a system to take decisions intelligently. It plays vital role in cryptography and is used in many areas for providing better information security. Neural cryptography is an emerging branch of cryptography in which artificial neural network is used for designing secure intelligent algorithms. Weak keys are generated because of the loopholes in key generation algorithms those may lead a key or a subpart of a key vulnerable to cryptanalyst [22] [23] [126] [127].

4.1 Biological Neurons

The study of the Artificial Neural Networks (ANN) is based on the study of biological learning systems where thousands of other neurons are interconnected to make a very complex web of these structures. The ANN are made up of simple program units where each unit takes an input from a number of real value inputs, where these inputs may be through keyboard or the output of some other program units and produce a single real valued output. This output produced by the neuron may be the input to another neuron. These neurons need not have the similar characteristics, and almost each neuron may function differently from another neuron, which performs some specific task. To clearly understand the analogy of ANN, we consider some facts about the neurobiology.

There are approximately 10^{11} neurons in the human brain those work together to make an interconnected network. Each neuron is connected to approximately 10^3 to 10^4 other neurons. The processing speed of a neuron in human body is too slow than that of the processing speed of a computer system. However, the speed is not the only matter; rather there are some very important features in the natural structure of the neurons. A human can make surprisingly complex decisions quickly, for example, in pattern recognition the basic task is to identify the structures and images in motion or static. The human mind can process very fast whereas, the computer systems have to pass through very complex operations to recognize a pattern and the end result may also an inappropriate decision. This was the vision of the early 80's, but due to advancements in artificial intelligence has now empowered computer systems to perform some of the operations more precisely and better than the human beings **[128]** [129] [130] [131].

While ANNs are loosely motivated by biological neural systems, there are many complexities to biological neural systems that are not modeled by ANNs, and many features of ANNs are known to be inconsistent with biological systems. Individual unit of ANNs produces output a single constant value, whereas biological neurons produce output a complex time series of spikes. This work uses Artificial Neural Networks (ANN) to avoid the traceable bit stream patterns in the logic based stream ciphers is due to the following two main reasons:

i. The resemblance of biological learning process of the ANN to human beings gives more understanding power and knowledge extraction to ANN.
ii. The enhanced decision making power when an attack is established to find the pattern of the bit stream in the original key of the stream cipher algorithm.

4.1.1 Biological Neuron Structure

Biological neurons are the foundation for ANN. ANN's functional model is designed to resemble the biological neurons in many ways. "Neurons are the basic signaling units of the nervous system" and "each neuron is a discrete cell whose several processes arise from its cell body" [132]. The biological neuron has the following four regions as shown in **Figure 4.1** [132]:

i. Cell body also termed as soma
ii. Dendrites
iii. Axon
iv. Presynaptic terminals

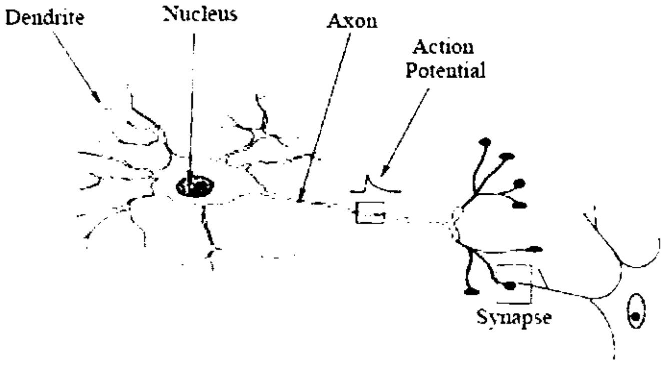


Figure 4.1: General Structure of a Biological Neuron

All four regions of a neuron has a specific task to perform and work as a unit, where it accepts multiple inputs and produce output a single result. The functionality of the biological neurons is discussed briefly in the following:

i. The cell body or soma is the main part of the neuron cell, which contains protein synthesis and nucleus, which also works as storage and decision making body locally, such as reflex actions which are initiated as a result of these local decision makers inside the neuron.
ii. The dendrites which are the offshoots of the soma have branches in a treelike structure, which are used to receive signals from other neurons or the organs which senses the information from the outside.
iii. The axon that grows outside the cell body also termed as the hillock and conduct electric signals that grows from the cell body and ends at the presynaptic terminals. Only one axon grows from the cell which transfer signals to other neurons or brain. The action potential that carries the electric signals is identical to each other so the brains can easily judge that which type of signal has been received based on the path that the signal took. The brain analyzes the signal patterns and issues orders where appropriate.

iv. The synapses are again the threads at the end of the axons which are connected to the dendrite of the other neurons where the information is passed from the synapse to the dendrites of the following neuron.

4.1.2 ANN Based Stream Cipher

Artificial Neural networks works similar to biological neural networks. The dendrites takes input from the outer world or other neurons and the cell body containing some local information and has the decision making power required for the reflex action. On the basis of the decision taken by the cell body the axon takes the information and further provide it to the next neurons as shown in **Figure 4.2**. It is important to understand the working process of ANN as it has been made a foundation for the Intelligent Stream cipher algorithm.

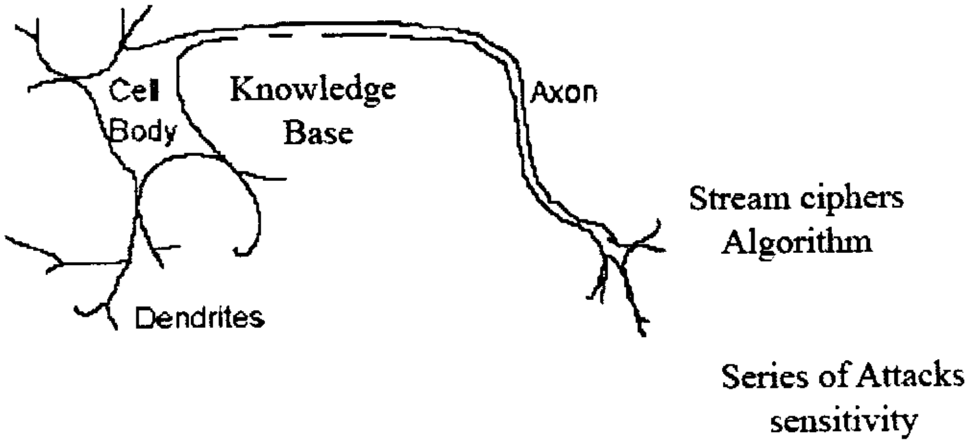


Figure 4.2: Biological Neuron Model for Stream Cipher

The cell body preforms two operations:

i. It is trained and takes decision based on the trained data. It takes input from other neuron and decides whether the given key has some consecutive bit patterns like consecutive 0's or 1's.

ii. It takes a decision to disrupt this sequence and coverts some of 0's to 1's or otherwise.

The output is given to axon that is further connected to other neurons and they also add their functionality. The key produced during this process is passed through multiple cryptanalytic tests and NIST statistical tests are also applied too. If the results produced come under the acceptance region then the key is declared as secure otherwise the back propagation function is invoked that changes the neuron weights and the process is continued to produce a secure key.

4.2 Artificial Neural Network (ANN) Architecture

The word network in ANN refers to the interconnections between neurons in different layers of each system. An example system is shown in **Figure 4.3** has three layers. First layer has input neurons which send data via synapses to the second layer of neurons, and then via more synapses to the third layer of output neurons. More complex systems will have more layers of neurons with some having a number of layers of input and output neurons. The synapses store parameters called "weights" manipulate the data in the calculations.

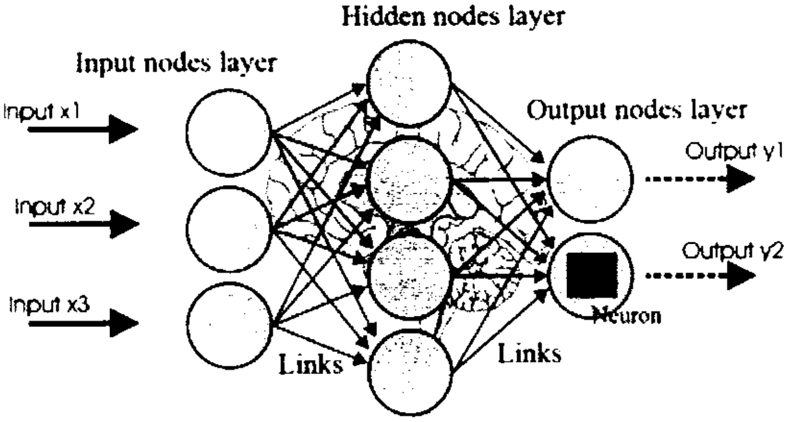


Figure 4.3: Three Layered Structure of Artificial Neural Network

The Artificial Neural Networks (ANN) working is a complex process. The hidden Layer works as a brain that receives inputs from multiple neurons. The neurons in the hidden layers are also interconnected to each other and one neuron in the hidden layer gives its output to another neuron and the neurons in the hidden layer produce a mesh topology. The purpose of each neuron in the hidden layer is predefined and they perform that specific task. Each neuron operates differently on the inputs given to it and adds its expertise to the results based on the training data and decides accordingly. The data is refined and processed at each neuron and the final output is handed over to the neurons in the output layer. If the final output is not lying in the acceptance region, the back propagation invokes and the neuron weights are readjusted.

Mathematically, a neuron's network function $f(x)$ is defined as a composition of other functions $g_i(x)$, which can further be defined as a composition of other functions. This can be conveniently represented as a network structure, with arrows depicting the dependencies between variables. A widely used type of composition is the nonlinear weighted sum, where $f(x)=K\left(\sum_i w_i g_i(x)\right)$, where K (commonly referred to as the activation function) is some predefined function, such as the hyperbolic tangent. It is convenient to refer to a collection of functions g_i as simply a vector $g=(g_1,g_2,\ldots,g_n)$. An ANN is typically defined by three types of parameters:

i. The interconnection pattern between the different layers of neurons
ii. The learning process for updating the weights of the interconnections
iii. The activation function that converts a neuron's weighted input to its output activation.

4.3 Multilayer Perceptron (MLP)

In multilayer perceptron (MLP) structure, neurons are grouped into layers. The first and last layers are called input and output layers respectively, because they represent inputs and outputs of the overall network. The remaining layers are called hidden layers. Typically an MLP neural network consists of input layers, one or more hidden layers and an output layer.

Suppose total number of layers in MPL is L. Then the first layer is input layer, and L^{th} layer is output layer and layers number 2 to L-1 are the hidden layers. Let the number of neurons in the l^{th} layer be N_l, l= 1, 2, 3, 4 .... L, is shown in **Figure 4.4.**

Let $w^l_{i,j}$ represent the weight of the link between j^{th} neuron of l-1^{th} layer and the i^{th} neuron of l^{th} layer, where $1 \leq j \leq N_{l-1}$, and $1 \leq i \leq N_l$. Let x_i represents the i^{th} external input of the MLP, and Z_i be the output of the i^{th} neuron of the l^{th} layer. The extra weight parameter w_{i0}, for each neuron, represents the bias of i^{th} neuron of the l^{th} layer. As such w of MLP includes $w_{i,j}$, where j = 1, 2, 3,... N_{l-1}, i = 1, 2, 3, ...N_l, l= 2, 3, ... L, that is

$$W = [w^2_{10}\, w^2_{11}\, w^2_{12}.........w^l_{NlNl-1}] \qquad (4.1)$$

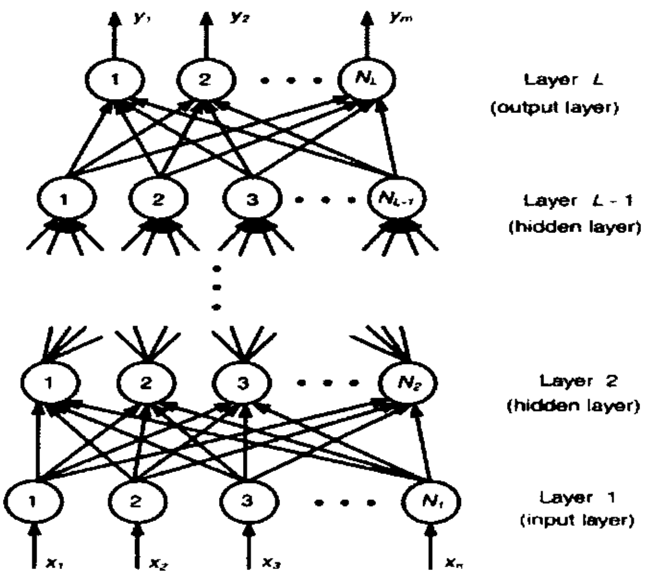


Figure 4.4: A Multilayer Perceptron (MLP) Structure.

where *W* is the overall weight of the connected perceptron that affect the final outcome of the neural network. The output layer that results in generating the final key is controlled by adjustment and readjustment of neurons weights available in the hidden layers. The process continues till the required output is generated fulfilling almost all randomization tests to a threshold value.

4.4 Activation Function of the MLP

Usually, in neural networks the neuron's output is between 0 and 1 or -1 and 1. The activation function is denoted by Φ(.) is of three types:

i. If the summed input is less than the threshold value, then threshold function takes on a value of 0, and if the summed input is greater than or equal to the threshold value then it take the value as 1 as shown in **Figure 4.5.**

$$\Phi(\mathrm{v}) = \begin{cases} 1 & if\ v \geq 0 \\ 0 & if\ v < 0 \end{cases} \tag{4.2}$$

This kind of function is often used in single layer network.

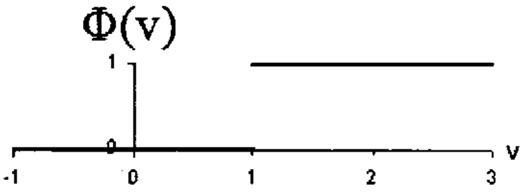


Figure 4.5: Binary Step Function

ii. The piecewise-linear function, taken on the values of 0 or 1 and also may be taken on the amplification factor in a certain region of linear operation.

$$\Phi(\mathrm{v}) = \begin{cases} 1 & v \geq \frac{1}{2} \\ v & -\frac{1}{2} > v > \frac{1}{2} \\ 0 & v \leq -\frac{1}{2} \end{cases} \qquad (4.3)$$

iii. The sigmoid function ranged between 0 and 1, but can also be used for the range -1 and 1, the example of sigmoid function is hyperbolic function, which is applicable while drawing the graph for the region of similar patterns.

$$\Phi(\mathrm{v}) = \tanh(\mathrm{v}/2) = (1 - \exp(-\mathrm{v})) \,/\, (1 + \exp(\mathrm{v})) \qquad (4.4)$$

4.5 Intelligent Cryptographic Model (ICM)

The Intelligent Cryptographic Model (ICM) is a compact solution for removing the cryptographic loopholes from logic based stream ciphers. The key generated by logic based stream ciphers having cryptographic vulnerabilities are addressed in Intelligent Cryptographic Model (ICM) and trained Artificial Neural Network (ANN) regenerates a strong keys having no statistical weaknesses.

4.5.1 ICM Structure

The structure of Intelligent Cryptographic Model (ICM) is shown **Figure 4.6**. When a request is submitted from client side with an initial seed for a key to be generated, this request is forwarded to the inference engine. The inference engine is connected to a number of other components to communicate different information to generate a statistically strong key.

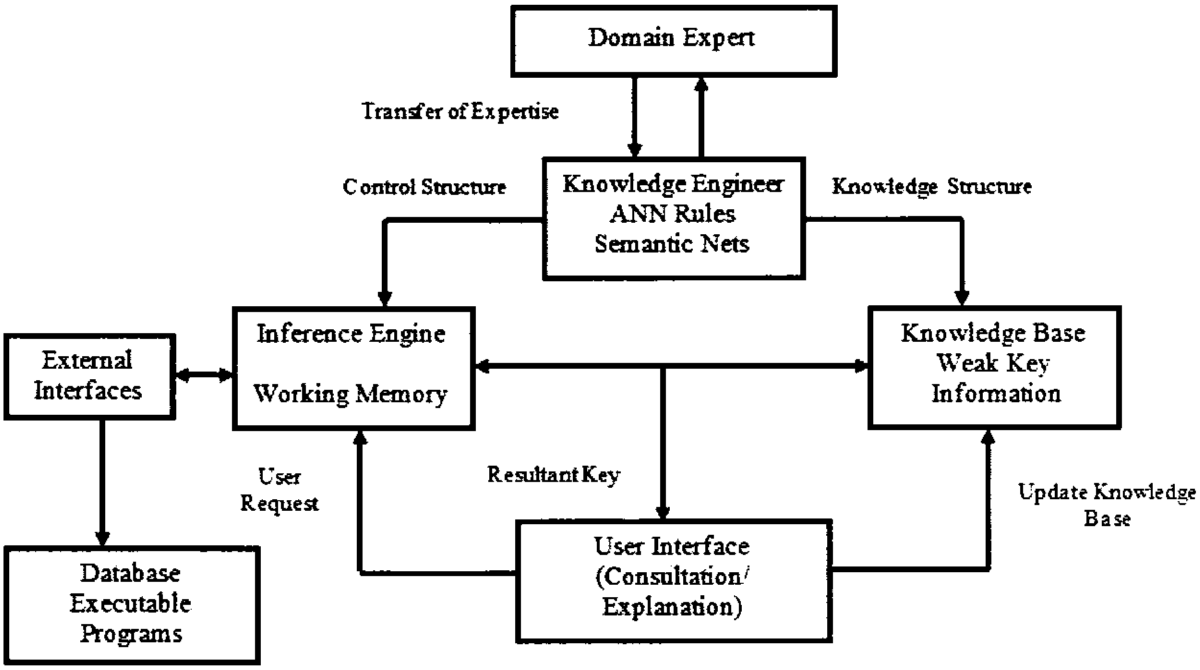


Figure 4.6: General Structure of Intelligent Cryptographic Model (ICM)

Following are main components of ICM:

i. User Interface (Submit Final Result)

ii. Knowledge Base/ Key Information Center

iii. Knowledge Engineer (ANN Rules/ Semantic Nets)

iv. External Interfaces (To get seeds from outside world)

v. Domain Expert (For Strong key Certification and Confirmation)

The Knowledge Base or Key Information Center represents facts about the world. In our model the knowledge base keeps a list of keys having been marked as weak keys and has been proven to breakable. These weak keys have such a bit patterns those have been proved to be traceable. For an efficient and secure systems we need to have enough knowledge base and it must be updatable to add new keys if proven to be traceable.

A knowledge engineer integrates knowledge into computer systems in order to solve complex problems normally requiring a high level of human expertise. The expert system we propose contains the neural networks and semantic rules which applies a series of tests upon the key generated by, the LFSR based stream cipher, in our case SNOW 2.0. These tests include the pattern checking in the key recently produced, and the detailed process is explained in the next sections of this chapter.

External Interfaces in this model are database and program functions that get the key and keep it in a list of secure keys once used. Alternatively it is marked as weak key as per decision of the knowledge engineer after a though processing of the key generated. The key is assigned its category and is stored in the database.

The Domain Expert is a central program that gets information from knowledge base, knowledge engineer and external interfaces i.e. databases and checks with the threshold value that the user considers acceptable. When key is sent to domain expert he then issues a certificate to the user about the cryptographic strength of the key.

4.5.2 ICM Illustration

The proposed model is based on the amalgamation of modern stream cipher and artificial neural network. The model works in a systematic manner and the output of one part of the model is taken as input to another part. In the following we illustrate important points of the Intelligent Cryptographic Model (ICM):

i. **Instance Representation:** The learning algorithm educates the target function is defined over the instances. A vector of predefined features is described, such as past attacks those have been successfully implanted. The tendency towards the success that in case of a successful attacks how much damage may occur. The input values can be any real values, which may be independent of one another or may highly correlate.

ii. **The output of the Target Function:** It is a vector of several real or discrete-valued, real valued or discrete valued attributes. If the output is a real number then it will range between 0 and 1, which in this case corresponds to the confidence in predicting the corresponding steering direction. We can also train a single network to output both the steering command and suggested acceleration, simply by concatenating the vectors that encode these two output predictions.

iii. **Training Errors:** Errors may occur in the training examples. If there is a noisy data with errors in training examples, then it depends on the learning methods that how they respond to these errors. In the other applications of the ANN, it is kept in mind that the training data must not contain errors, but in the ciphering algorithms we permit to larger instance, because here we are interested to maintain a training base and not in the quality of data.

iv. **Training Time:** It may be long in case of an isolated system. To fully train an artificial neural network algorithm requires longer times ranging from a few seconds to many hours and even days and weeks, depending on the factors such as the settings of various learning algorithm parameters, number of training examples considered, and number of weights in the network. Keeping in view that in public area where the outer world has access to the system, longer time is never permissible, and in such a case the point iii, may be applicable for the time being.

v. **Learning Target:** It is required to be evaluated with fast speed. Long time is required to train an ANN. In case of a successful attack the there is a need for an instantaneous and accurate decision. The data transfer among different nodes needs to be very fast so as to update their weights.

vi. **Learned Target Function:** The learned target function need not to be understood by the humans. It is very difficult for the humans to clearly understand the complex nature of the weights learned by the ANN, and is very difficult to interpret for them. The learned rules can easily be communicated to the learning rules, but it is very difficult to communicate it to human beings. Thus learned rules are designed in such a way that if the data store contains some data for what the rules have already been defined, then a human may ignore it, but the processing device cannot ignore it, which may cause to increase the security, i.e. to avoid the similar bit patterns in the key stream, produced by ANN based Intelligent Pseudorandom Number Generator.

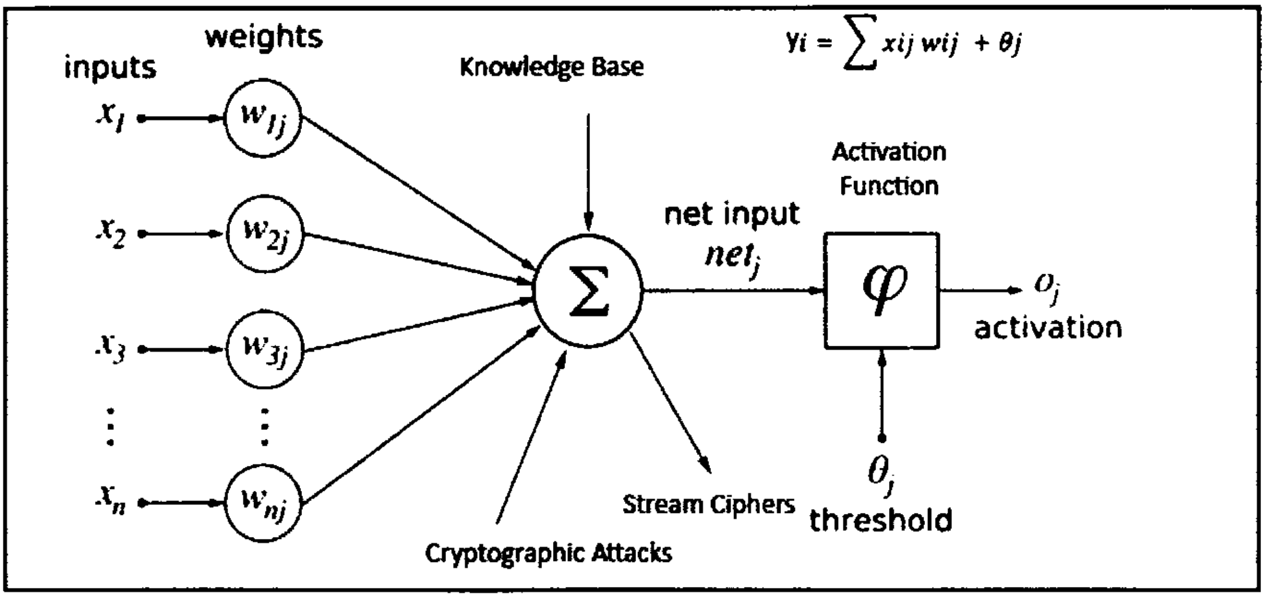


Figure 4.7: Mathematical Representation of Artificial Neural Network

The model shown in **Figure 4.7** accepts different inputs for training the learning algorithm those are kept in the database, which contain the rules for extracting the knowledge from the knowledge base, sensitivity of serious attacks, the stream ciphers and dozens of the weak keys, proved by the analysts.

$$\mathrm{yi} = \sum wij.xij \quad + \ \theta j \tag{4.5}$$

The activation function gets output from the summing junction where the weight w_{ij}, is multiplied with the x_{ij} and are added with θj. The computed output is forwarded as output to the next neuron. Every neuron in the hidden layer transfer its output to the next neuron in the hidden layer, except the neurons in the first layer which gets input from the input layer and the last layer forward its output to the output layer.

4.6 The Perceptron Learning Algorithm

The patterns generated in this step need further evaluation to ensure that the key is unbreakable. We apply cryptanalytic tests to ensure that the ultimate security is gained while using this system. During this whole process the neurons also gets training and store the results generated by these tests regarding that specific key.

```
Algorithm 4.1: Perceptron Learning Algorithm for Intelligent SNOW 2.0
Input: Threshold value t, weight of Neuron w, Initial Vector IV, Result File RF
Output: Learned Perceptron
Initialize:
        N(w) ← current weight of the Neuron
        t ← assign random threshold value
        w ← assign random weight value
Compute new weight of the Neuron
$N ( w_i ( t - 1) ) = w_i ( t ) + \eta ( d - y ) x_i$
$e = w_i ( t + 1 ) - w_i ( t )$
/*      where
           i ← represent neuron no.
           d ← the expected output of the neuron
           y ←the actual output generated by the neuron and
           $\eta$ ← learning rate or step size ranging between ($0 < \eta < 1$)
           e ←error difference of the neurons
 */
$N ( w_i ) = N ( w_i )$ XOR IV
do
        $N ( w_i( t+1 ) ) = w_i( t ) + \eta ( d - y ) x_i$
        $e = w_i( t + 1 ) - w_i( t )$
while ( e > t )
```

Algorithm 4.1: Perceptron Learning Algorithm for Intelligent SNOW 2.0

This algorithm readjusts the weights *w* of the perceptron. Initially the weights *w* are randomly assigned to each perceptron. These weights are used for the operations with the input values. In the training session these weights are adjusted and readjusted if an input is wrongly classified. The algorithm takes N(W), *t* and weight *w* initially as input and compute the weight of the neuron for a threshold value t-1 in **Line 4**. **Line 5** computes the error by finding the difference between the current and previous neuron weights. The learning rate value ŋ is carefully selected because for a smaller value of ŋ the algorithm converges slowly and takes more time to produce optimal solution and for a larger value of ŋ, the algorithm will oscillate or diverge.

The neuron weight is computed by XORing $N(w_i)$ with the Initial Vector (IV). The neuron weights are computed in **Line 8** and the error difference is computed in **Line 9** by subtracting the current weight $W_i(t)$ from the next neuron state $W_i(t+1)$. The process continues till the error is reduced to a threshold value. The output of the algorithm is a learned perceptron.

4.6.1 Back Propagation

It is a common method of training the ANN in which an input is given to the network and is checked for the desired output. It is a supervisory learning method where the ANN is trained in order to get a desired output. If the desired output is not obtained then the error is propagated and the neurons weights are readjusted. Back propagation learning algorithm works in two phases:

i. Propagation
ii. Weight Update

When an attack becomes successful in finding the similar bit patterns in the key stream, it propagates that the pattern generated does not meet the expected output requirements and the weights are updated accordingly **[133] [134] [135] [136]**.

The back propagation algorithm trains a given feed-forward multilayer neural network for a given set of input patterns with known classifications. When each entry of the sample set is presented to the network, the network examines its output responses to the sample input pattern. The output response is then compared to the known and desired output and the error value is calculated. Based on the error, the connection weights are adjusted. The set of these sample patterns are repeatedly presented to the network until the error value is minimized **[137]**.

The **Algorithm 4.2** manages the knowledge base and is called B*ack Propagation Learning algorithm* and is illustrated below:

```
Algorithm 4.2: Back Propagation Learning Algorithm for Intelligent SNOW 2.0
Input: Randomly initialize the weights present an input vector pattern to the network
Output: Error Free Learned Perceptron
1   Evaluate the network's outputs by propagating the signals forward
2   Calculate
        $\delta_j = (y_j - d_j)$,
        for all output neurons
    Where $d_j$ is the output of neuron $j$, which is desired and $y_j$ is the output that we receive
    currently, where $y_j$ is computed from the following equation.
        $y_j = g(\sum_i w_{ij}x_i) = (1 + e^{-\sum_i w_{ij}x_i})^{-1}$,
        which is sigmoid activation function
3   For rest of the neurons compute
        $\delta_j = \sum_k w_{jk}g'(x)\delta_k$,
        where $\delta_k$ is the $\delta$j of the succeeding layer, and
        $g'(x) = y_k(1 - y_k)$,
4   Update the weights according to:
        $w_{ij}(t+1) = w_{ij}(t) - \eta y_i y_j(1 - y_j)\delta_j$.
        Where, $\eta$ (neo) is called the learning rate in the above equation.
5   Go to step 2 for a certain number of iterations, or until the error is less than a pre-specified
    value. Repeat step 1 to 4, until the error becomes less than the pre-specified threshold value.
```

Algorithm 4.2: Back Propagation Learning Algorithm for Intelligent SNOW 2.0

The perceptron initially trained may not properly classify the cryptographically weak keys and strong keys. If a cryptographically strong key is classified as weak key may not affect the secure model but wrong classification of a weak key may lead to a very serious attack. The back propagation algorithm retrains the perceptron in such a situation. The algorithm finds δ_j by finding the difference between the current neuron output y_j and the desired output d_j, recomputed compute y_j and d_j in **step 2** and **3**. Update all the neuron weights by computing it in **step 4**. The output of the back propagation algorithm is the error free learned perceptron.

The input key to the neural network is checked whether the bit sequences are random or there is some biasness in it. The desired output is always the patterns that are generated randomly and meet the random sequence requirements. If the desired output is not achieved then the error is propagated and the bits in the given sequence are readjusted. The weights are updated and new weights are assigned to the neurons. The process continues until the desired sequence is generated.

4.6.2 Perceptron Convergence Theorem

"If there is a set of weights that correctly classify the (linearly separable) training patterns, then the learning algorithm will find one such weight set, w^* in a finite number of iterations (Rosenblatt)" [25]

Assumptions:

i. There is a set of weights w^*

ii. A known number of patterns exists to train the perceptron

iii. A unipolar threshold function with resultant output 0 or 1

Proof:

The process continues to train the perceptron and at k^{th} iteration we have

$$\omega_{k+1} = \omega_k + \mu\, e_k x_k , \qquad (4.6)$$

where

$$e_k = d_k - y_k \qquad (4.7)$$

Reduce both the sides with the quantity w^*

$$\omega_{k+1} - w^* = \omega_k - w^* + \mu\, e_k x_k , \qquad (4.8)$$

There is no update if y_k is accurately categorized

Normalize the above equation, by putting that $||e||^2 = 1$, since e = ± 1:

$$||\ \omega_{k+1} - w^*||^2 = ||\ \omega_k - w^*||^2 + \mu^2\, ||x_k||^2 + 2\ \mu\, e_k(\omega_k - \omega^*)^T x_k \qquad (4.9)$$

For each of the unclassified vector x_k, we can show that:

$$e_k\ \omega^{*T} x_k = |\ \omega^{*T} x_k| \geq 0 \qquad (4.10)$$

$$e_k\ \omega_k^T x_k = |\ \omega_k^T x_k| = 0 \qquad (4.11)$$

Rearranging **Equation 4.9**

$$|| \omega_{k+1} - w^*||^2 = || \omega_k - w^*||^2 + \mu^2 ||x_k||^2 - 2 \mu (| \omega^{*T} x_k | + | \omega_k^T x_k |) \tag{4.12}$$

If μ is sufficiently small

$$\mu^2 ||x_k||^2 \rightarrow 0 \tag{4.13}$$

putting the value from Eq. 4.15 in Eq. 4.14 we get

$$||w_{k+1} - w^*||^2 = ||w_k - w^*||^2 - 2\mu(|w^{*T} x_k| + |w_k^T x_k|) \tag{4.14}$$

Since $\mu \geq 0$ and

$$|w^{*T} x_k| + |w_k^T x_k| \geq 0, \tag{4.15}$$

Where $||w_{k+1} - w^*||$, must decrease with each of the iteration.

However, $||.||$ cannot go negative, so it must converge in a finite number of steps.

Note that this may not necessarily converge to 0 (i.e. $w = w^*$)

Rewriting **Equation 4.12** in terms of the weight error,

$$e = w - w^*, \tag{4.16}$$

$$||e_{k+1}||^2 = ||e_k||^2 + \mu^2 ||x_k||^2 - 2\mu(|w^{*T} x_k| + |w_k^T x_k|) \tag{4.17}$$

Minimizing e_k with respect to μ gives:

$$\mu_{opt} = \frac{{}^{*T} x_k| + |w_k^T x_k|}{||x_k||^2} = \frac{|(w^* - w_k)^T x_k|}{||x_k||^2} \tag{4.18}$$

In practice, it is impossible to evaluate this, since it contains w^*, (the optimal weight vector) μ_{opt} can be interpreted as the optimal distance to move w_k along the direction of x_k . When the perceptron becomes learned by getting from the relative attacks on SNOW 2.0, the perceptron converges and the resultant vector or the output will become closer to the threshold values selected by the cryptanalyst, and if the threshold value is increased to a significant level, the **Equation 4.12** will diverge, and the analyst will become away from its destination value. The algorithm will be successful but the accurate result cannot be achieved. The perceptron learning algorithm becomes learned by providing the prior knowledge to it and is stored in the centralized knowledge base (KB), which increases time to time by adding more information about the weak keys where the patterns have been found to be traceable. The errors produced during the iteration due to the incorrect classification if a weak is classified as strong and vice versa. It is a greedy algorithm and stops iterations as it finds the first feasible outcome. The perceptron will diverge in case of faulty training data therefore it is required to check the error difference in each step.

4.6.3 Universal Approximation Theorem

The linear approximation theorem is applied on SNOW 2.0, for the approximation of the Addition Modulo 2^n, so as to find the approximate probability of any of the key stream with some specific bit patterns. The universal approximation theorem states that: "The simplest form of multilayer perceptron, a single hidden layer, in a feed forward network, containing a finite number of neurons is a universal approximator among continuous functions on the reduced form of R_n, where the activation functions are molded" **[138]**. The mathematical form of the universal approximation theorem is as under:

Consider a bounded, non-constant and monotonically increasing continuous function $\varphi(\cdot)$, Let I_m denote the m-dimensional unit hypercube $[0,1]^m$. The space of continuous functions on I_m is denoted by $C(I_m)$. Then, given any function $f \in C(I_m)$ and $\epsilon > 0$, there exist an integer N and real constants a_i, $b_i \in \mathbf{R}$, $w_i \in \mathbf{R}^m$, where $i = 1, ..., N$ such that we may define:

$$F(x) = \sum_{i=1}^{N} \alpha_i \varphi\left(w_i^T x + b_i\right) \tag{4.19}$$

as an approximate realization of the function f where f is independent of φ; that is,

$$|F(x) - f(x)| < \varepsilon \tag{4.20}$$

for all $x \in I_m$. In other words, functions of the form $F(x)$ are dense in $C(I_m)$.

Let's find out the linear approximation on Addition Modulo 2^n in SNOW 2.0,

i. In the linear approximation of the Finite State Machine (FSM) of SNOW 2.0, there are two strongly dependent approximations: When one approximates over two subsequent additions modulo 2^{32}, that is the output from the first addition is an input to the second addition.

ii. The fact that the value in register R_1 is both an input to the modular addition and an input to the S-box ensembles S.

The interesting feature of the linear approximation with one-bit mask is that in two input case, the linear approximation degrades when moving towards the most significant bits. For addition with three inputs the bias values are almost the same in all positions of the key stream. Also the linear approximation over modular addition with three inputs is more flexible and gives better bias values when no all input masks are the same. The same holds in general for very sparse masks and therefore exhaustive search over all masks for linear approximation over addition modulo 2^n is the most appropriate for finding the patterns in some specific key.

4.6.4 An Illustrative Example

A random sequence of bits is produced using SNOW 2.0 128-bit Pseudorandom Number Generator (PRNG).

Random key: 0011010011000100*00001100*00011101*00001100*000001110110010

11000010010111110*00001100*100100010000111010010110101010

101000100100110100

Key size: 128-bits

Visible Characteristics: Random

Let's critically analyze the given sequence; the key is divided 16 chunks, where each chunk contains 8-bts. The sequence contains 77 zeros and 51 ones, i.e. unbalanced number of zeros and ones and the key contains long run of zeros. The given key contains five repeated patterns, the key has visible signs and is key cryptanalyst can trace key by tracing the patterns in key and resultant ciphertext to recover the plaintext without having the original key. Other statistical tests are discussed in **Chapter 5**. The randomly generated keys is given as input to Intelligent

Cryptographic Model (ICM) for removing statistical faults and generate a true random key. The internal process of Artificial Neural Network (ANN) given in **Figure 4.8**

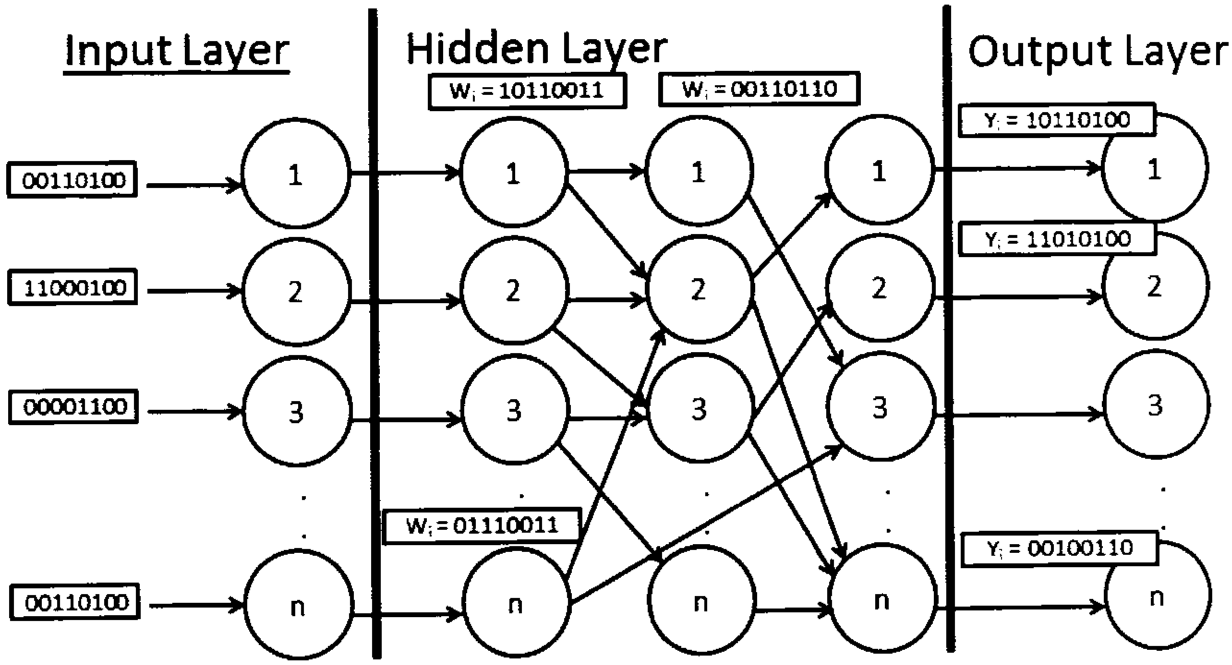


Figure 4.8: Artificial Neural Network (ANN) Interconnected nodes and Layers

The input layer of the Artificial Neural Network (ANN) containing 16 nodes takes the input key generated by Original SNOW 2.0. Each node of the input layer takes one patterns of the given 16 patterns and forward it to the hidden layer. The hidden layer is more complicated and containing 16 X 16 neurons interconnected to each other producing partial mesh topology. The randomly assigned weights are XORed with the new input to neuron and the result is forwarded to the next neuron. The 16th level of neuron in the hidden layer gives its output to the output layer. The error difference is computed between the perceptron current state and previous state. The process continues till the error difference becomes than a threshold value defined by field expert. The final outcome of the hidden layer is passed through various statistical tests to verify that the bit sequence exhibits true random number characteristics.

Note: Given the same input to Artificial Neural Network (ANN) will never give the same output. The input key is randomly treated by the neurons in the hidden layers and the Final outcome is the random operation of each individual neuron. The output of the 16th layer cannot be predicted that which of the neurons actually participated. The complex nature of the hidden layer does not give any valuable information to cryptanalyst to determine the output key.

4.6.5 Perceptron Worst Case Scenario

To determine the time complexity of partially connected neurons is a complex scenario and it is very difficult to determine the total numbers of operations are performed by the hidden layer of Artificial Neural Network (ANN). A true mesh topology having n numbers of nodes connected together has n (n - 1) / 2 number of connections and each node operates XOR operations with the given input and its weight. The output is forwarded to the next neuron which again generates its outcome the similar way and forwards its outcome. The partially connected neurons lies in the $O(n^2)$ region, similarly forward its outcome to the next neurons.

The worst case time complexity grows exponentially and become impractical if the perceptron does not converge. The field expert defines the minimum threshold value and the process continues till the error difference is reduced to a threshold value. In an uncontrolled neural structure millions of keys are generated and forwarded to the output layer. In a controlled neural structure the resultant key at the output layer is verified by passing maximum number of statistical tests.

4.7 Training Artificial Neural Networks (ANN)

In Artificial Intelligence the knowledge acquisition is the bottleneck and most of the time is wasted in this phase. The system learns by induction or conduction analogy and therefore requires a long duration in this process. The expert systems acquire knowledge in the form of rules or frames etc., and process whenever require. It is very difficult to formulate this knowledge and to understand the entire process. Thus in case of missing any step in the process will lead to erroneous knowledge gained by the system. In case of a proper acquirement of the knowledge to be stored in the knowledge base, and gradually increase with process of experience and real world application and thus the decisions made by the system are more appropriate.

In order to get the experts knowledge into an expert system we propose to process these databases in the attempt to learn the particularities of the domain. Whatever is learnt by this process can be discussed with the expert, who is now in the role of a supervisor and consultant that corrects and completes knowledge instead of (often) an unwilling teacher who has to express himself in some form he is not common and not comfortable with. Experts are required to describe their knowledge in the form of symbolic rules, i.e. in a usually unfamiliar form. In particular to describe the knowledge acquired by experience is very difficult. Therefore, knowledge based system (KBS) may not be able to diagnose cases that experts are able to. Some machine learning algorithms, for example Iterative Dichotomiser 3 (ID3) **[47]** have the capability to learn from examples. *Ultsch A. et al* (1994) **[139]** propose to use Artificial Neural Networks (ANN) as a first step of a machine learning algorithm. ANN claim to have advantages over non intelligent systems, being able to generalize and to handle inconsistent and noisy data. Interesting features of natural neural networks are their ability to build receptive fields in order to project the topology of the input space.

To realize the integration, an algorithm has to be constructed, that converts symbolic knowledge for the KBS out of the sub symbolic data of the ANN. The knowledge can be extracted from various data sets in case of serious cryptographic threats to determine a strong key that has more resistance capability. Due to their inherent parallelism ANN are well suited to be mapped on massively parallel computer **[140]**, which can be used for breaking the security of the cryptographic algorithms. If enough knowledge is provided to these systems then, these can be used to improve the security of the system. For example in case of dictionary attacks where the system is enriched with the key words used in English and also with the keys those once used in past are avoided. This work adopts different approach because it not only looks into the data dictionary with the exact match but it also checks the probability of the key to be broken down at run time to nullify the weak key generation.

4.7.1 Frequent Patterns in key and Dictionary

This algorithm compares the key stream with the key dictionary and finds the bit patterns in the given dictionary. It takes variable length patterns and checks with each individual key in the dictionary, and produces "0" output for no match and produces "1" if the pattern exists.

```
Algorithm 4.3: Pattern Matching Algorithm based on Dictionary Attack
Input: Initial Vector ( IV ), Knowledge Base KB[i,j]
Output: Resultant file Containing 0's 1's
Initialize: IV as String
Psize as integer 1,2,3,... bytes
key as string[ ]
set IV ← Input Through keyboard (or any random string)
key = SNOW2(IV)
set i = vcount = 0, j = hcount = 0
do{
        for(i = 0; i ≤ no_of_rows(KB); i++)
          {
            for(j = 0; j ≤ length(key), j++)
             {
                     If (key[i,j] = = KB[i,j])
                     write 1 to Result file
                     else
                     write 0 to Result file
             }
          }
}while (!EOF)
```

Algorithm 4.3: Pattern Matching Algorithm based on Dictionary Attack

The algorithm takes Initial Vector *IV* from key board and takes the knowledge base file containing *M* keys of arbitrary length. The default key size of SNOW 2.0 is 128-bit but the designer can adjust it to any size. This work considers multiple sizes for key analysis and verification and takes a 64-bit and 100-bit long key. This algorithm can be adjusted accordingly depending on the processing power and computational cost. A random key is generated by calling SNOW 2.0 for the random seed IV. The index values of the key are compared with the keys in the dictionary and if the key values at the given index match each other then 1 is stored in the output file otherwise 0. The key is compared with all the values in the KB file and the output is stored in the form of 0's and 1's. The resultant file is used for further processing and is passed through multiples statistical tests and the patterns are observed that how much they match. It helps in deciding that whether the key in stronger enough produced by a given PRNG and can safely be used as secret key for a given security system.

4.7.2 Time complexity

If there is a finite set of *n* elements for *s*, then $|S| = n$, the subsets of this set *S* is $|P(S)| = 2^n$ which is the motivation notation 2^n, and is described as follows:

Let ωi, such that $1 \leq i \leq n$, and the subsets of *S* are $\{ \omega 1, \omega 2, \omega 3, \ldots\ldots, \omega n\}$, where ωi can take values of 0 or 1.If $\omega i = 1$, then any element lies at i^{th} position will be the element in the subset or otherwise. All the elements in the subset are distinct and non-overlapping and will lie under the $O(2^n)$. If m is the size of the knowledge base then the time complexity for the above algorithm will be $M * 2^n$, and lies in the $O(2^n)$ worst case, which is too high to reduce the performance of the key generation of SNOW 2.0.

Instead of searching and comparing individual bit, the keys are arranged in 8-bit patterns. A 64-bit long key has 8 patterns, which are compared with the 8-bit key patterns stored in the dictionary. The patterns matching keys in the dictionary exceeding a threshold limit are rejected and request for a new key.

This work focuses on the similarity of bit patterns and does not care about if a single bit is matching at some specific position. If 8 consecutive bits are similar means one byte is matching at the given index. A 64-bit long key represent 8 ASCII characters and if 4 characters are matching in the key means 50% of the key is compromised. Therefore the cryptanalyst will try for rest of the 32 bits in a given sample space and reduce the target in exponential times i.e. $2^{n/2}$.

4.8 Frequent patterns in Key Stream

The following algorithm checks the key strength by comparing with the knowledgebase of Artificial Neural Network (ANN) and the patterns existing at their respective indexes generated by SNOW 2.0.

```
Algorithm 4.4: Key Verification Algorithm for Key Generated by Intelligent SNOW 2.0
Input:      Seed, KB[i,j]
Output:     Result File
Initialize: IV as String
                Psize as integer 1,2,3,... bytes
                Pcount=0
                key as string[]
                key = precomputedkey_SNOW2(IV)
                tvalue = input;          //Threshold value,
                set i = vcount =0, j = hcount =0
do{
                Pcount = 0;
                for(i = 0; i ≤ no.ofrows(KB); i++)
                {
                        for(j = 0; j ≤ length(key), j++)
                        {
                        if(key[i,j] = = KB[i,j])
                        {
                                write 1 to Result file/Array[]
                                Pcount++;
                        }
                                else
                                write 0 to Result file/Array[]
                        }
                                if pcount ≥ tvalue
                                break;
                }
} while (!EOF)
```

Algorithm 4.4: Key Verification Algorithm for Key Generated by Intelligent SNOW 2.0

The array index and the values at the respective indexes produced by **Algorithm 4.4** are represented in **Table 4.1**. The algorithm works as follows:

Table 4.1: Neural Net Setup Keys for Decision Making

Index	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Value	v0	v1	v2	v3	v4	v5	v6	v7	v8	v9	v10	v11	v12	v13	v14	v15

In first iteration if the value at index {0} is compared with the knowledge base and the values at index {1,2} are compared with the knowledge base, and so on for values at index

{1,2,3},.....,{1,2,3,....15}. In second iteration the value at index {2}, {2,3}, and so forth, wherever the key match is successful unto some threshold level, the key is rejected. The algorithm generates the next key and the process continues until and unless a secure key is found. To compute the time complexity of the algorithm define the parameter used in the algorithm, *n* is the numbers of bytes in key stream, and *m* is the size if the knowledge base, then the worst case time complexity of the algorithm is m (n (n + 1) / 2).

4.9 Summary

Stream Cipher SNOW 2.0 has many challenges as discussed in **Chapter 2**. The major issues in both versions of SNOW are that PRNG is generating the sequence of bits that are susceptible to cryptographic attacks. The key fails to pass most of the statistical tests recommended by NIST. These tests ensure that whether the key stream produced is having such patterns those may lead to break down the entire security. The intelligent algorithm for stream cipher works in different phases. The input layer of Artificial Neural Network (ANN) gets the key generated by SNOW 2.0's PRNG as input that is given to the hidden layer. The hidden layer verifies that whether the key has the bits those may not pass the statistical tests. If the desired output is not received the back propagation algorithm propagates the error message to other neurons and the randomly assigned neuron weights are updated. The process continues till the desired output is received. Artificial Neural Networks is a supervisory learning technique and requires enough data to train the neural system involved in the process. The training data that is initially feed into the ANN is updated time to time and is produced once should not be regenerated and ensure the one time key, that is the major requirement for stream cipher algorithms. The resultant key streams produced by Intelligent SNOW 2.0 pass most of the statistical tests discussed in **Chapter 5**.

CHAPTER 5
RESULTS AND DISCUSSIONS

The key stream generated by the stream ciphers needs to be random and should not be traceable. The whole security of the ciphering algorithm depends upon the resultant key generated by the Pseudorandom Number Generators (PRNG). It is passed through certain statistical and cryptographic tests to determine whether the key is strong or not. The statistical tests ensure that the key is randomly generated and the cryptographic tests ensure that the key is generated randomly, cannot be traced by any known cryptanalytic attack and produce enough confusion in the ciphertext when an encryption is done.

This chapter discusses and compares the resultant key streams generated by SNOW 2.0 and Intelligent SNOW 2.0. A set of candidate keys are selected out of the entire series of keys generated by both versions of algorithms. Random samples of keys of various lengths are selected among the millions of keys produced by their respective PRNGS and passed through various tests. These randomly selected samples are tested to ensure that intelligently generated keys satisfy various statistical tests. These keys can be used as secret keys for encrypting plaintext messages. Following are the main objectives of this research study:

i. To ensure that *Intelligent SNOW 2.0* generates a true random bit sequence and does not have repeated bit patterns.

ii. To show that *Intelligent SNOW 2.0* is more secure and best recommended to be used for encrypting secret data.

iii. To guarantee that *Intelligent SNOW 2.0* is more reliable and the cryptanalyst cannot break its security by any means.

5.1 Repeated Bit Patterns

Various statistical tests are required to determine the repeated bit patterns in a randomly generated weak key. Guess-and-Determine attack **[8]** **[4]** **[141]** have very effective results to compromise the security of SNOW 2.0. Correlation attacks find the correlation between the actual key stream and that of the best matching key [**114**]. In exhaustive search attack the key is traced by checking different patterns in incremental order. The correlation attacks and the Guess-and-determine attacks are based on the algebraic rules and have been proven to be more efficient against stream ciphers. The key generated by the pseudo random numbers generators is used only once and for the second use we need to generate another key.

The cryptanalysts have discovered the techniques that the new key generated by logic based stream ciphers is traceable and the algebraic attacks are more successful. One alternative is to use Nonlinear Feedback Shift Register (NLFSR) based key stream generators but it has its own implications and drawbacks. Another approach that we propose is to use Artificial Neural Networks along with the LFSR based Pseudorandom Number Generators (PRNG). The ANN is proved to be the best suited intelligent algorithm for cryptography, to avoid the traceability in the patterns of the key stream of any of the algorithm specifically the SNOW 2.0.

5.1.1 Error Propagation and Dictionary Attacks

A Pseudorandom Number Generator (PRNG) generates a random number and could not be regenerated by the same or PRNG in polynomial time. Hypothesis testing and other statistical tools are ensuring that the key generated is having maximum characteristics required for a random number. A hypothesis is made that a resultant sequence could either be random or nonrandom. NIST recommended test for verifying randomness are the best ones to ensure that is key is random or otherwise. If a PRNG generates faulty keys then the sequence may be repeated after some period or a part of the key would be repeated that could let a cryptanalyst to successful attack.

In Guess-and-Determine attack, the guess was made on secret key and initialization values, which was used to initialize the LFSR and FSM's registers. On the basis of these guesses, key streams were generated. These key streams were then matched with the original sequence of key streams. If there were large numbers of similarities in both sequences then it was concluded that the attack was successful. Otherwise, more guesses are required or changed the algorithm.

5.1.2 Statistical Tests for Randomness Verification

The resultant key generated by the Pseudorandom Number Generator (PRNG) is passed through various statistical tests to verify the randomness in the generated key. We cannot rely on single test because by passing a single test and it cannot be guaranteed that the generated key is true random but passing various statistical tests it can be guaranteed that the candidate key secure and unbreakable. Intelligent SNOW 2.0 generates 128 bit, 256 bit or any arbitrary size key depending on the choice of the designer. The results discussed in this chapter considered a 64 bit, and 100 bit long keys.

5.2 Randomness Verification

Apart from many other characteristics of Intelligent SNOW 2.0 it behaved like a random sequence. Multiple candidate keys for randomness are tested using National Institute of Standards and Technology (NIST) randomness verification suite. It is observed that randomly generated keys behave differently to these statistical tests. We give a brief detail of the application of NIST-Statistical suite recommended for randomness verification:

1. The Frequency (Monobit) Test
2. Frequency Test within a Block
3. The Runs Test
4. Test for the Longest-Run-of-Ones in a Block
5. The Binary Matrix Rank Test
6. The Discrete Fourier Transform (Spectral) Test
7. The Non-overlapping Template Matching Test
8. The Overlapping Template Matching Test
9. Maurer's "Universal Statistical" Test
10. The Lempel-Ziv Compression Test
11. The Linear Complexity Test
12. The Serial Test
13. The Approximate Entropy Test
14. The Cumulative Sums (Cusums) Test

15. The Random Excursions Test
16. The Random Excursions Variant Test

A brief description of these tests is given in **Chapter 1**. These tests are necessary to verify that whether a sequence is random **[41]** or otherwise. In this section we discuss these tests and its results on both versions of SNOW 2.0 i.e. LFSR based SNOW 2.0 and Intelligent SNOW 2.0.

5.2.1 Frequency (Monobit) Test

The purpose of the frequency (monobit) test is to determine whether the proportion of ones and zeros are approximately same. This test assesses the closeness of the fraction of ones and zeros to ½. Following parameters are required for the test:

n The length of the bit string.

ε The sequence of bits as generated by the RNG or PRNG being tested; this exists as a global structure at the time of the function call; $\varepsilon = \varepsilon_1, \varepsilon_2, \dots, \varepsilon_n$.

s_{obs}: The absolute value of the sum of the *Xi* (where, *Xi* = 2ε - 1 = ±1) in the sequence divided by the square root of the length of the sequence. The reference distribution for the test statistic is half normal (for large *n*). If the sequence is random, then the plus and minus ones will tend to cancel one another out so that the test statistic will be about 0. If there are too many ones or too many zeroes, then the test statistic will tend to be larger than zero.

The Test Description is as follows:

1. Conversion to –1: The zeros and ones of the input sequences (ε) are converted to values of –1 and +1 and are added together to produce $S_n = X_1 + X_2 + \ldots\ldots X_n$ where $X_i = 2\,\varepsilon_i - 1$.

   For example, if ε = 1011010101, then n=10 and S_n = 1 + (-1) + 1 + 1 + (-1) + 1 + (-1) + 1 + (-1) + 1 = 2.

2. Compute the test statistic

$$S_{obs} = |Sn|/sqrt(n) \qquad (5.1)$$

Here in our case S_{obs} = |2| / sqrt (10) = 0.632455532.

3. Compute P-value = erfc (S_{obs} / sqrt (2)) , where *erfc* is the complementary error function defined in **[41]**. Putting values from point 1 in this equation we get:

   P-value = 0.527089

4. Decision Rule (at the 1 % Level)

   If the computed *P-value* is < 0.01, then conclude that the sequence is non-random. Otherwise, conclude that the sequence is random.

5. Conclusion

   Note: In this case the P-value > 0.01, so the given input is random.

Input Size Recommendations

It is recommended that each sequence to be tested consist of a minimum of 100 bits (i.e., n ≥ 100).

Example

i. (input) ε = 1000100000101101000110000100011010011000100110001100110
0010100010111000110010010000111111011010101000

ii. (input) n = 100

iii. (processing) S_{100} = -29

iv. (processing) S_{obs} = 1.16

v. (output) P-value = *0.199509*

vi. (conclusion) Since P-value ≥ 0.01, accept the sequence as random.

5.2.2 Randomness Test Results of Intelligent SNOW 2.0

Randomness verification tests recommended by National Institute of Standards and Technology (NIST) are also applied on the key generated by Intelligent SNOW 2.0. The results are given in **Table 5.1**.

Table 5.1: Intelligent SNOW 2.0 PRNG Randomness Verification Results

S. No	Test Name	Key size	χ 2 value	P-value	Decision Rule (P-value)	Test Result
1	The Frequency (Monobit) Test,	100	NA	*0.199509*	≥0.01	Random
2	Frequency Test within a Block,	100	*6.5*	*0.743806*	≥0.01	Random
3	The Runs Test,	100	NA	0. 400432	≥0.01	Random
4	Test for the Longest-Run-of-Ones in a Block,	100	0.500798	*0.360918*	≥0.01	Random
5	The Binary Matrix Rank Test,	100	1.2619656	0.069532	≥0.01	Random
6	The Discrete Fourier Transform (Spectral) Test,	100	NA	0.339030	≥0.01	Random
7	The Non-overlapping Template Matching Test,	100	*5.999377*	*0.730264*	≥0.01	Random
8	The Overlapping Template Matching Test,	100	*8.965859*	*0. 001104*	≥0.01	Non Random
9	Maurer's "Universal Statistical" Test,	100	NA	0. 733427	≥0.01	Random
10	The Lempel-Ziv Compression Test,	100	NA	*0.005804*	<0.01	Non Random
11	The Linear Complexity Test,	100	*2.700348*	*0. 026845*	≥0.01	Random
12	The Serial Test,	100	*0.845406* *0.336400*	*0. 764843* *0.561915*	≥0.01	Random
13	The Approximate Entropy Test,	100	*5.550792*	0. 530123	≥0.01	Random
14	The Cumulative Sums (Cusums) Test,	100	$z = 1.6$ (forward) $z = 1.9$ (reverse)	0. 194219 (forward) 0.186614 (reverse)	≥0.01	Random
15	The Random Excursions Test, and	100	15.692617	0.037977	≥0.01	Random
16	The Random Excursions Variant Test	100	$J = 1490$	0. 946858	≥0.01	Random

Table 5.1 shows that the candidate random key generated by Intelligent SNOW 2.0 passed 14 tests and failed only two tests i.e. the overlapping template matching test and the linear complexity test. It failed the overlapping template matching test which is about the number of occurrences of pre-specified target strings. It also fails the Lempel-Ziv Compression tests which focus on the number of cumulatively distinct patterns (words) in the sequence. Lempel-Ziv test in included in initial documentation of NIST Statistical tests [44] but it has been removed in the later report published by NIST [142] because the LZ-complexity has a defect such that its distribution of P-values is strictly discrete for random sequences of length 10^6 [143]. It successfully clears all other 14 tests which show that the Intelligent SNOW 2.0 can safely be used for data security and can be deployed in different systems safely.

5.3 Randomness Test Results of Original SNOW 2.0

The tests were also conducted to generate candidate keys by the Pseudorandom Number Generator (PRNG) of the stream cipher SNOW 2.0. Results are reported below:

The candidate sample key i.e. ε =010101010100101111011010110101110000111000010000100

000110101011010111110000010101000111100011010100111

Key size = 100

No. of Zeros = 52

No. of Ones = 48

The test results are given in the **Table 5.2**.

Table 5.2: LFSR-based SNOW 2.0 PRNG Randomness Verification Results

S. No	Test Name	Key size	χ 2 value	P-value	Decision Rule (P-value)	Test Result
1	The Frequency (Monobit) Test,	100	NA	*0.203432*	≥0.01	Random
2	Frequency Test within a Block,	100	*9.5*	*0.564731*	≥0.01	Random
3	The Runs Test,	100	NA	0.654312	≥0.01	Random
4	Test for the Longest-Run-of-Ones in a Block,	100	0.850432	*0.0013432*	≥0.01	Non Random
5	The Binary Matrix Rank Test,	100	3.454343	0.094351	≥0.01	Random
6	The Discrete Fourier Transform (Spectral) Test,	100	NA	0.453213	≥0.01	Random
7	The Non-overlapping Template Matching Test,	100	*9.343464*	*0.765439*	≥0.01	Random
8	The Overlapping Template Matching Test,	100	*4.543453*	*0. 001342*	≥0.01	Non Random
9	Maurer's "Universal Statistical" Test,	100	NA	0. 023412	≥0.01	Random
10	The Lempel-Ziv Compression Test,	100	NA	*0.0342451*	<0.01	Random
11	The Linear Complexity Test,	100	*7.347681*	*0. 004327*	≥0.01	Non Random
12	The Serial Test,	100	*3.4565781* *2.345797*	*0. 986211* *0.456212*	≥0.01	Random
13	The Approximate Entropy Test,	100	*3.235681*	0. 002432	≥0.01	Non Random
14	The Cumulative Sums (Cusums) Test,	100	$z = 3.4$ (forward) $z = 2.14$ (reverse)	0. 45321 (forward) 0.245431 (reverse)	≥0.01	Random
15	The Random Excursions Test,	100	14.563123	0.098311	≥0.01	Random
16	The Random Excursions Variant Test	100	*J = 3451*	0. 234251	≥0.01	Random

Table 5.2 shows that the candidate random key generated by SNOW 2.0 passed 12 tests and failed four statistical tests i.e. the overlapping template matching test and the linear complexity test. It failed the Test for the Longest-Run-of-Ones in a Block which is about the longest run of ones within M-bit blocks, The Overlapping Template Matching Test which is about the number of occurrences of pre-specified target strings, The Linear Complexity Test which is about the longest run of ones within M-bit blocks and The Approximate Entropy Test which is about the frequency of all possible overlapping m-bit patterns across the entire sequence. It also fails the Lempel-Ziv Compression tests which focus on the number of cumulatively distinct patterns (words) in the sequence. It successfully clears all other 12 tests which show that SNOW 2.0 cannot be used safely for data security all the times and the risk of breaking the security algorithm is more.

Hence the bit sequence generated by Intelligent SNOW 2.0 is more reliable acceptable as a random sequence as compared to the bit sequence generated by SNOW 2.0. The efficiency of the stream cipher system can be compromised on the security to an extent, that the major advantage of the stream ciphers i.e. speed must not be reduced to a nominal level. This deficiency does not reflect later on in the implementation of the ciphering algorithm. The key is generated using Intelligent SNOW 2.0 can then be used for the encryption of the plaintext with the same way as in the Original SNOW 2.0 stream cipher algorithm.

5.4 Experimental Analysis

We conducted a series of experiments on both versions of SNOW 2.0 and Intelligent SNOW 2.0 and collected sequence of bits produced by the PRNGs of both versions and passed those through different statistical and cryptographic tests defined in **Chapter 1** and **2**. These sequences of bits also called candidate keys have different characteristics and properties. These keys are generated randomly by the given PRNGs and resulted differently for different samples and populations of the key stream.

It is important to mention that a sample is the representative of the population passes some statistical tests means that the given population of key stream will also pass those tests. If the statistical tests are applied on the entire population then the results are accurate but it is sometimes impossible to conduct it especially in cryptography where the population may have infinite entities. SNOW 2.0 and Intelligent SNOW 2.0, recommends different key sizes i.e. 64 bit key, 128 and 256 bit keys. We limited our experiments to 64 bit keys which resulted in 2^{64} = 18446744073709551616 candidate keys and require 2097152 Terabyte (TB) disk space. It is impossible by all means to conduct such kind of experiments. The key sequence is generated a number of times and stored in different data files to verify the security of PRNGs. The data is analyzed in two ways: The original key is compared with the GD-key file for finding the maximum number of pattern similarities and the frequency tables and graphs are produced. The patterns compared with the respective indexes in the entire file and the frequencies are computed for each index.

The experiments are conducted in two phases:

Phase I: Experiments are conducted on partially trained Intelligent SNOW 2.0 having training data ranging between 200 and 10000.

Phase II: Experiments are conducted on fully trained Intelligent SNOW 2.0 having training data.

5.4.1 Test Phase I (Partially Trained Intelligent SNOW 2.0)

Multiple experiments are performed in this phase in order to determine the traceable bit positions in key. Each experiment has 500, 2000, 5000 and 10,000 attacking key stream respectively. The neural network is partially trained and guessing key is compared with variable size GD-Key file. The Intelligent model is trained with the random sequences, which are stored in the central database. The data file having 500 attacking keys generated by GD-attack is listed in **Table 5.3**:

Table 5.3: 500 Random keys Generated by GD-Attack

1887D6BC	84598765	1A2A57D3	3BFFA8A4	6984AC65	CA283F02	9248191	F55CF774	3C9C82E7	C77A1057
C8802C06	18D5E5FB	73D17E00	0D174AFC	89277D52	F36803D6	EC38DC11	FC630E24	56443c37	B3F560E7
F37D1CE1	A8E69093	187C2BD6	1F9E9F62	D8EA02DA	F5B6C2C5	779F8D3A	7B80E1BE	475AE5BA	7519D21B
69BBF060	A3CAE0A0	62ACFA54	47F61052	87655fe2	CC1A43AF	334E9971	7549482F	77A589FA	67C81551
33011751	F7F8FBA7	9655195D	DA6374C5	BDA0FF8D	4ED18819	F5AE7AC6	7A38EF65	CAD6ABB0	F95A19E3
7258835	83C52C16	D04447E9	4A07E2E4	88D08F9C	1091D7F4	BF095E9C	42742D51	867976DC	D3CCB10F
9A21C15A	17E87F45	A77D4EFA	2EBA4064	FEE22062	828FA275	62BF1037	DCC60057	ED6BA2BC	263CC6E0
286CB4E9	82FD3EC3	8EF4B959	8E88F793	DE9707E6	6A921D30	6.94E+08	DA9601D0	E303053B	2FF02C08
F5731EC4	6E34CB37	567F3E6F	D723291A	BD19CAFD	9684F255	8095EF97	A024FAB7	7964757B	BE3CCFE6
FA88B128	582AD64C	D4A53A89	54CC96CE	8B0870E0	AF83C045	CA04ED2D	FC3C5BDB	F455DB93	6A5AAD73
5BB3E777	96E4C3B1	35A57B6A	5A1C615C	B9D69F85	064AB527	0AB7D54A	6822A679	640FFAFE	3AFB909B

ABDB700D	6A0963A3	B6FF7D46	60C193E8	A09B5318	B1FBD839	CB48DF10	18B2C17F	B9F23A60	491FF9FE
43FCE854	B040E480	A0C33B6F	85114C40	137463D5	129FAF7A	0F9D2D08	48DC76D9	EAB9E402	4A847F80
1FE30D24	D09BB251	5F3A7AD3	64D821E1	BB15D0C8	CF3A69A7	7B258996	D837702B	D9DBF9AE	06D8538A
DAD4F054	3B1CF25B	E7CF8801	6AF64C6C	CEE8062E	0CDCBEA5	11DE6B80	37BD0AC7	D8CCDF42	F223F93E
F15D5836	224D971E	4542356A	F67D9918	85AF8688	9EC00006	DBB9CF0C	309F6856	9912679	3071F90E
58F45628	88EC2020	9A01EEAA	8CD216E7	C11760DB	3D8D48E2	CA2AD97D	00528BFD	09CF59CE	6A6F1CF0
1823EEE4	8EE1F69E	7E184602	F5D3A47A	5E50B1D9	FB4D6A17	80C6A040	05A289AC	E5474FFB	43CBD3B2
52B79164	490DB7D8	FCC917D6	993B7AFA	B0416EE1	ACCD98CB	59C664E4	DCA14BCC	F797604D	DBC74C10
F32D2C5C	A9A75ADE	230A3C8E	16FE3125	6A68EA00	CD0700AD	8F5FC15E	24CB36B7	FEDC81B7	45DE64DD
6AD9D242	B6B616B3	4CBB25B0	EA8511D4	FB845C33	D9C5FA6C	754BF29D	3F206560	93E93CC9	8FC52B2F
098939BC	B0B454C1	302F3980	DBC3028E	2426A104	C9F46FDE	0E5DA2D6	AF448A87	957B32D8	8F39262C
9BCA5A00	D0371E6C	A68C2F09	46639172	40AD3B3D	CA6BC21D	FE285C0E	C5403E14	2E4B2793	C0508915
E8F8D8ED	22F529CC	A4A78725	501F84EA	4F59CC78	C79CAB9F	9BD2C35A	BFFE9617	3F841614	C08FBC12
558E77FE	5824CF15	51B28009	DB798847	0DF66B13	E6D8E218	19875AB8	CB7693FD	D1364D0B	383CA274
F12FE44C	2ED51015	36B64ED5	94B79A08	01E34FAA	ECF135EC	94E181AD	B2A948EE	F1F509EF	052E7538
D68D1D62	4A630B14	316D846E	D5B519FA	FE754721	091228FB	2B0363EC	456DFDC5	979537AE	FB0C5EAA
2B5343F4	2CEC59D2	8080A008	29508881	56B21F2A	F0A40B09	C1F9D674	5F83F9BF	856D1BBB	7FEC6753
0DA3CEF0	EDC77B2F	95F16984	FBDDB6B2	85BF000C	982D3548	C8DBE8F8	9E0905DA	F74D4DAD	73A3645E
6A13E29E	947FFEA9	71174798	F56E33D2	39E21DCE	4104CE04	0BA80FF8	71DFDF0F	A0597BAB	1F6F0DFC
8E1E6633	01B724BE	5A7895CA	3B508011	30879636	29E10FB2	10248ECA	88AC0268	4DDE23B3	928EBBC1
252985D9	3FA70F98	C3FE9CA8	A45F13D7	632ECBB1	15F71B1A	C5E39E6F	DB2482D2	0F7F10F2	C6565454
376919C0	03DBC6C6	A4063F54	28B87F9C	2C35C0E9	BBEA633D	28C8F84A	70C9F3E6	8A20FC0C	20D3040B
93ED75C2	44C67BEF	F98D770E	16B35786	262A29DC	AE53DBA7	FB280EA5	271025A1	FA13E102	E813F861
295BF61E	CC6219C4	8DC93FB1	9C08F1C3	A91160FF	5B9D28F9	7984751	0374CF05	AE1DA4DE	A3DBB38B
F38772A7	398985AF	339BF824	BC3E5F9C	F2798E39	166A68DC	665478A2	CAFB1342	595A209B	7E077B0D
266FBCE2	6590D0F9	2968BB1A	21223B0B	BAB49554	F0CB2A94	A98AF31C	11027551	CD1571CE	39BE4028
1D3DDD47	335567B5	7C1E4D62	509246FA	7C3FB782	D17EF98A	86817804	28A2DEFF	F7EF46B0	28F3D9B6
BD0CC883	DA4F0A4E	FA588387	2B620E8F	2E6BB091	2EB71D27	10AFD551	BA9C6083	C7AE09CB	E3E2C41D
160524C4	AD36BA07	85665373	8B003A4C	395638C1	53CAE422	88C6DC70	89F3F573	3A8EFBC0	05B5E8EA
E44092AE	7818B677	04EF9FA6	DB8C1404	8354B2BD	7D4AA384	059DF7FB	F32CD071	08C86CDE	8394C934
AC1F5391	5F264339	89B74957	93176C95	70FF8736	0326B239	83A00B59	C8889646	4A1DB227	29116345
2E241CA9	9FE24EC8	069F91B9	A5685DAA	9C75E968	5A474F19	20681893	2610E819	0B7219FC	E9F48F84
1462C28A	58F698EB	0D7B27D9	2171B5DA	BD9CD391	2A024755	7A4F9EA1	5F168DE3	B98B78A0	40B2495B
38255F12	ADC870D4	4615C8D1	58E29DEB	4F0F9A13	7B9F0D1D	91CAED1C	BDACCEDB	3A49984A	26649FF1
825B38F9	4D44D37B	888D040C	27250701	10B92234	3DECA663	40AF7FE6	17C43B6A	9F8538F5	8DA8677B
AA4CC44B	448AAF19	3FBBC874	F3673266	51CDE002	A6AF6AA2	0F8D56A4	4546E22F	C6DBAE12	423CA43A
4B3CDD64	0A5E7A66	77D99C03	42513FE3	451AF7C8	3FC873D8	D9373700	AFE257EF	202C6F7E	D5D2D15A
FF3B60C1	AF7E09D3	5BA7B613	262C8C96	A04813CE	29A7FB86	35CF7248	2FC8AD85	2CF7ED17	47733088
4C56A68B	C93A2569	CDCA1334	6C6E0FD6	70EF3340	73D67AD2	E1D71DAA	1A20521C	26DD2B8B	D216A02E

The dictionary size is increased gradually to 1000, 2000, 5000, 10000 and so on, up to the maximum size of 5139189. The cross examination of the results show that if the Intelligent SNOW 2.0 is not properly trained then Intelligent SNOW 2.0 also produce improper keys. The untrained Intelligent SNOW 2.0 will work approximately similar to Original SNOW 2.0.

5.4.1.1 Determining Sub Key Frequency

In first experiment the guess key is 3 and Initialization Vectors (IV) are IV0=2, IV1=2, IV2=2, IV3=2, and compare the key with the attacking key by taking different size of patterns. The results are described in **Table 5.4**.

Table 5.4: Frequency of Sub Keys in Attacking Key Dictionary of Size 217

No. of Patterns (n) **(1 Pattern= 8 bits)**	**Frequency**	**Percentage**
Pattern size n=1	126	58.1%
Pattern size n=2	56	25.8%
Pattern size n=3	35	16.1%
Pattern size n = 4	0	0%
Pattern size n = 5	0	0%
Total		**100.0%**

The data in **Table 5.4** is graphically represented in Figure **5.1.** It can be observed that a pattern of size 8-bits is more frequent in the attacking keys index but by increasing the number of patterns (sub key size) the frequency in the attacking key index gradually decreasing. A 32-bit and 40-bits has a zero frequency in the attacking key index. The graph in **Figure 5.1** behaves like a χ^2distribution because the frequency near the origin is more and the as we go away from the origin the χ^2 graph touches the X-axis, means the frequency of a sub key tends to zero.

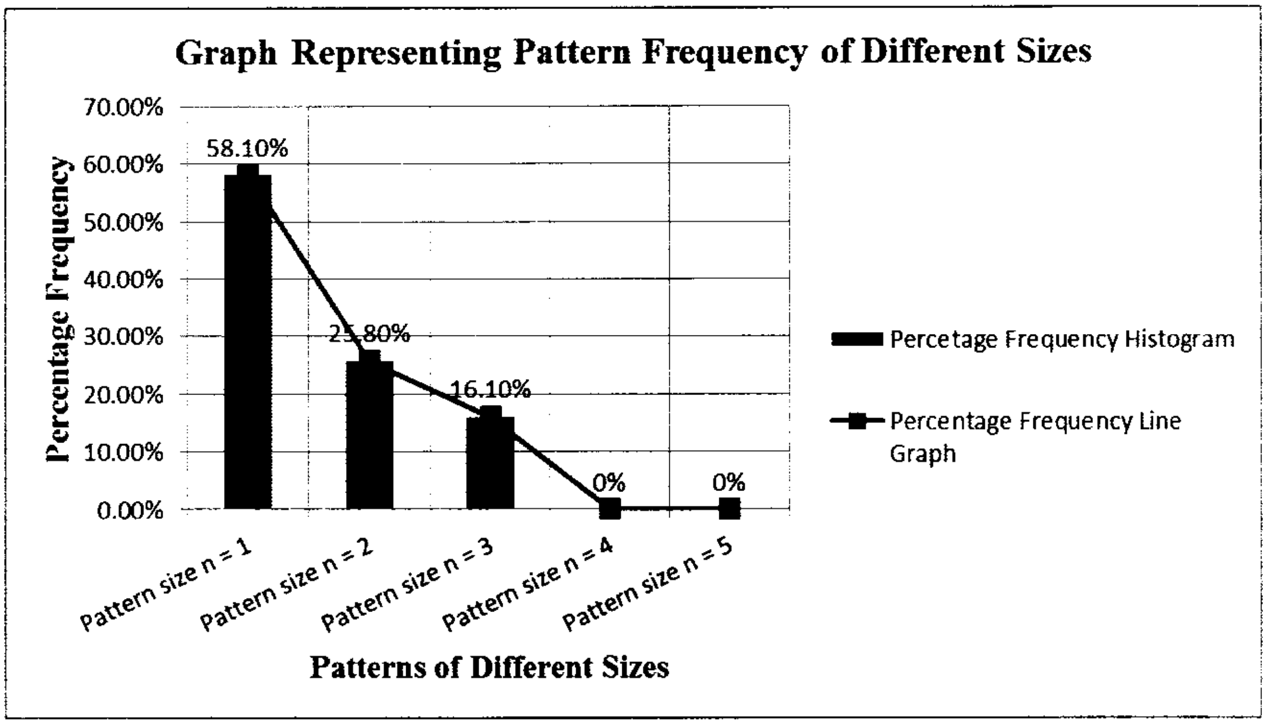


Figure 5.1: Graph Representing Frequency of Sub Keys in Attacking Key Dictionary of Size 217

Taking a 64-bit key that contains 8-patterns of size 8-bits. The pattern match test on a small attacking dictionary of size 217 results that a pattern of size 8-bits has 126 similarities which is 58.1% of the total keys in the attacking dictionary. Increasing the pattern size to 16-bits the attacking dictionary has 58 similarities which is 25.8% of the entire attacking key dictionary. A pattern size is increased to 24-bits and onward has 16.1% and 0% similarities, which shows that GD-attack fails in this case. The graph in **Figure 5.1** lies under the Chi-Square (X^2) having a bell shape towards origin and touching the x-axis for a pattern of size 4 (32-bits). A GD-attack having limited iterations fails in this case, which partially shows that an untrained Intelligent SNOW 2.0 also generates secure key. (Note: This experiment does not prove that an untrained or partial trained Intelligent SNOW 2.0 is unconditionally secure.)

5.4.1.2 Determining Pattern Frequency

A 64-bit key containing 8-patterns is checked against limited GD-attack. The frequency of each pattern in limited key dictionary generated through GD- attack is given in **Table 5.5**.

Table 5.5: Frequency Table Representing the Frequency of Each Pattern (8 bit size) in Attacking Key Dictionary of Size 217

Pattern No.	Frequency	Percentage
P1	24	11.1%
P2	24	11.1%
P3	32	14.7%
P4	33	15.2%
P5	24	11.1%
P6	23	10.6%
P7	35	16.1%
P8	22	10.1%
Total		**100.0**

A 64-bit is divided in 8 patterns of 8-bit size and treated at their respective indexes. The key is checked against 217 key generated through Guess-and-Determine attack, results the facts in represented in **Table 5.5**. The data is graphically represented in **Figure 5.2**.

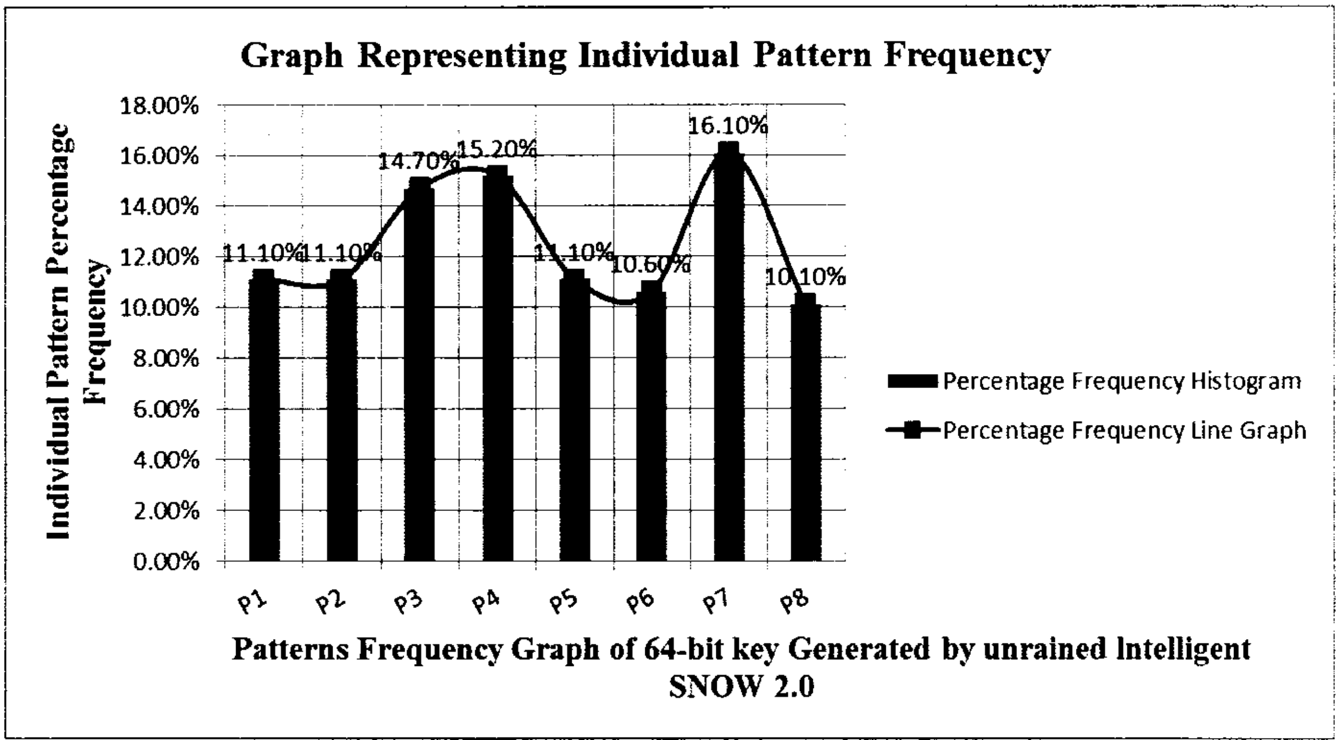


Figure 5.2: Frequency Graph Representing Frequency of Each Pattern (8 bit size) in Attacking Key Dictionary of Size 217

It is clear that some of the patterns are more frequent as compared to others. The cryptanalyst determine the more frequent patterns and guess the other patterns. This shows that when a clever attack is implanted can guess the more frequent patterns and the next frequent patterns. We can see that the value at the index 6 and 3 are 35 and 33, the most frequent values. The cryptanalyst can use different statistical techniques to guess the values at the index 6 and 3 and determine that what could be the values of the next most frequent indexes in the given table. It must be noted that each pattern is 8 bit long and in case of a successful guess 8 bits of the key are cracked and the next are at the risk of hunting them. The cryptanalyst guess the frequent patterns in the key and determines the rest of the key.

The PRNG must produce a series of unbiased bit sequences where each pattern in general and each bit in special must be equally probable. We let the graph be in normal so that the probability of occurrence of each bit pattern must be same. The data gathered must be confused and the cryptanalyst must not get any clue towards the collective collapse of the entire cryptosystem. A closer look at each individual pattern and its relevant frequency shows that there is a reasonable gap in the frequency difference that made the higher frequency patterns more vulnerable to the cryptanalyst. It is required to minimize the frequency difference in given patterns by training Intelligent SNOW 2.0 and adjusting neurons weights. Hence the cryptanalyst could not easily generate the key using dictionary attack or using the Guess-and-Determine attack. A detailed description is given in **Annexure A**.

5.4.2 Test Phase II (Fully Trained Intelligent SNOW 2.0)

Several experiments are performed in this phase in order to determine the traceable bit positions in key. Each experiment had 40,000, 60,000, 80,000 and 100,000 attacking key streams respectively. In the following we give experimental details of a dataset having approximately 1000000 keys.

5.4.2.1 Determining Sub Key Frequency

In this experiment the guess key with initial parameter is 449 and Initialization Vectors (IV) are IV0=6, IV1=6, IV2=6, IV3=2, and compare the guessing key with attacking keys by taking different size of patterns. The results are described in **Table 5.6**.

Table 5.6: Frequency of sub keys in attacking key dictionary of size 100000

No. of Patterns (n) **(1 Pattern= 8 bits)**	**Frequency**	**Percentage**
Pattern size n = 1	59697	59.7%
Pattern size n = 2	31641	31.6%
Pattern size n = 3	7523	7.5%
Pattern size n = 4	1053	1.1%
Pattern size n = 5	79	0.1%
Pattern size n = 6	7	0.0%
Pattern size n = 7	0	0.0%
Total		**100.0**

In **Table 5.6** we see that the guessed key is checked against 100000 keys in the dictionary. A single pattern (8-bits) is observed in 59697 keys, two consecutive patterns (16-bits) are observed in 31641 keys, three consecutive patterns (24-bits) are observed in 7523 keys and so on. Seven and eight consecutive patterns (56-bits / 64-bits) are not observed in the entire GD-attacking file. Other experiments over a million of attacking keys also contain no sign patterns for 56-bits, 64-bits long patterns generated by fully trained Intelligent SNOW 2.0.

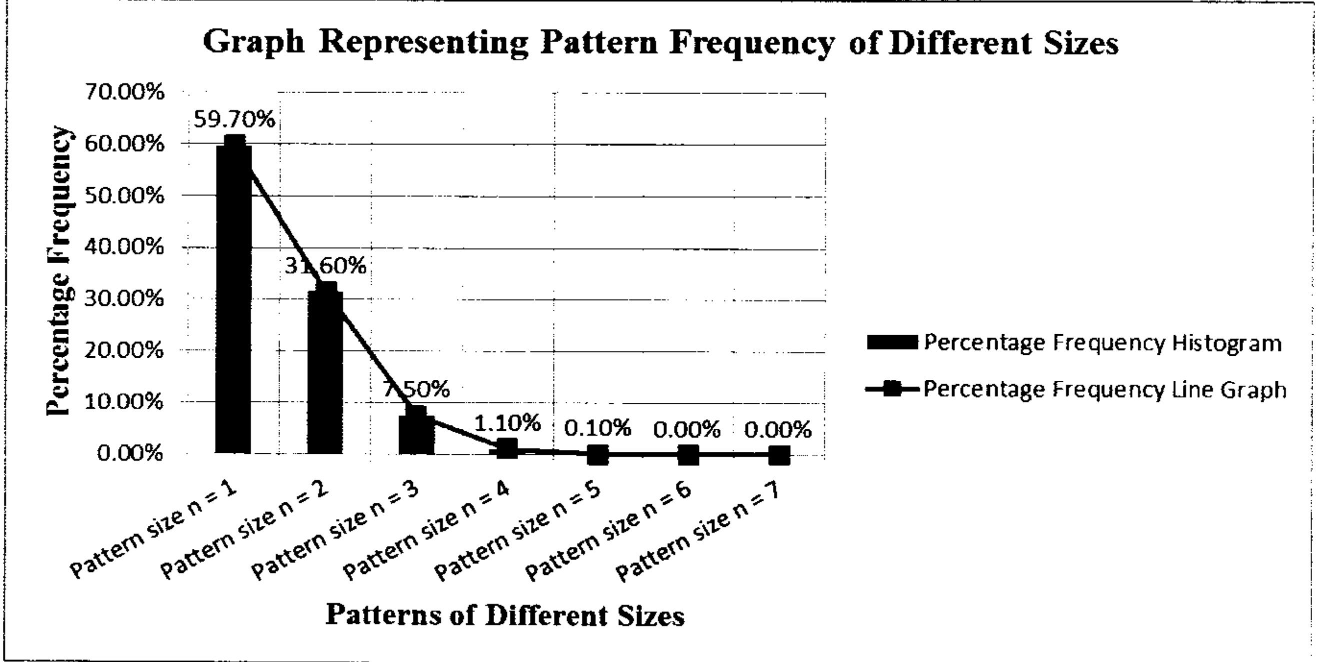


Figure 5.3: Graph Representing Frequency of Sub Keys in Attacking Key Dictionary of Size 100000

The graph in **Figure 5.3** lies under the Chi-Square (χ^2) having a bell shape towards origin and touching the x-axis for a pattern of size 7 (56-bits). A GD-attack having enough iteration fails in this case, which shows that a trained Intelligent SNOW 2.0 generate unconditionally secure key. (Note: This experiment shows that a fully trained Intelligent SNOW 2.0 is unconditionally secure.)

5.4.1.2 Determining Pattern Frequency

A 64-bit key containing 8-patterns is checked against GD-attack. The frequency of each pattern in the given key dictionary generated through GD- attack is given in **Table 5.7**.

Table 5.7: Frequency Table Representing the Frequency of Each Pattern (8 bit size) in Attacking Key Dictionary

Pattern No.	Frequency	Percentage
P1	6393	12.7%
P2	6387	12.7%
P3	6154	12.3%
P4	6165	12.3%
P5	6327	12.6%
P6	6225	12.4%
P7	6223	12.4%
P8	6323	12.6%
Total		**100.0**

Table 5.7 represents the frequency at the respective indexes of the patterns in the keys of the GD-file. The frequency column shows that some of the patterns are more frequent as compared to others. This shows that when a clever attack is implanted can guess the more frequent patterns and the next frequent patterns. We can see that the patterns at all indexes are approximately equally probable. The cryptanalyst has very limited chances to guess the key because the bit patterns in the file are equally probable and there is no biasness in the PRNG that can give a chance to even a single pattern to be guessed by the cryptanalyst. A 128 bit long key that is generated by Intelligent SNOW 2.0 has no chances to be traced by a cryptanalyst in polynomial time. It can be justified that a random sequence generated by Intelligent SNOW 2.0 PRNG is

unconditionally secure. The problem in the untrained Artificial Neural Network (ANN) is resolved by proper training.

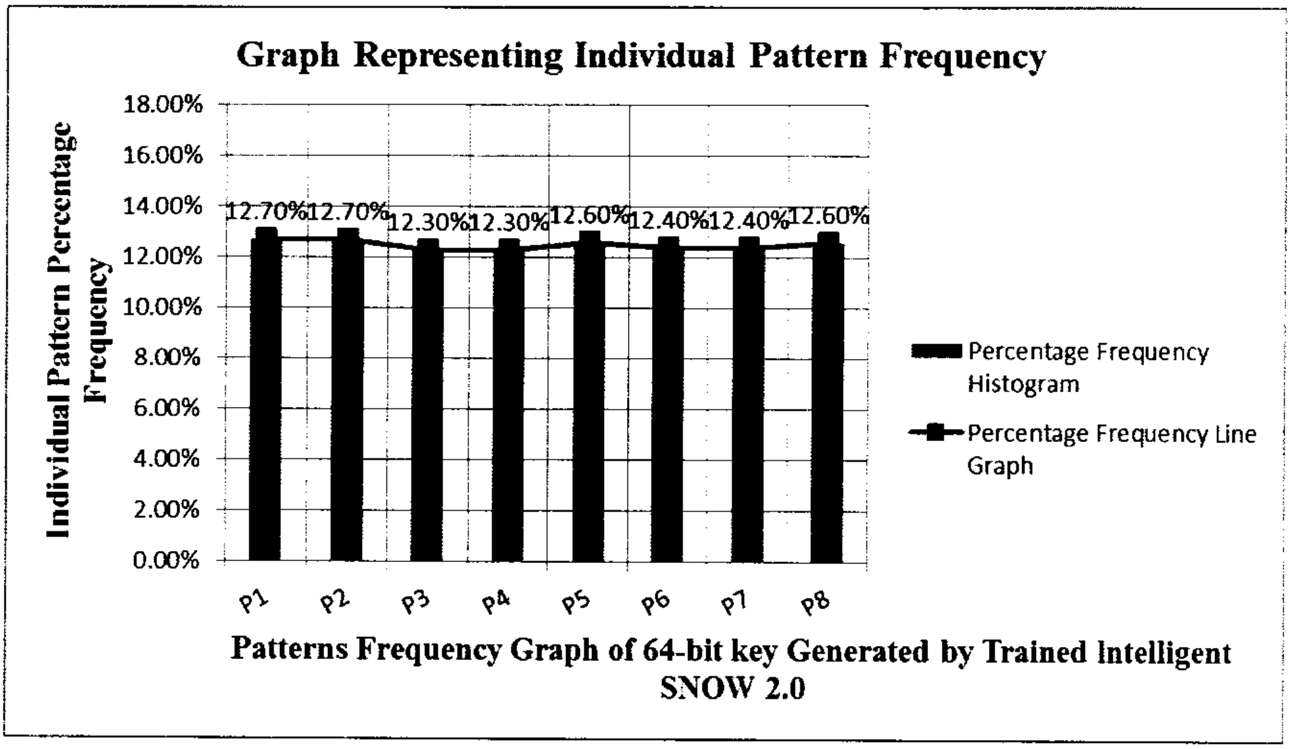


Figure 5.4: Frequency Graph Representing Frequency of Each Pattern (8 bit size) in Attacking Key Dictionary

A closer look at each individual pattern and its relevant frequency revealed that there is no reasonable gap in their frequency difference, and the probability of occurrence of each individual byte was approximately same as shown in **Figure 5.4**. A cryptanalyst cannot clearly determine the frequent patterns because the frequency of all patterns at their respective indexes is approximately same. A 128-bit and 256-bit key cannot be recovered in polynomial time.

5.5 Comparison of SNOW 2.0 and Intelligent SNOW 2.0

A series of experiments are conducted to break the code generated by Original SNOW 2.0 and fully trained Intelligent SNOW 2.0. **Table 5.8** contains data collected after analysis of Original SNOW 2.0 and Intelligent SNOW 2.0.

Table 5.8: Frequency Table for Sub Key of Size n=1,2,…., 8, Using Original SNOW 2.0 and Intelligent SNOW 2.0

	Intelligent SNOW 2.0		Original SNOW 2.0		
No. of Patterns (1 Pattern= 8 bits)	**Total No. of Similarities**	**Percentage Similarity**	**Total No. of Similarities**	**Percentage Similarity**	**Difference**
1	1629420	31.7057808%	1631213	31.7406696%	1793
2	379276	7.38007495%	396494	7.71510836%	17218
3	50016	0.97322749%	51210	0.99646073%	1194
4	4003	0.07789167%	5120	0.09962661%	1117
5	144	0.002802%	542	0.01054641%	398
6	2	3.8917E-05%	52	1.01E-03%	50
7	1	1.9458E-05%	8	1.56E-04%	7
8	0	0%	1	1.94583E-05%	1

Table 5.8 contains information about Original SNOW 2.0 and Intelligent SNOW 2.0. It can be observed at all stages that Intelligent SNOW 2.0 has collectively low similarity index values as compared to Original SNOW 2.0. The Difference Column shows the difference in similarities between both the competing algorithms. The sample data contains 8 keys that have 7 patterns similar to that of the original key generated by SNOW 2.0 and 1 key is 100% similar to that of the original key. A 64-randomly generated key generated by Original SNOW 2.0 Pseudorandom Number Generator (PRNG) has been cracked and it cannot be used as a secret key.

The sample is the representative of the actual data keys of its original length, the GD attack is equally likely applicable to that of the key containing 128 bits. Intelligent SNOW 2.0 algorithm produce a sample key that cannot be cracked and the key list produced by GD attack does not contain any key similar to the secret key produced by Intelligent SNOW 2.0. The statistical facts in **Table 5.8** prove that Intelligent SNOW 2.0 is more secure and the series of keys produced by it can be used unconditionally as a secret key. The data is graphically represented in **Figure 5.5** and **Figure 5.6**.

The graph in **Figure 5.5** represents that a single pattern (byte=8 bits) has the index similarity with 1631213 keys and 396494 keys are having 2 patterns similar to that of the Original key produced by Original SNOW 2.0. We can see that the secret key exactly match with a single attacking key in the list. It shows that a 64-bit key generated by Original SNOW 2.0 has been compromised and cannot be used as secret key.

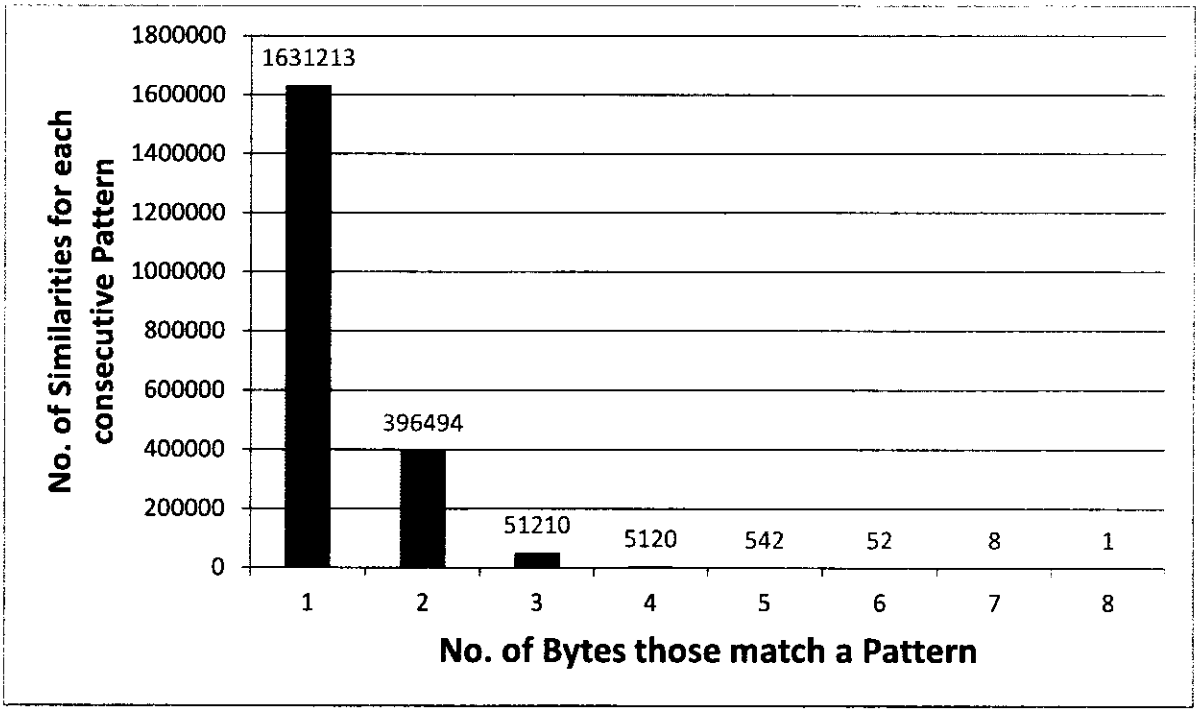


Figure 5.5: Frequency Graph for Sub Key of Size n=1,2,...., 8, Using Original SNOW 2.0

The graph in **Figure 5.6** represents that a single pattern(byte=8 bits) has the index similarity with 1629420 keys and 379276 keys are having 2 patterns similar to that of the Original key produced by Intelligent SNOW 2.0. A 64-bit key generated by Intelligent SNOW 2.0 is not fully recovered. In large amount of guesses it is observed that the Intelligent SNOW 2.0 has more similar patterns, however it is not always the case. The probabilistic and intelligent nature of Intelligent SNOW 2.0 algorithm restricts statistically weak keys generation.

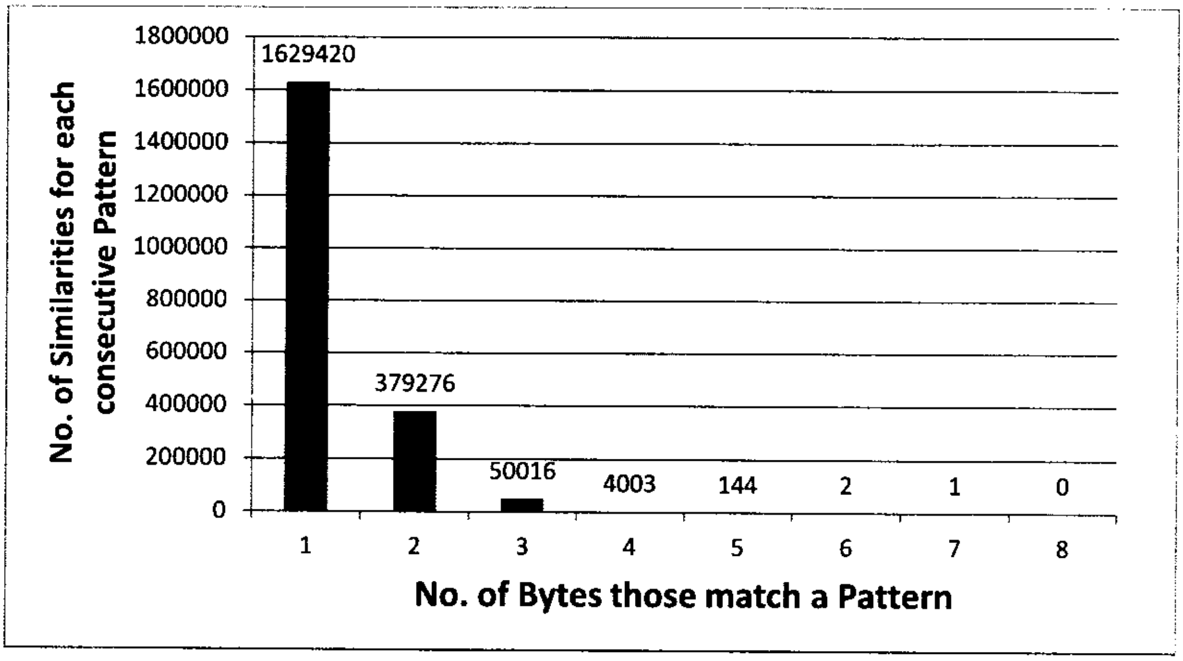


Figure 5.6: Frequency Graph for Sub Key of Size n=1,2,...., 8, Using Intelligent SNOW 2.0

A true random and cryptographically strong key generation is a critical issue in stream cipher algorithms. A randomly selected key is also required to be guaranteed random and has no statistical weaknesses that may support an analyst to break it. Intelligent Cryptographic Model (ICM) store weak key information in knowledge base and a new randomly generated key is required to avoid weak key properties. If the model is trained more intelligently, new situations are fed to the inference engine and the knowledge base is also increased then it respond more intelligently and produce keys containing more security features. The keys produced can be trusted to be used as secret keys. The intelligent algorithm reports the weak keys and is kept in the knowledge base so as to avoid the chances of reuse such a weak key.

Figure 5.7 presents combined graph of Original SNOW 2.0 and Intelligent SNOW 2.0. The Red bar represents Intelligent SNOW 2.0 and the Green shaded bar represents Original SNOW 2.0. The bar graphs and the labels over it show that the patterns similarity is more frequent in Original SNOW 2.0 as compared to Intelligent SNOW 2.0. The Intelligent SNOW more reliably produces the random number which can be used as a secret key. Moreover the pictorial representations of these facts show that Intelligent SNOW 2.0 is more secure than Original SNOW 2.0.

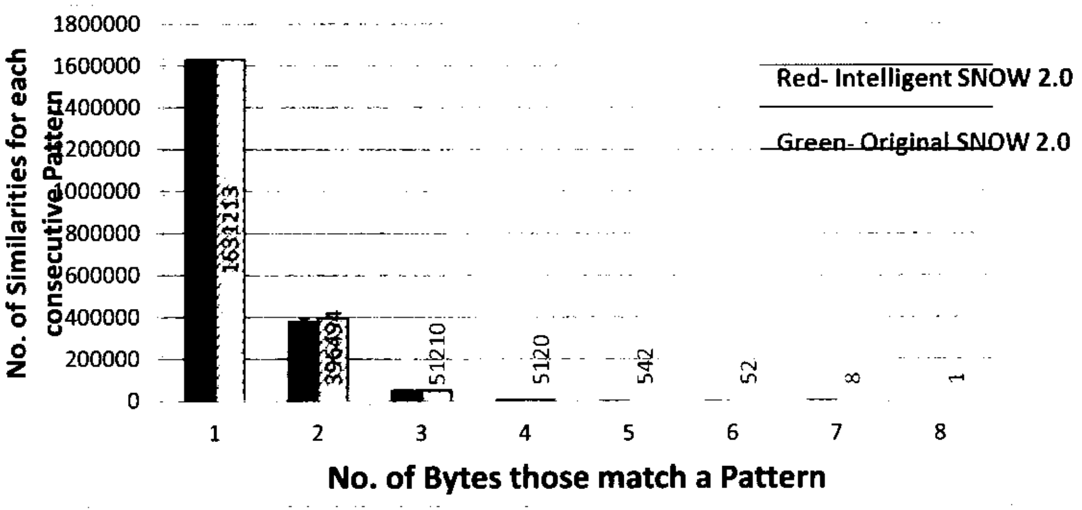
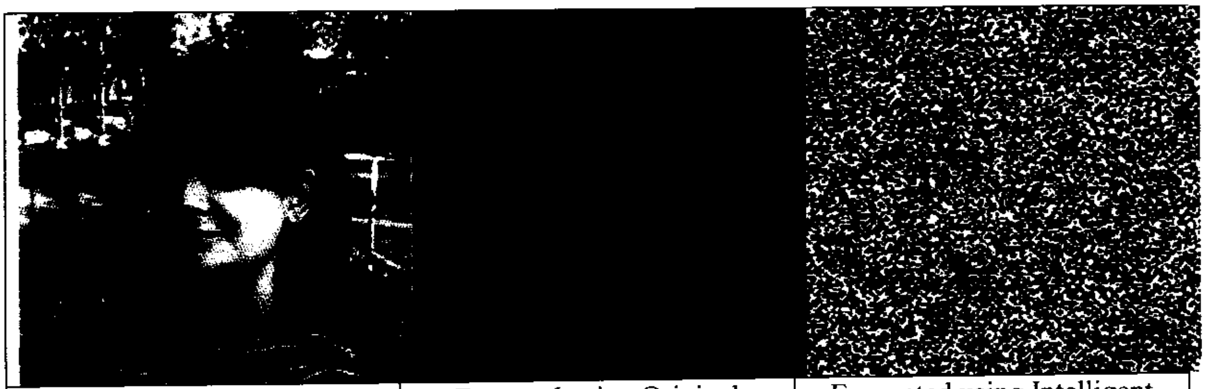


Figure 5.7: Frequency Graph for sub key of size n=1,2,…., 8, Using Original SNOW 2.0 and Intelligent SNOW 2.0

5.6 Entropy in Ciphertext

Entropy destroys structure by adding confusion and disorder to the data during an encryption process - it is the complete converse of ordered structure. Entropy negates structure and converts plaintext into the state of neutral latency i.e. ciphertext thus making it meaningless to adversaries by direct inspection at least. That ambition is not always fully realized in secure communications. However, due to the clever skills of the cryptanalyst who believes that there is still some residual structure in the ciphertext even after Alice has done her best to conceal it and which he must find.



Original Image	Ecrypted using Original SNOW 2.0	Encrypted using Intelligent SNOW 2.0

Figure: 5.8: Entropy in Image using Original SNOW 2.0 and Intelligent SNOW 2.0

Entropy can easily be understood in an encrypted image. For example an encrypted image gives shades of background and a foreground having scars of male or female sketches. This means that the encrypted image is not fully confused and may provide enough information to the analyst to recover the secret key. A statistically proven secure secret key having no repeated bit patterns may highly confuse the encrypted image having no supported information that may enable a cryptanalyst to recover the secret key.

Data + Structural Information = Visible Information

Entropy = (-ve) Structure

Data + Structural Information + Entropy = Minimizes Visual Information

If

Entropy = Structural Information

Then

Data + Structural Information + Entropy = No visible Information

Entropy is the negative structure information and negates the structural orientation of the data provided. Thus by negating the structured form of data it will not be possible to collect meaningful information. An encrypted text must not contain any valuable information. A cryptanalyst checks the ciphertext whether it contains some sign from which the secret key can be guessed. If a statistically weak key having repeated patterns is used as a secret key for encrypting the plaintext may produce the same ciphertext repeated at various parts in the encrypted message. The cryptanalyst on successful guessing one ciphertext pattern recover the key pattern used for encrypting it. If some patterns of the ciphertext are successfully recovered the cryptanalyst deduce rest of the text. A cryptographically strong key will destroy the structural information by XORing with the plaintext and a clever ciphertext only attack will not be able to successfully recover the secret key patterns used for encrypting the relevant plaintext. To concentrate on the patterns of smaller size and work with it has no fruits to bring into front because in that case the rest of the key is again untraceable for the cryptanalyst while applying different statistical tests. There was a big amount of disbursement in the data we collect and it was difficult for the analyst to find the regularity in data.

A 1-MB (10^6-Bytes) plaintext data is encrypted with 64-bit randomly generated key using both Original SNOW 2.0 and Intelligent SNOW 2.0. Correlation attack is used to recover a 64- bit long key used for encryption in SNOW 2.0 and Intelligent SNOW 2.0. The pattern similarities observed in both algorithms are given in **Table 5.9**.

Table 5.9: Frequency Table for Sub Key of Size n=1, 2, …., 8, using SNOW 2.0 and Intelligent SNOW 2.0 (Large Patterns only)

No. of Patterns **(1 Pattern= 8 bits)**	**Intelligent SNOW 2.0** *(Total No. of Similarities)*	**Original SNOW 2.0** *(Total No. of Similarities)*
Pattern of Size n=5	144	542
Pattern of Size n=6	2	52
Pattern of Size n=7	1	8
Pattern of Size n=8	0	1

A chunk of 5, 6, 7 patterns is observed at 144, 2, 1 locations in the ciphertext and the key patterns used for encryption are recovered using correlation attack. A ciphertext generated through Original SNOW 2.0 is containing one 64-bit chunk used for recovering the 64-bit randomly generated key. The cryptanalyst may also use chosen ciphertext only attack by attaching compression virus with the plaintext message. On successful receiving the ciphertext on public channel the cryptanalyst recover the plaintext by observing the visual information trend the ciphertext. The data from the **Table 5.9** above has been summarized in the line graph (**Figure 5.9**) below for the larger patterns created so far. It is evident to see that the line is below the Original SNOW 2.0 showing significant improvement in using the Intelligent SNOW 2.0.

The graph in **Figure 5.9** contains blue line representing Intelligent SNOW 2.0 and red line representing Original SNOW 2.0 clearly show that it converges in both cases but the Intelligent algorithm converges more rapidly. It shows that visible information is deeply negated and does not give enough support to cryptanalyst to recover the secret key.

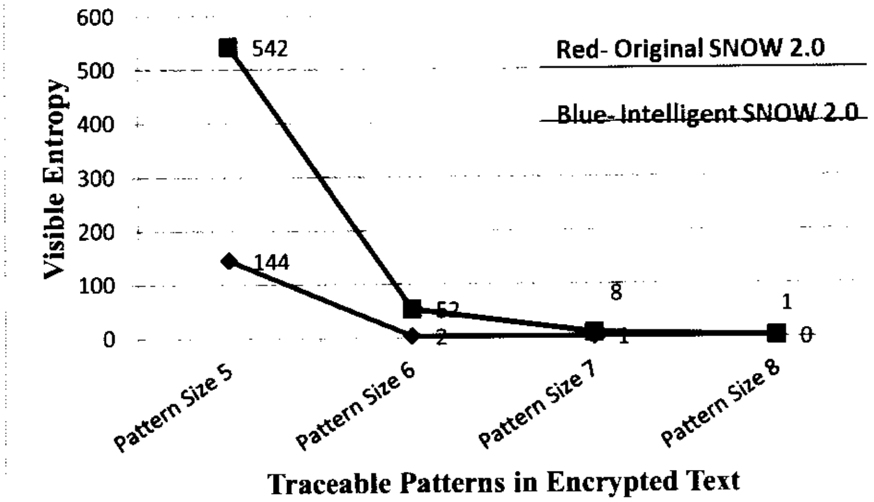


Figure 5.9: Conversion Graph for Original SNOW 2.0 and Intelligent SNOW 2.0.

5.7 Data Dispersion

It is in contrast with the central tendency which is a central or typical value for a probability distribution. The dispersion is a statistical property in which the central or typical value for a probability distribution is dispersed and is confused with the given data to find out any clue over a given region. In cryptography it has a great importance because a cryptanalyst tries to find out a central or typical value for a probability distribution. The resultant key patterns must not be traceable by applying various probability distributions which are used for normalizing a statistically disbursed data. The ciphertext does have valuable information to recover the relevant plaintext and a series of randomly generated sequence also does not give a clue about the next randomly generated key. The cryptanalyst brings data to normal form and compute the Coefficient of Variance. The Coefficient of Variance below 100% can be normalized to collect valuable information and the data is considered reliable to some extent.

A cryptographer always tries to increase dispersion in a given sequence and the given data or information must not fit under any of the aforementioned distributions. If the data is more dispersed more that data will be secure and the cryptanalyst will not be able to get any clue from a sample over a given population. If the Coefficient of Variance (CV) is more than 100%, then statistically this data is considered unreliable which is the ultimate goal of a cryptographer in a given security system. To find the dispersion and uncertainty in data that has been obtained while using the original SNOW 2.0 and the Intelligent SNOW 2.0. The mean variance and the Coefficient of Variance of the data generated by Intelligent SNOW 2.0 are computed below:

Table 5.10: Data Obtained using the Intelligent SNOW 2.0

S. No	x_i	$(x_i - \bar{x})$	$(x_i - \bar{x})^2$
1	1629420	1371560	1881183005625.0625
2	379276	121418.25	14742391433.065
3	50016	-207841.75	43198193043.0625
4	4003	-257855.75	6444223409.75625
5	144	-257713.75	66416376939.0625
6	2	-257855.75	66489587808.0625
7	1	-257856.75	66490103520.5625
8	0	-257857.75	66490619235.0626
n=8	$\sum x_i = 2062862$		$\sum (x_i - \bar{x})^2 = 2269452511701.5$

Mean of $x = \frac{\sum x_i}{n}$ (5.2)

$$\bar{x} = \frac{2062862}{8}$$

$$\bar{x} = 257857.75$$

The percentage coefficient of variance is:

$CV\% = \frac{\sigma}{\bar{x}} \times 100$ Where Standard deviation, $\sigma = \sqrt{(x_i - \bar{x})^2 / n}$ (5.3)

$$CV\% = 206.555$$

The coefficient in the given is 206.555% which shows that the data collected from Intelligent SNOW 2.0 is highly abnormal and unreliable. The cryptanalyst tires to find any relevancy among randomly generated keys which is highly dispersed and fails to guess the patterns in a given key. The cryptanalyst cannot get sufficient information from the set of keys randomly generated by Intelligent SNOW 2.0 PRNG. The randomly generated key is untraceable and cannot be compromised through any cryptanalytic attacks.

Table 5.11: Data Obtained using Original SNOW 2.0

S. No	x_i	$(x_i - \bar{x})$	$(x_i - \bar{x})^2$
1	1631213	1370633	1878634820689
2	396494	135914	18472615396
3	51210	-209370	43835796900
4	5120	-255460	65259811600
5	542	-260038	67619761444
6	52	-260528	67874838784
7	8	-260572	67897767184
8	1	-260579	67901415241
n=8	$\sum x_i = 2084640$		$\sum (x_i - \bar{x})^2 = 2277496827238$

$$\text{Mean of } x = \frac{\sum x_i}{n} \tag{5.4}$$

$$\bar{x} = \frac{2084640}{8}$$

$$\bar{x} = 260580$$

The percentage Coefficient of Variance is:

$$CV\% = \frac{\sigma}{\bar{x}} \times 100 \qquad \text{where Standard Deviation, } \sigma = \sqrt{(x_i - \bar{x})^2 / n} \tag{5.5}$$

$$CV\% = 204.759$$

Coefficient of Variance is expressed as the ratio of Standard Deviation and Mean value. It is the measure of variability and stability of the data. In this work it is computed for both versions of SNOW 2.0, i.e. Original SNOW 2.0 and Intelligent SNOW 2.0. When the value of Coefficient of Variance is high, the data has high variability, less stability and less reliability. The Coefficient of Variance (CV) of Intelligent SNOW 2.0 data is very high i.e. 206.555% and the data is highly unreliable and a cryptanalyst cannot get any valuable information. The Coefficient of Variance (CV) of Original SNOW 2.0 original data is 204.75% which is too high and unreliable but less than Intelligent SNOW 2.0. Thus the intensity of the statistical attacks on the Intelligent SNOW 2.0 algorithm is reduced due to the vast dispersion in its data. In case of a clever attack it is impossible to break the entire key in the polynomial time and to find the correlation between the plaintext and ciphertext by applying different statistical tests because of the increased dispersion in the key produced by using Intelligent SNOW 2.0.

5.8 Summary

The probabilistic nature of the pseudorandom number generators does produce results of deterministic in nature. It is impossible to achieve 100% unbiased results in computer systems. Because computers are based on logic and the logic based systems always produce biased results, although its level may be too less as we have shown in **Chapter 4**. The Intelligent Cryptographic Model (ICM) is more secure as it maintains the knowledge base. When a biased key is generated by the Pseudorandom Number Generator (PRNG), its biasness will be determined as shown and justified in this chapter. After a series of experiments and statistical proofs we see that the data collected for the intelligent stream cipher has more confusion and dispersion as compared to the data collected for the original stream cipher algorithm. In this chapter we have tested Intelligent SNOW 2.0 by different means as discussed in **Section 5.4**, **5.5** and **5.6**, which proves that Intelligent SNOW 2.0 is more secure. In **Section 5.2** and **5.3** Original SNOW 2.0 and Intelligent SNOW 2.0 are tested separately and it has been shown that Intelligent SNOW 2.0 passed 14 NIST recommended statistical tests whereas SNOW 2.0 passed 12 NIST recommended randomness test. The cryptanalyst who tries to enter through the backdoors to compromise the security of the system will not get any way to produce the plaintext without having the real key. It is shown by applying different statistical tests that Intelligent SNOW 2.0 is more secure than LFSR based SNOW 2.0 and data can be transferred more safely over a public channel.

CHAPTER 6
CONCLUSION AND FUTURE WORK

Information security is one of the burning issues since inception of electronic data transfer. Its history is as old as the human race. The world has now become a global village and each and every individual from any part of the world can communicate with each other through internet and other electronic systems. Information security is now considered as necessary as other necessities of life. Both sender and receiver of data have reservations about information security. If information is of national interest then it has become more important to take special measures for its security. The most widely used medium for data transfer is the public network i.e. the Internet. The most threatening challenge to this communication is that digital communications are intercepted by an individual whom these are not intended. The purpose of this dissertation is to keep the information more secure and confidential during digital communications. If an unauthorized person gets an access to data even then he/she should not be able to decipher the text in polynomial time. Block cipher and stream cipher algorithms are used for this purpose.

6.1 Conclusion

Stream cipher algorithms are best recommended in high speed communication where computational resources are weak and communication channels are noisy. The main problem with stream ciphers is that these are comparatively less secure than block ciphers. The security of a stream cipher algorithm is dependent upon the secret key size and its strength. We can increase the key size to improve the stream cipher's security in a hope that it may stop the brute force attacks to trace the secret key in polynomial time. The cryptanalyst also try to implant various cryptanalytic attacks like algebraic attacks, Guess-and-Determine attacks, differential key analysis and other statistical attacks those may compromise the security of a stream cipher algorithm. Intelligent SNOW 2.0 can also be used in the time critical environments because it does not increase the processing time during the encryption and decryption process. The key size of Intelligent SNOW 2.0 is same i.e. 128-bits or 256-bits and using the same encryption and decryption techniques. Intelligent SNOW 2.0 generates a true random key which can be used as a secret key for a given session. The cryptanalyst tries by all means to compromise the security of a ciphering scheme by applying dozens of attacks and decides that which of the cryptanalytic attack would be more successful to break its security. Following are the main features of this work:

i. The focus in this dissertation is not to produce new stream cipher algorithms but to present a new model i.e. Intelligent Cryptographic Model (ICM) which uses Artificial Intelligence techniques to improve the security of stream ciphers.

ii. Artificial Intelligence (AI) is one of the growing and dominant fields and covers approximately all areas of computer science. We believe that in near future computers will also think and act intelligently like human beings. This work fills up a gap to use artificial intelligence in cryptography to generate a sequence of bits that would be used as a secret key.

iii. The Intelligent Cryptographic Model (ICM) generates a random key and then it confirms that weather the key is strong enough to be used as a secret key. It is a very important task because if a randomly generated key is blindly used, it may let a cryptanalyst to implant very serious attacks on a cryptographic model.

iv. Randomly generated key are categorized as strong and weak keys and a newly generated key is ensured that has no statistical weaknesses. A newly generated key require passing various statistical tests before using it as secret key to verify that it has no cryptographic vulnerabilities.

v. Intelligent Cryptographic Model (ICM) is flexible and it can be fused with any other stream cipher. It is proved that Intelligent Cryptographic Model (ICM) is more secure than logic based stream ciphers.

vi. Different kinds of statistical analysis proved that Intelligent SNOW 2.0 is more secure than Original SNOW 2.0. as discussed below:

   a. It has been found that Intelligent SNOW 2.0 passed 14 statistical tests and failed 2. It failed The Overlapping Template Matching Test and the Lempel-Ziv Compression Test where the Lempel-Ziv Compression Test has been removed from the current Statistical suite [**140**].

   b. It has been found that Original SNOW 2.0 passed 12 statistical tests and failed 4. Thus comparing both the results it has been justified that Intelligent SNOW 2.0

generates random sequence of bits more effectively than that of Original SNOW 2.0.

c. Both Intelligent SNOW 2.0 and Original SNOW 2.0 are tested for 8-bit pattern frequency and it has been observed that Intelligent SNOW 2.0 produces the patterns at the respective index with the same frequency. This test ensures that the cryptanalyst cannot get any information that some of the patterns are generated more frequently. This test proved that the probability of all patterns at their respective indexes is same.

d. It has been observed that the keys generated by both Original SNOW 2.0 and Intelligent SNOW 2.0 are compared against a large data file having a large number of keys generated against them. Intelligent SNOW 2.0 does not give 100% resemblance with the keys generated through Guess-and-Determine attack, whereas the Original SNOW 2.0 has serious cryptographic vulnerabilities.

e. It has been observed that Intelligent SNOW 2.0 hides the entire information with higher degree of entropy in the ciphertext and converges more frequently as compared to Original SNOW 2.0.

f. It has been concluded that the keys produced by Intelligent SNOW 2.0 are extremely dispersed and the data collected by cryptanalyst does not give any support to trace the original key.

We conclude that Intelligent SNOW 2.0 is more secure than Original SNOW 2.0. The future cryptographers can trust on Intelligent Stream Ciphers to use these algorithms in their security models. No matter what ever stream cipher algorithm they use but its intelligent version will always support their security system as most of the weaknesses recommended by the cryptanalysts have been removed during its implementation process.

6.2 Recommendations and Future Work

In the following we give guidelines for the future researchers:

i. Intelligent stream ciphers for firmware require special care, because no significant work is done in this area. It may perform faster than intelligent software but the major issue in firmware is to update the knowledge base required for Artificial Neural Network (ANN).

ii. Making encryption process intelligent may produce more confusion in the ciphertext and the cryptanalyst will not be able to recover the plaintext back without having the key and the equivalent decryption mechanism. The major issue in this case is that it requires more computational cost and memory, which may not be possible some times. It may also propagate error in case of erroneous data transfer.

iii. Instead of using single key, several keys can be used for encrypting a plaintext. Only the communicating parties may know its deciphering scheme, to recover the relevant plaintext. It requires special training that Artificial Neural Network (ANN) decides properly the key utilization during encryption and decryption algorithm.

iv. Special Intelligent stream cipher algorithms are required for mobile ad hoc networks. Secure communication in ad hoc networks is still a burning issue which can be resolved by designing special intelligent stream ciphers for mobile ad hoc networks. Low memory and processing power are the major issues in designing intelligent algorithms for ad hoc networks.

v. Special intelligent stream cipher algorithms are required for secure communication in vehicular networks. Military convoys are mostly targeted due to the leakage of information when they communicate in hazardous zones. Intelligent stream ciphers will provide more security during communication among the vehicles and headquarters.

We believe that much and more are done in this area but the researchers still need to explore the diamond from the mines of coals. The Artificial Intelligence is most dominant area in the next future and still more need to be done for its use in cryptography.

Bibliography

[1] Leeuwen J.V., *Handbook of Theoretical Computer Science, Algorithms and Complexity.* New York: Elsevier, 1990.

[2] Pelzl J., Paar C., *Stream Ciphers, Understanding Cryptography, Atext book for students and Practitioners*, 2nd ed.: Springer, 2009, vol. 1.

[3] Arani D., Dasgupta A., "Analysis of Different Types of Attacks on Stream Ciphers and Evaluation and Security of Stream Ciphers," *Cite-Seerx*, 2005.

[4] Matt J.B., Robshaw A, "Stream Ciphers, RSA Laboratories Technical Report TR-701," RSA Technical Laboratory, Red Wood city, 1995.

[5] Patrik E., "On LFSR based Stream Ciphers - analysis and design," Department of Information Technology, Electrical and information technology, Lund, Sweden, Ph.D Dissertation (Monograph) 91-628-5868-8, 2003.

[6] Matt J. B., Robshaw A., "Stream Ciphers Technical Report TR-701," RSA Laboratories, Technical Report 1995.

[7] Irfanullah S., Khiyal S. H., Tabassum A. A., "Cryptanalytic Weaknesses in Modern Stream Ciphers and Recommendations for Improving Their Security Levels," in *Proceedings of the IEEE Symposium on Emerging Technologies*, Islamabad, 2005, pp. 236 - 241.

[8] Irfanullah S., Khiyal M.S.H., Naz T., "Traceable Bit Streams in SNOW 2.0 using Guess-and-Determine Attack," *World Applied Sciences Journal*, pp. 190-195, 2010.

[9] Ahmadi H., Salehani Y.E., "A Modified Version of SNOW2.0," in *International Conference on Digital Communication, 2007, Santa Carla*, Santa CArla, 2007, pp. 123-133.

[10] Ekdahl P., Johansson T., "A New Version of Stream Cipher SNOW," *Selected Areas in Cryptography*, pp. 47-61, February 2003.

[11] Stallings W., *Cryptography and Network Security, Principles and Practices*, 3rd ed. NJ: Prentice Hall, 2003.

[12] IBM Knowledge Center. (2015, October) Cryptography. [Online]. http://www-01.ibm.com/support/knowledgecenter/SSFKSJ_7.0.1/com.ibm.mq.csqzas.doc/sy10500a.gif

[13] Ayushi, "A Symmetric Key Cryptographic Algorithm," *International Journal of Computer Applications*, vol. 1, no. 15, pp. 1-4, 2010.

[14] Barry K. Shelton, *Introduction to Cryptography*, 2nd ed. Loss Angelles, USA: Auerbach Publications, 2010.

[15] Menezes A., Vanstone S., Oorschot P.V., *Handbook of Applied Cryptography*, 1st ed.: CRC Press, 1996.

[16] Rivest R. (1987) Vocal. [Online]. http://www.vocal.com/cryptography/rc4-encryption-algoritm/

[17] Wikipedia. (2015, October) Wikipedia(Block Cipher Mode of Operation). [Online]. https://en.wikipedia.org/wiki/Block_cipher_mode_of_operation

[18] Menezes A., Vanstone S., Oorschot P.V., *Handbook of Applied Cryptography*, 2nd ed.: CRC Press, 1996.

[19] Ye L., "Applied Stream Ciphers for Mobile Communications," B.Eng. in Computer Science & Technology, , Beijing Polytechnic University, China, Beijing, Ph. D. Thesis 2006.

[20] Wikipedia. (2015, July) Wiki Pedia(Stream Cipher). [Online]. https://en.wikipedia.org/wiki/Stream_cipher

[21] Ward E. (2015, October) Cryptanalysis of the GSM Algorithms. [Online]. http://www.oocities.org/eoinward/images/gsmcracked.html

[22] Bhadeshia H. K. D. H., "Neural Networks in Materials Science," *Journal of the Iron and Steel Institute of Japan*, pp. 1-25, 1999.

[23] Bishop C.M., *Neural Networks for Pattern Recognition.* London: Oxford University Press, 1995.

[24] Jackson P., *Introduction To Expert Systems*, 3rd ed. Boston, MA, USA: Addison-Wesley Longman Publishing Co., 1998.

[25] Kevin L., Priddy K.L., Keller P.E., *Artificial Neural Networks: An Introduction.*: SPIE Press, 2005. [Online]. http://www.eeng.dcu.ie

[26] Wikipedia. (2015, October) Artificial Neural Networks/Print Version. [Online]. https://en.wikibooks.org/wiki/Artificial_Neural_Networks/Print_Version

[27] Nielsen M., "Using Neural Nets to Recognize Handwritten Digits," in *Neural Networks and Deep Learning*. USA: Bugfinder Hall of Fame, 2015, ch. 1, pp. 1-1000.

[28] Siegenthaler T., "Decrypting a Class of Stream Ciphers Using Ciphertext Only," *IEEE Transactions on COmputers*, vol. C-34, no. 6, pp. 81-85, August 2006.

[29] Johansson T., "Analysis and Design of Modern Stream Ciphers," *Cryptography and Coding*, vol. 2898, pp. 66-72, December 2003.

[30] Golić J.D., "Linear Cryptanalysis of Stream Ciphers," *Fast Software Encryption*, vol. 1008, pp. 154-169, June 2005.

[31] Johansson T., Jönsson F., "Improved Fast Correlation Attacks on Stream Ciphers via Convolutional Codes," *Advances in Cryptology*, no. 1592, pp. 347-362, 1999.

[32] Biham E., Dunkelman O., "Cryptanalysis of the A5/1 GSM Stream Cipher," *Progress in*

Cryptology, vol. 1977, pp. 43-51, April 2002.

[33] Ekdahl P., Johansson T., "Another Attack on A5/1," *IEEE Transactions on Information Theory*, pp. 284 - 289, January 2003.

[34] Rogaway P., Coppersmith D., "Software-Efficient Pseudorandom Function and the use thereof for Encryption," Computer Security US5454039 , 1995.

[35] Xiao G., Shan W. Ding C., "The Stability Theory of Stream Ciphers," *Journal oj Information Security and Cryptography*, vol. 561, 2011.

[36] Couture N., Kent K.B., "The Effectiveness of Brute Force Attacks on RC4," in *Proceedings of the Second Annual Conference on Communication Networks and Services Research, 2004.*, 2004, pp. 333 - 336.

[37] Ahmadi H., Eghlidos T., "Advanced Guess and Determine Attacks on Stream Ciphers," in *Fifth International Conference on Information Assurance and Security, 09/2009*, Tehran, 2009.

[38] Sekar G., Preneel B., "Improved Distinguishing Attacks on HC-256," *Advances in Information and Computer Security*, vol. 5824, pp. 38-52, October 2009.

[39] Banegas G., "Attacks in Stream Ciphers: A Survey," *Journal of Applied Cryptography*, vol. 26, no. 2, pp. 1-16, August 2014.

[40] Rose G., Hawkes P., "On the Applicability of Distinguishing Attacks Against Stream Ciphers," in *In Proceedings of the 3rd NESSIE Workshop*, 2006, pp. 321-329.

[41] Barker W., Burr W., Polk W., Smid M., Barker E., "NIST Recommendations for Key Management," U.S. Department of Commerce, National Institute of Standards and Technology, Gaithersburg, NIST Special Publication 2012.

[42] Rock A., "Pseudorandom Number Generators for Cryptographic Applications," Paris-Lodron-Universit¨at Salzburg, 9780691025469, 2005.

[43] Jakobsson M., Bruce K. H., Juelsy A., Shriver E., "A Practical Secure Physical Random Bit Generator," in *CCS '98 Proceedings of the 5th ACM conference on Computer and communications security*, New Yourk, 1998, pp. 103-111.

[44] Rukhin A., Nechvatal J., Smid M., Barker E., Leigh S., Levenson M., Soto J., "A StatisticalL Test Suite For Random and Pseudorandom Number Generator for Cryptographic Applications," National Institute of Standards and Technology- NIST, Technical Teport 2001.

[45] Chambers W. G., "On Random Mapping and Random Permutation," *Fast Software Encryption*, vol. 1008, pp. 22-28, June 2005.

[46] Vazirani U.V., "Towards a Strong Communication Complexity Theory or Generating Quasi-random Sequences from Two Communicating Slightly-random Sources," in *Proceedings on the 17th Annual ACM Symposium on theory of Computing*, Toranto, 2012, pp. 366-378.

[47] Quinlan J.R., "Learning Efficient Classification Procedures and their Application to Chess End Games," *Machine Learning*, pp. 463-482, 1983.

[48] Ultsch A., Korus D., Li H., Guimaraes G., "Knowledge Extraction from Artificial Neural Networks and Applications," *Parallele Datenverarbeitung mit dem Transputer*, pp. 148-162, 1994.

[49] Eberhart R., Simpson P., "Computational Intelligence PC Tools," *IEEE Transactions on Neural Networks*, pp. 8-17, August 2002.

[50] Frisch A., Allen J., "What Are Semantic Networks? A Little Light History," Toranto, 1982.

[51] Mantin I., Shamir A., Fluhrer S., "Weaknesses in the Key Scheduling Algorithm of RC4," *Selected Areas in Cryptography*, vol. 2259, pp. 1-24, December 2001.

[52] Wagner D., Dawson E., Kelsey J., Millan W., Schneier B., Simpson L., "Cryptanalysis of

ORYX," *Selected Areas in Cryptography*, vol. 1556, pp. 296-305, March 2002.

[53] Fluhrer S.R., McGrew D.A., "Statistical Analysis of the Alleged RC4 Keystream Generator," *Fast Software Encryption*, pp. 19-30, 2001.

[54] Rogaway P., Coppersmith D., "A Software-Optimized Encryption Algorithm," *Journal of Cryptology*, pp. 273-287 , September 1998.

[55] Boesgaard M., Pedersen T., Christiansen J., Scavenius O., Vesterager M., "Rabbit: A New High-Performance Stream Cipher," *Fast Software Encryption*, vol. 2887, pp. 307-329, 2003.

[56] Philip H., Gregory G. R., "Turing: A Fast Stream Cipher," *Fast Software Encryption*, vol. 2887, pp. 290-306, February 2003.

[57] Lu Y., Vaudenay S., "Cryptanalysis of Bluetooth Keystream Generator Two-Level E0," *Advances in Cryptology*, vol. 3329, pp. 483-499, 2004.

[58] Filiol E., Fontaine C., "A New Ultrafast Stream Cipher Design: COS Ciphers," *Cryptography and Coding*, vol. 2260, pp. 85-98, December 2001.

[59] Halevi S., Jutla C., Coppersmith D., "Scream: A Software-Efficient Stream Cipher," *Fast Software Encryption*, vol. 2365, pp. 195-209, July 2002.

[60] Wu H., "A New Stream Cipher HC-256," *Fast Software Encryption*, vol. 3017, pp. 226-244, February 2004.

[61] Hongjun W., "The Stream Cipher HC-256," *New Stream Cipher Designs*, vol. 4986, pp. 39-47, 2004.

[62] Wu H., "A New Stream Cipher HC-128," *New Stream Cipher Designs*, pp. 39 - 47, April 2008.

[63] Luo L., Qin Z., Zhang W., Zhu S., Wei Z., "Golden Fish: An Intelligent Stream Cipher Fuse Memory Modules," International Association for Cryptologic Research, Beijing,

Research Paper July 10, 2009. [Online]. https://eprint.iacr.org/2009/611.pdf

[64] Luo L., Qu Z., Hai Dai Q. Ye Y., "A Mini Fish Tailed Lion the Intelligent Fishbone Based on Golden Fish," in *International Conference on Systems and Informatics (ICSAI)*, Yantai, 2012, pp. 2556 - 2558.

[65] Masood M., Khiyal M.S.H., Arshad G., Khan A., "Analysis and Design of Non-Linear SNOW 2.0 for improved security," *International Journal of Computer Technology and Engineering*, vol. 3, no. 5, October 2011.

[66] Hawkes P., Rose G. G., "Guess and Determine Attack on SNOW," *Selected Areas in Cryptography*, pp. 37-46, February 2003.

[67] Zhang B., Meier W., Xu C., "Fast Correlation Attacks over Extension Fields, Large-Unit Linear Approximation and Cryptanalysis of SNOW 2.0," *Advances in Cryptology*, vol. 9215, pp. 643-662, August 2015.

[68] Watanabe D.,Cannière C. D., Biryukov A., "A distinguishing attack on SNOW 2.0 with linear masking method," *Selected Areas in Cryptography*, vol. 3006, pp. 222-233, 2004.

[69] Khan S., Khiyal M.S.H., Naz T., Khan A., "Dynamic Feedback Based Stream Cipher Modified SNOW 2.0," in *IEEE International Conference on Emerging Technologies*, Islamabad, 2010, pp. 244-250.

[70] Khan A., Khiyal M. S. H., Naz T. Khan S., "Dynamic feedback based modified SNOW 2.0," in *6th International Conference on Emerging Technologies (ICET), 2010*, Islamabad, 2010, pp. 250 - 255.

[71] Joan R., Daemen V., *The Design of Rijndael*, 1st ed. Berlin, Germany: Springer-Verlag Berlin Heidelberg, 2002.

[72] Bogdanović M., Stošić L., "RC4 stream cipher and possible attacks on WEP," *International Journal of Advanced Computer Science and Applications*, pp. 110-114, August 2012.

[73] Paul S., Preneel B., "A New Weakness in the RC4 Keystream Generator and an Approach to Improve the Security of the Cipher," *Fast Software Encryption*, vol. 3017, pp. 245-259, 2004.

[74] Vladimor V. C., Smeets B., Johansson T., "A Simple Algorithm for Fast Correlation Attacks on Stream Ciphers," *Fast Software Encryption*, vol. 1978, pp. 181-195, January 2002.

[75] Pudovkina M., "Statistical weaknesses in the alleged RC4 keystream generator," in *International conference on Cryptographic research*, Moscow, 2002, pp. 1-19. [Online]. https://eprint.iacr.org/2002/171.pdf

[76] Gilbert H., Handschuh H., "χ2 Cryptanalysis of the SEAL Encryption Algorithm," *Fast Software Encryption*, vol. 1267, pp. 1-12, May 2006.

[77] Pedersen T., Vesterager M., Zenner E., Boesgaard M., "The Rabbit Stream Cipher - Design and Security Analysis," in *In Workshop Record of the State of the Arts of Stream Ciphers*, 2004, pp. 150-159.

[78] Joux A., Muller F., "A Chosen IV Attack Against Turing," *Selected Areas in Cryptography*, vol. 3006, pp. 194-207, August 2004.

[79] Biryukov A., Wagner D. Shamir A., "Real Time Cryptanalysis of A5/1 on a PC," *Fast Software Encryption*, vol. 1978, pp. 1-18, january 2002.

[80] Babbage S.H. (2001, September) Cryptology ePrint Archive. [Online]. http://eprint.iacr.org/2001/078

[81] Babbage S., "The COS Stream Ciphers are Extremely Weak," , Newbury, UK, 2001, pp. 70-78.

[82] Maximov A., Johansson T., "A linear distinguishing attack on Scream," in *Proceedings. IEEE International Symposium onInformation Theory, 2003*, 2003.

[83] Zenner E., "A Cache Timing Analysis of HC-256," *Selected Areas in Cryptography*, vol. 5318, pp. 199-213, August 2009.

[84] Spillman R., Nelson B., Kepner M., Janssen M., "Use of A Genetic Algorithm in the Cryptanalysis of Simple Substitution Ciphers," *Cryptologia*, vol. 17, no. 1, pp. 31-44, Januay 1993.

[85] Clark J. A., "Optimization Heuristics for Cryptology," Queensland, univeristy of Technology, http://sky.fit.qut.edu.cn/~clarka/papers/thesis-ac.pdf, PhD thesis 1998.

[86] Van V. J. H., Grundlingh W. (2003) Using Genetic Algorithms to Break a Simple Cryptographic Cipher. [Online]. http://dip.sun.ac.za/~vuuren/abstracts/abstr_genetic.htm

[87] Servos W., "Using a genetic algorithm to break Albertic Cipher," *Journal of Computing Sciences in Colleges*, vol. 19, pp. 294-295, May 2004.

[88] Peleg S., Rosenfeld A., "Breaking Substitution Cipher using Relaxation Algorithm," *Advances in Artificial Intelligence*, pp. 598-605, 1979.

[89] Albassal M.B., Wahdan M.A., "Genetic Algorithm Cryptanalysis of the Basic Substitution Permutation Network," in *Proceedings of the 46th IEEE International Midwest Symposium on Circuits and Systems*, 2003, pp. 471 - 475.

[90] Hamza A., Al-Salami M., "Timing Attack Prospect for RSA cryptanalysts Using Genetic Algorithm Technique," *The International Arab Journal of Information Technology*, vol. 1, pp. 81-85, January 2004.

[91] Nalini N., Rao G.R., "Cryptanalysis of Simplified Data Encryption Standard Via Optimaisation Heuristics," in *Third International Conference on Intelligent Sensing and Information Processing*, vol. 6, December 2005, pp. 74 - 79.

[92] Clark J. A., "Invited Paper-Nature-Inspired Cryptography: Past, Present and Future," in *Conference on Evolutionary Computation, Special Session on Evolutionary Computation in Computer Security and Cryptography*, Canberra, 2003, pp. 1647-1654.

[93] Khaled M. G., Jalab H.A., "Data Security Based on Neural Networks," *Task Qarterly*, vol. 9, no. 4, pp. 409–414, 2005.

[94] Shihab K., "A Back Propagation Neural Network for Computer Network Security," *Journal of Computer Science*, vol. 2, no. 9, pp. 710-715, 2006.

[95] Necla Ö., Seref S., "Neural Solutions for Information Security," *Journal of Polytechnic*, vol. 10, no. 1, pp. 21-25, 2007.

[96] Dalkiran I., Danis K., "Artificial Neural Network Based Chaotic Generator for Chaotic Generator for Cryptology," *Turk J Elec Eng& Comp Science*, vol. 18, no. 2, pp. 255-240, February 2010.

[97] Karam M. Z. O., AL Jammas Mohammed H., "Implementation of Neural - Cryptographic System using FPGA," *Journal of Engineering Science and Technology*, vol. 6, no. 4, pp. 411 – 428, 2011.

[98] Volna E., Kocian V., Janosek M., Kotyrba M., "Cryptography Based on Neural Network," in *IEEE Annual India Con*, May 2005, pp. 1-7.

[99] Zhang J., Li T., Li J., "A Cryptanalysis Method based on Niche Genetic Algorithm," *International Journal of Applied Mathematics & Information Sciences*, vol. 8, no. 1, pp. 279-285, December 2013.

[100] Shaker W., Awad M., "Designing Stream Cipher Systems Using Genetic Programming," in *5th International Conference, LION 5, Rome, Italy, January 17-21, 2011. Selected Papers*, vol. 6683, Rome, Italy, 2011, pp. 308-320.

[101] John A. C., Maitra S., Stănică P., Jacob J.L., "Almost Boolean Functions: The Design of Boolean Functions by Spectral Inversion," in *The 2003 Congress on Evolutionary Computation, 2003. CEC '03.*, vol. 3, December 2003, pp. 2173 - 2180.

[102] Millan W., Dawson E., Clark A., "An Effective Genetic Algorithm for Finding Highly Nonlinear Boolean Functions," in *ICICS '97 Proceedings of the First International*

Conference on Information and Communication Security, Beijing, China, 1997, pp. 149–158.

[103] Ratten R., "The applications of Genetic Algorithms in cryptology," *Proceedings of the Third International Conference on Soft Computing for Problem Solving*, vol. 258, no. 1, pp. 821-831, 2014.

[104] Lawrence J., *Introduction to Neural Networks*, 2nd ed. California: California Scientific Software Press., 1995.

[105] Beavers A. F., *Alan Turing: Mathematical Mechanist.*: Waltham- Elsevier, 2013, vol. 1.

[106] Laskari E. C., Stamatiou Y. C., Vrahatis M. N., Meletiou G. C., "Cryptography and Cryptanalysis Through Computational Intelligence," *Computational Intelligence in Information Assurance and Security*, vol. 57, pp. 1-49, 2007.

[107] (2013, July) Wikipedia(Artificial Neural Network). [Online]. https://en.wikibooks.org/wiki/Artificial_Neural_Networks/Print_Version

[108] Ruttor A., "Neural Synchronization and Cryptography," Department of Computer Engineering, Cornell University, Ph.D Thesis 2007.

[109] Lan L., Wang J., Guang Z., "Intelligent Application Conversion of Block Ciphers for Different Network Layers," *International Journal of Innovative Computing Information and control*, vol. 3, no. 1, March 2009.

[110] Ruttor A., Naeh R., Kanter I., Kinzel W., "Genetic attack on neural cryptography," *Physical Review of statistical, nonlinear, and soft matter physics*, vol. 2, pp. 1-8, December 2005.

[111] Stankovski P., "Cryptanalysis of Selected Stream Ciphers," Department of Electrical and Information Technology, Faculty of Engineering, LTH, Lund University, Lund, Sweden, Ph. D thesis 2013.

[112] Babbage S. H., "Improved "Exhaustive Search" Attacks on Stream Ciphers," in *European Convention on Security and Detection*, May 1995, pp. 161–166.

[113] Armknecht F., "Improving Fast Algebraic Attacks," *Fast Software Encryption*, vol. 3017, pp. 65-82, 2004.

[114] Canteaut A., Anne C., "Correlation Attack for Stream Ciphers," *Encyclopedia oj Cryptography and Security*, pp. 261–262, 2011.

[115] Adi S., Jonathan J. H., "Fault Analysis of Stream Ciphers," *Cryptographic Hardware and Embedded Systems*, vol. 3156, pp. 240-253, 2004.

[116] Biryukov A., Wagner D., "Slide Attacks," *Fast Software Encryption*, pp. 245-259, May 2001.

[117] Biryukov A., Wagner D., "Advanced Slide Attacks," *Advances in Cryptology*, pp. 589–606, May 2000.

[118] Biham E., Keller N., Dunkelman O., "Improved Slide Attacks," *Fast Software Encryption*, vol. 4593, pp. 153-166, 2007.

[119] Priemuth-Schmid D., Biryukov A., "Slide Pairs in Salsa20 and Trivium," *Progress in Cryptology*, pp. 1-14, 2008.

[120] Dinur I., Shamir A., "Cube attacks on Tweakable Black Box Polynomials," *Advances in Cryptology*, pp. 278–299, April 2009.

[121] Dinur I., Adi R., "Applying Cube Attacks to Stream Ciphers in Realistic Scenarios," *Cryptography and Communication*, vol. 4, no. (3-4), pp. 217–232, August 2012.

[122] Biryukov A., Adi S., Shamir A., "Cryptanalytic Time/Memory/Data Tradeoffs for Stream Ciphers," *Advances in Cryptology*, vol. 1976, pp. 1-13, October 2000.

[123] Verdult R., Balasch J., Flavio D. G., "Gone in 360 Seconds: Hijacking with Hitag2," in *In Proceedings of the 21st USENIX Conference on Security Symposium, Security'12*,

Berkeley, CA, USA, 2012, pp. 37-47.

[124] Michael L., *Pseudorandomness and Cryptographic Applications*, 13th ed.: Princeton University Press, 1996.

[125] Ihaka R., Fenstermacher P., Davis D., "Cryptographic Randomness from Air Turbulence in Disk Drives," *Advances in cryptology*, vol. 839, 1994, pp. 114-120, 2011.

[126] Cybenko G., "Approximation by superpositions of a sigmoidal function," *Mathematics oj Control, Signals and Systems*, vol. 5, no. 4, p. 455, December 1989.

[127] Siegelmann H.T., Sontag E.D., "Analog computation via neural networks," in *Proceedings of the 2nd Israel Symposium on theTheory and Computing Systems*, Natanya, Israel, June 1993, pp. 98 - 107.

[128] Belavkin R.V., "Do Neural Models Scale up to a Human Brain?," in *International Joint Conference on Neural Networks, 2007. IJCNN 2007.*, Orlando, FL, 2007, pp. 2312 - 2317.

[129] Warren S., Pitts W., Culloch M., "A Logical Calculus of the Ideas Immanent in Nervous Activity," *The bulletin of mathematical biophysics*, vol. 5, no. 4, pp. 115-133, December 1943.

[130] Stanton P. K., Terrence J. S., "Covariance Storage in the Hippocampus," *An Introduction to Neural and Electronic Networks*, pp. 365--377, 1989.

[131] Huyck C., Belavkin R., "Counting with Neurons: Rule Application with Nets of Fatiguing Leaky Integrate and Fire Neurons," in *Proceedings of the seventh international conference on cognitive modelling*, 2006, pp. 142–147.

[132] Gupta D.K., "Modeling the Relationship Between Air Quality and Intelligent Transportation System (ITS) with Artificial Neural Networks," Department of civil and Environmental Engineering, University of Louisville, Ph. D thesis 2008.

[133] Hornik K., White H., Stinchcombe M., "Multilayer Feedforward Networks are Universal Approximators," *Neural Networks*, vol. 2, no. 5, pp. 359–366, 1989.

[134] Ji C., Psaltis D., Snapp R., "Generalizing Smoothness Constraints from Discrete Samples," *Neural Computation*, vol. 2, no. 2, pp. 188 - 197, May 2014.

[135] Poggio T., Girosi F., "Networks for Approximation and Learning," in *Proceedings of the IEEE*, vol. 78, 2002, pp. 1481 - 1497.

[136] Cun Y. Le., "A Theoretical Framework for Back Propagation," in *Peoceedings of 1988 Connectionist Model Summer School Carnegie Melon University*, 1989, pp. 21-28.

[137] Irfanullah S., Khiyal M.S.H. Khan M. A., "Intelligent Algorithm Design of the LFSR Based Stream Cipher," *World Applied Sciences Journal*, vol. 30, no. 4, pp. 498-505, 2014.

[138] C Csáji B. C., "Approximation with Artificial Neural Networks," Faculty of Mathematics and Computer Science, Eötvös Loránd University Hungary, Hungry, M. Sc. Thesis 2001.

[139] Ultsch A., Korus D., Li H., Guimaraes G., "Knowledge Extraction from Artificial Neural Networks and Applications," *Parallele Datenverarbeitung mit dem Transputer*, pp. 148-162, September 1994.

[140] Palm G., Rückert U., Ultsch A., "Wissensverarbeitung in neuronaler Architektur," *Verteilte Künstliche Intelligenz und kooperatives Arbeiten*, pp. 508-518, 1991.

[141] Lano J., "Cryptanalysis and Design of Synchronous Stream Ciphers," Department Elektrotechniek- East, Katholike University Leuven, Kasteelpark Arenberg, Ph. D thesis June 2006.

[142] Rukhin A., Nechvatal J., Smid M., Barker E., Leigh S., Levenson M., Soto J., "A Statistical Test Suite for Random and Pseudorandom Number Generators for Cryptographic Applications," National Institute of Standards and Technology- NIST, Technical Report 2010.

[143] Hamano K., Yamamoto H., "A Randomness Test Based on T-Codes," in *International Symposium on Information Theory and Its Applications*, Auckland, 2008, pp. 361-373.

ANNEXURE– A

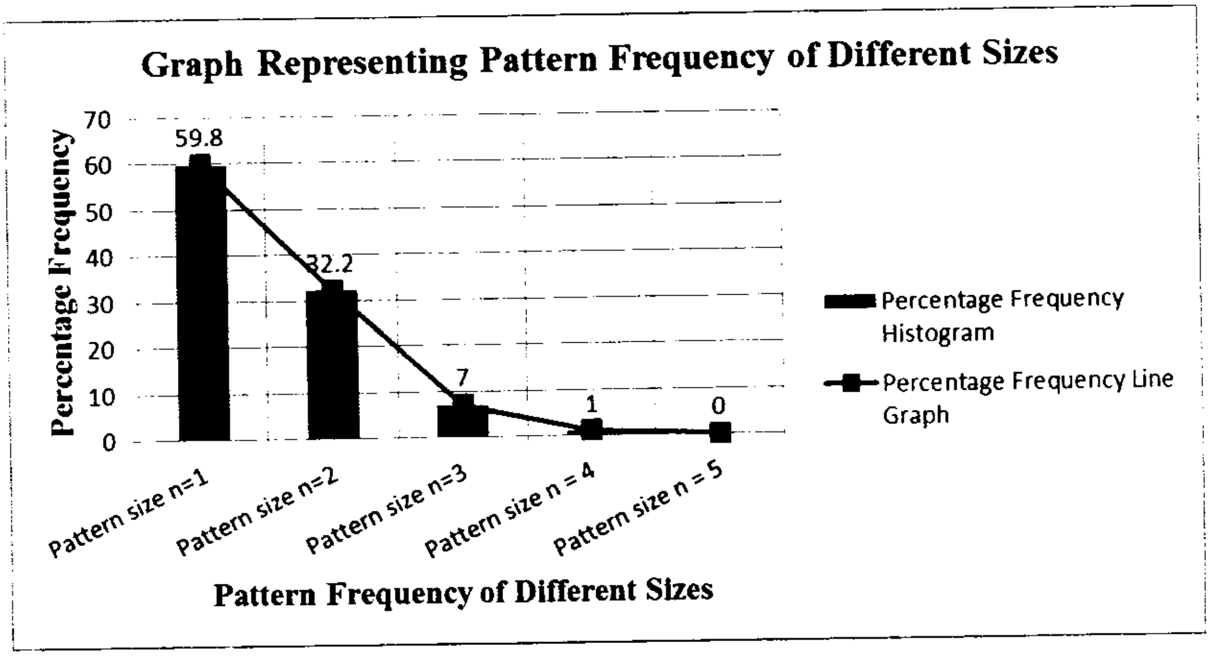
Phase I

Experiment-2

No. of Attacking Keys = 2000

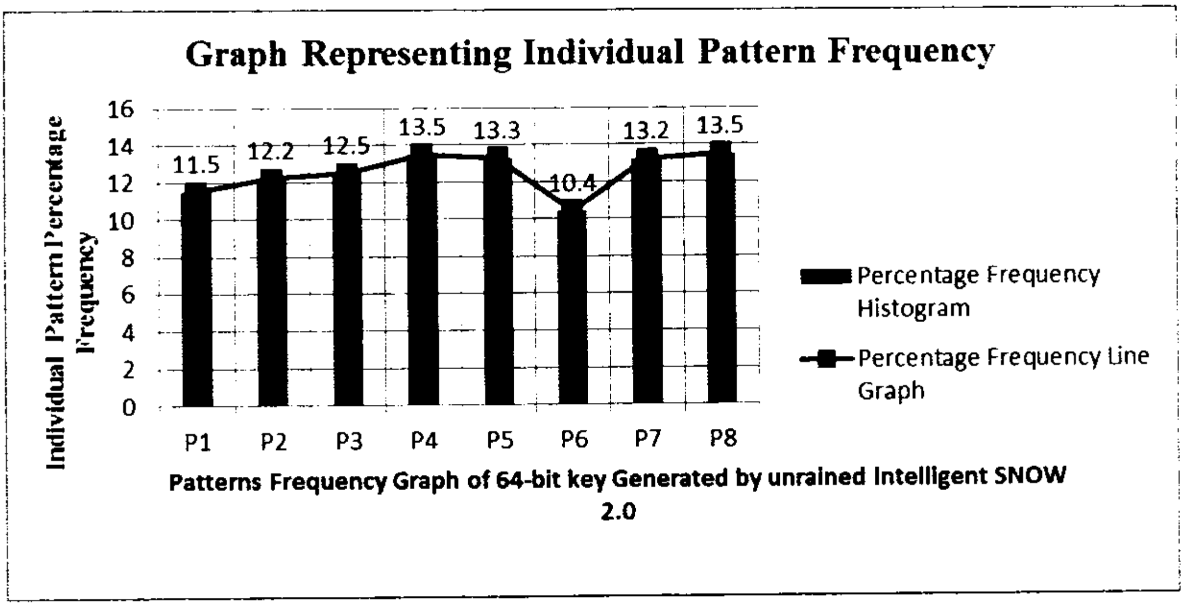
Determining Sub key Frequency

No. of Patterns (n) (1 Pattern= 8 bits)	Frequency	Percentage
Pattern size n=1	1195	59.8
Pattern size n=2	645	32.2
Pattern size n=3	140	7.0
Pattern size n = 4	19	1.0
Pattern size n = 5	1	.0
Total	**2000**	**100.0**



Determining Pattern Frequency

Pattern No.	Frequency	Percentage
P1	113	11.5
P2	120	12.2
P3	123	12.5
P4	133	13.5
P5	131	13.3
P6	103	10.4
P7	130	13.2
P8	133	13.5
Total	**986**	**100**

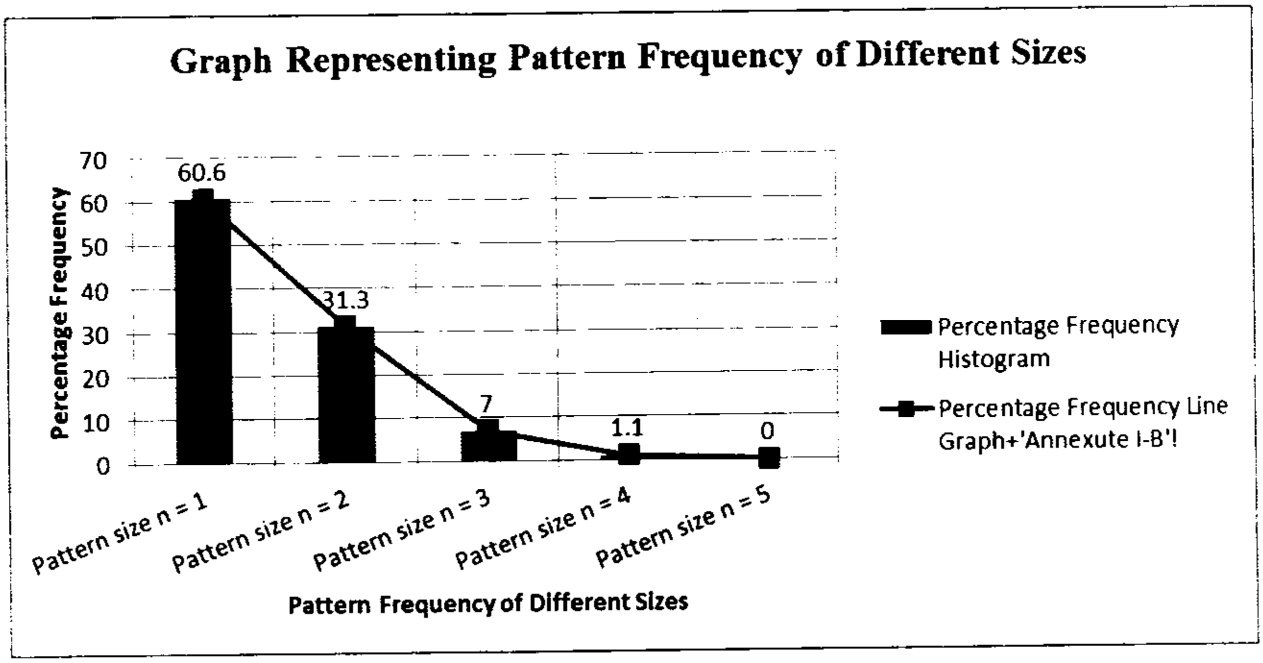


Experiment-3

No. of Attacking Keys = 2000

Determining Sub key Frequency

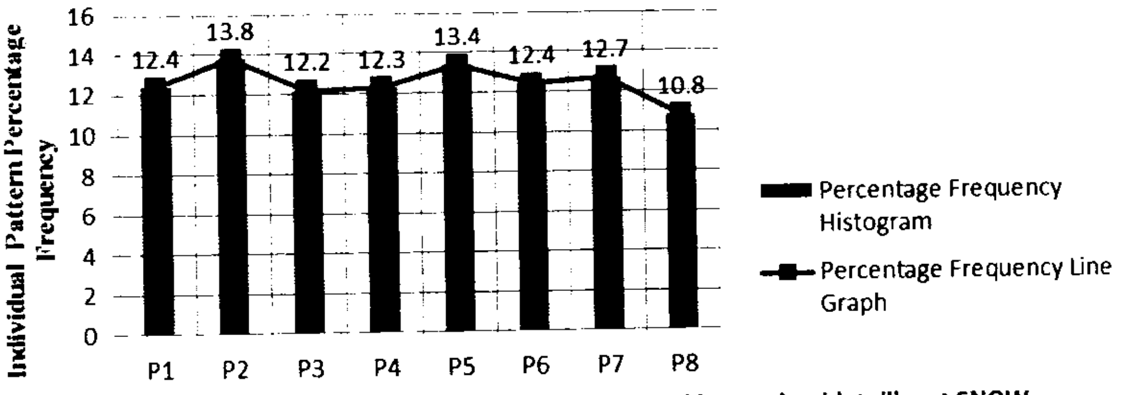
No. of Patterns (n)	**Frequency**	**Percentage**
Pattern size n=1	3031	60.6
Pattern size n=2	1567	31.3
Pattern size n=3	348	7.0
Pattern size n = 4	53	1.1
Pattern size n = 5	1	.0
Total	**5000**	**100.0**



Determining Pattern Frequency

Pattern No.	Frequency	Percentage
P1	301	12.4
P2	336	13.8
P3	296	12.2
P4	298	12.3
P5	326	13.4
P6	300	12.4
P7	307	12.7
P8	262	10.8
Total	**2426**	**100**

Graph Representing Individual Pattern Frequency



Patterns Frequency Graph of 64-bit key Generated by unrained Intelligent SNOW 2.0

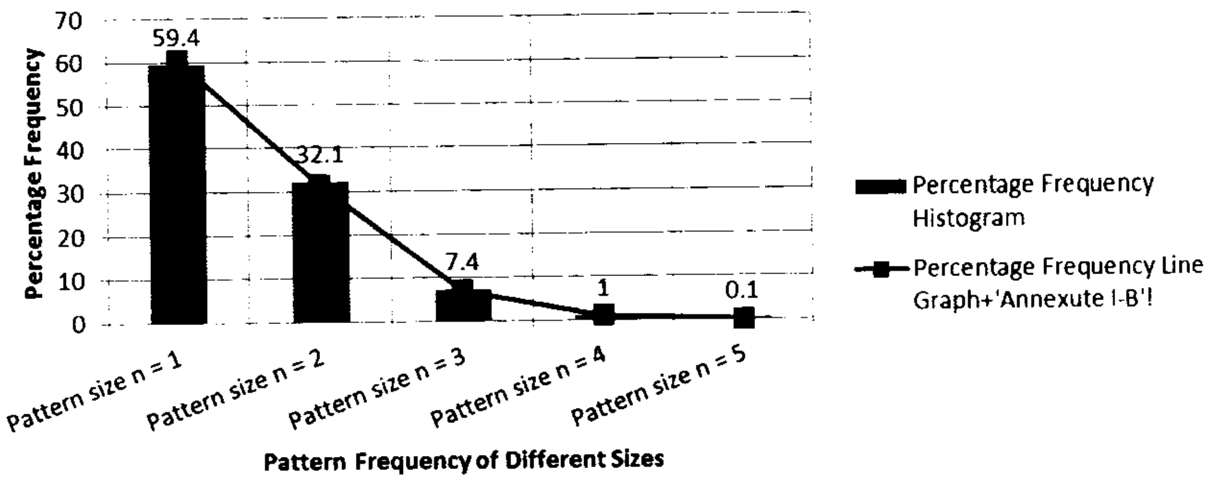
Experiment-3

No. of Attacking Keys = 10,000

Determining Sub key Frequency

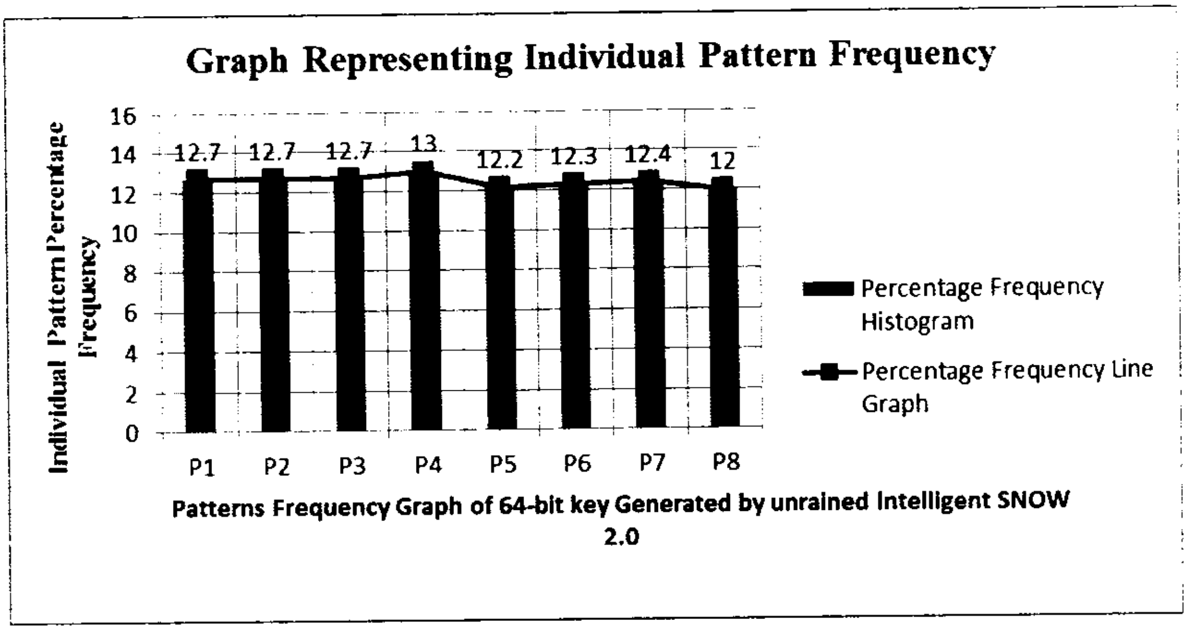
No. of Patterns (n)	Frequency	Percentage
Pattern size n = 1	5942	59.4
Pattern size n = 2	3208	32.1
Pattern size n = 3	744	7.4
Pattern size n = 4	100	1.0
Pattern size n = 5	6	.1
Total	**10000**	**100.0**

Graph Representing Pattern Frequency of Different Sizes



Determining Pattern Frequency

Pattern No.	Frequency	Percentage
P1	639	12.7
P2	640	12.7
P3	636	12.7
P4	654	13.0
P5	612	12.2
P6	615	12.3
P7	620	12.4
P8	604	12.0
Total	**5020**	**100.0**



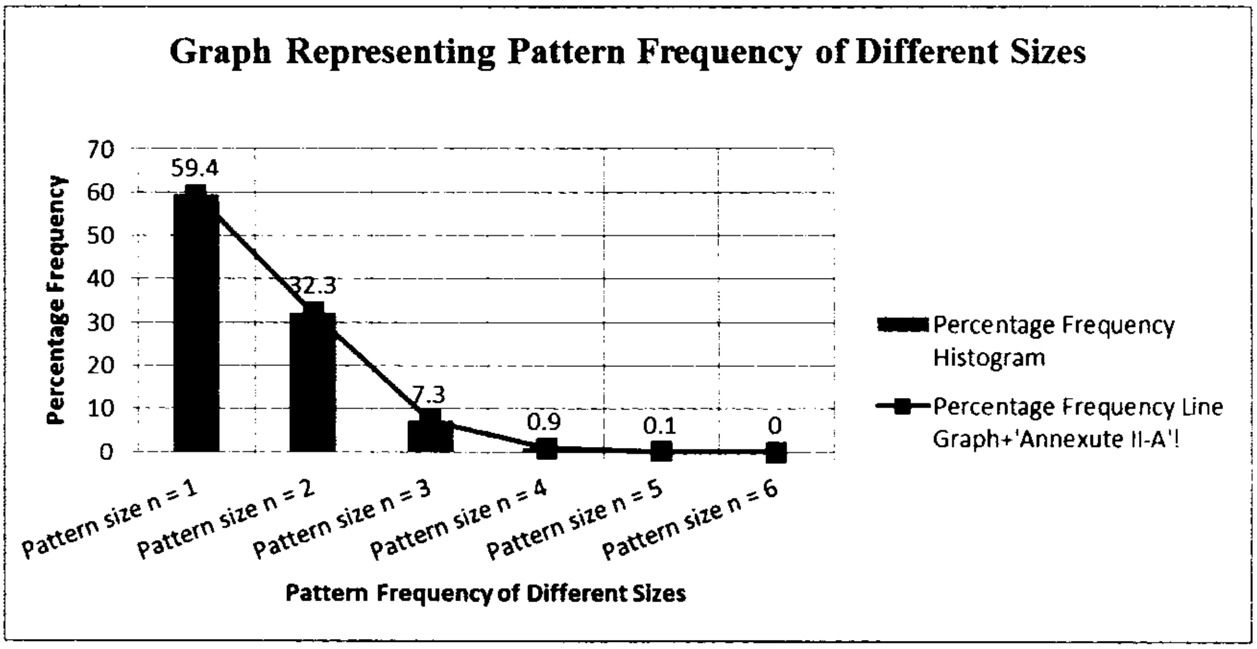
Phase II

Experiment-1

No. of Attacking Keys = 40000

Determining Sub key Frequency

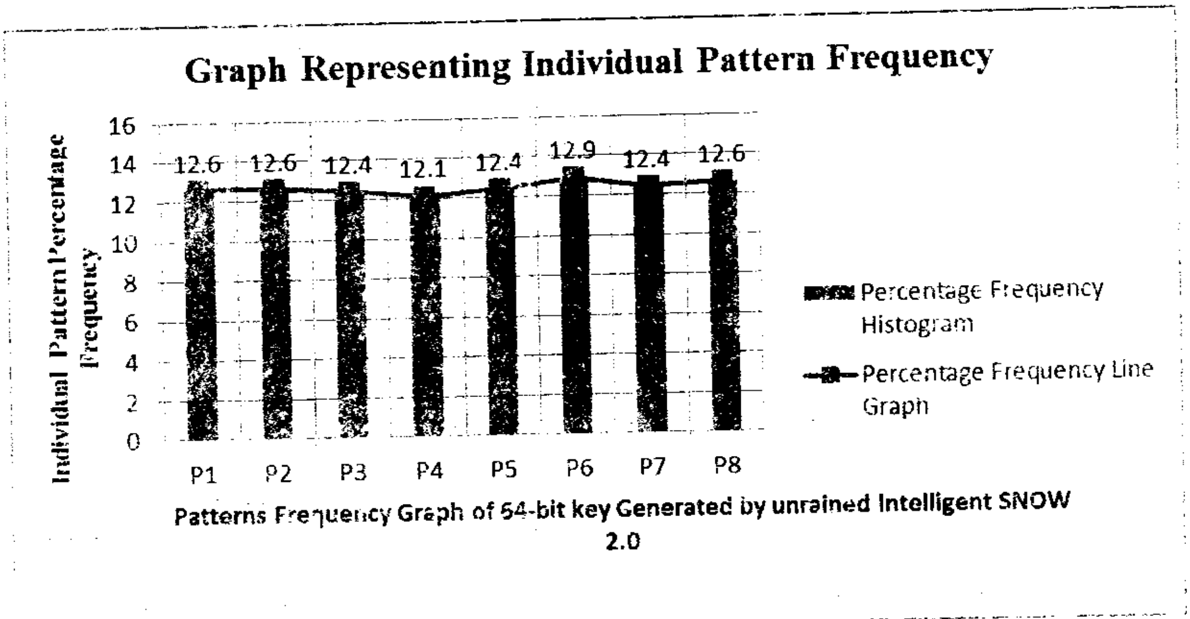
No. of Patterns (n)	Frequency	Percentage
Pattern size n = 1	23762	59.4
Pattern size n = 2	12924	32.3
Pattern size n = 3	2902	7.3
Pattern size n = 4	376	.9
Pattern size n = 5	31	.1
Pattern size n = 6	5	.0
Total	**40000**	**100.0**



Graph Representing Pattern Frequency of Different Sizes

Determining Pattern Frequency

Pattern No.	Frequency	Percentage
P1	2512	12.6
P2	2515	12.6
P3	2473	12.4
P4	2427	12.1
P5	2490	12.4
P6	2584	12.9
P7	2483	12.4
P8	2521	12.6
Total	**20005**	**100.0**

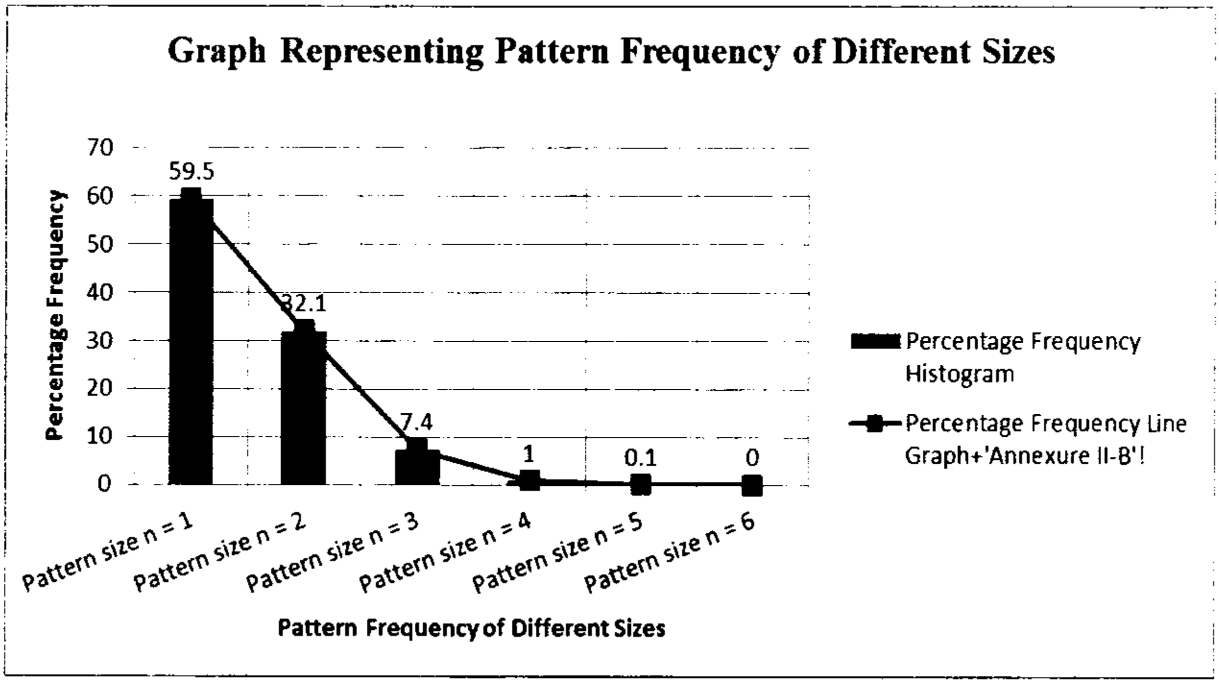


Experiment-2

No. of Attacking Keys = 60,000

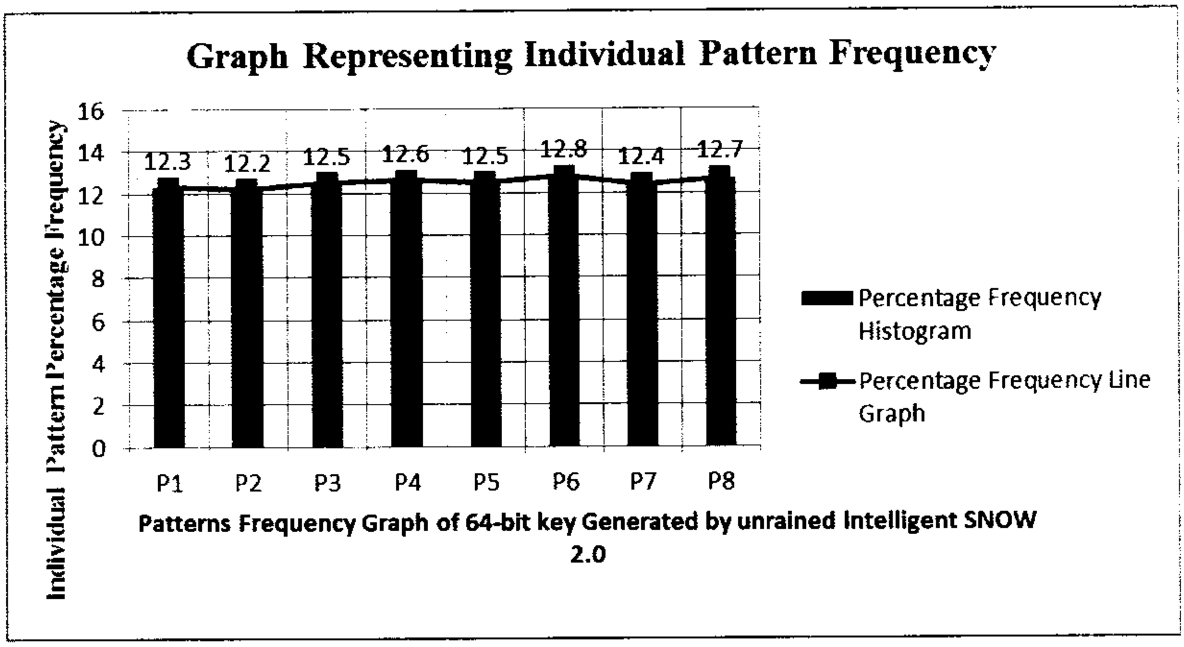
Determining Sub key Frequency

No. of Patterns (n)	Frequency	Percentage
Pattern size n = 1	35705	59.5
Pattern size n = 2	19232	32.1
Pattern size n = 3	4423	7.4
Pattern size n = 4	601	1.0
Pattern size n = 5	34	.1
Pattern size n = 6	5	.0
Total	**60000**	**100.0**



Determining Pattern Frequency

Pattern No.	Frequency	Percentage
P1	3686	12.3
P2	3675	12.2
P3	3756	12.5
P4	3772	12.6
P5	3767	12.5
P6	3851	12.8
P7	3728	12.4
P8	3807	12.7
Total	**30042**	**100.0**



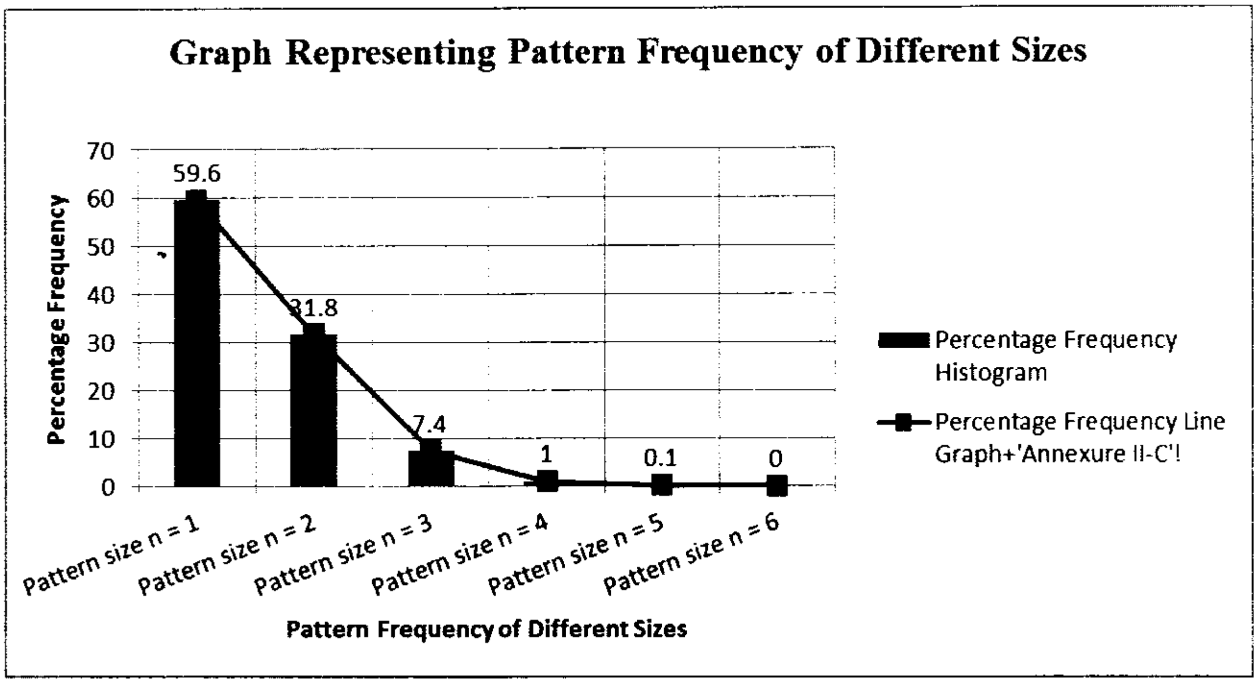
Patterns Frequency Graph of 64-bit key Generated by unrained Intelligent SNOW 2.0

Experiment-2

No. of Attacking Keys = 80,000

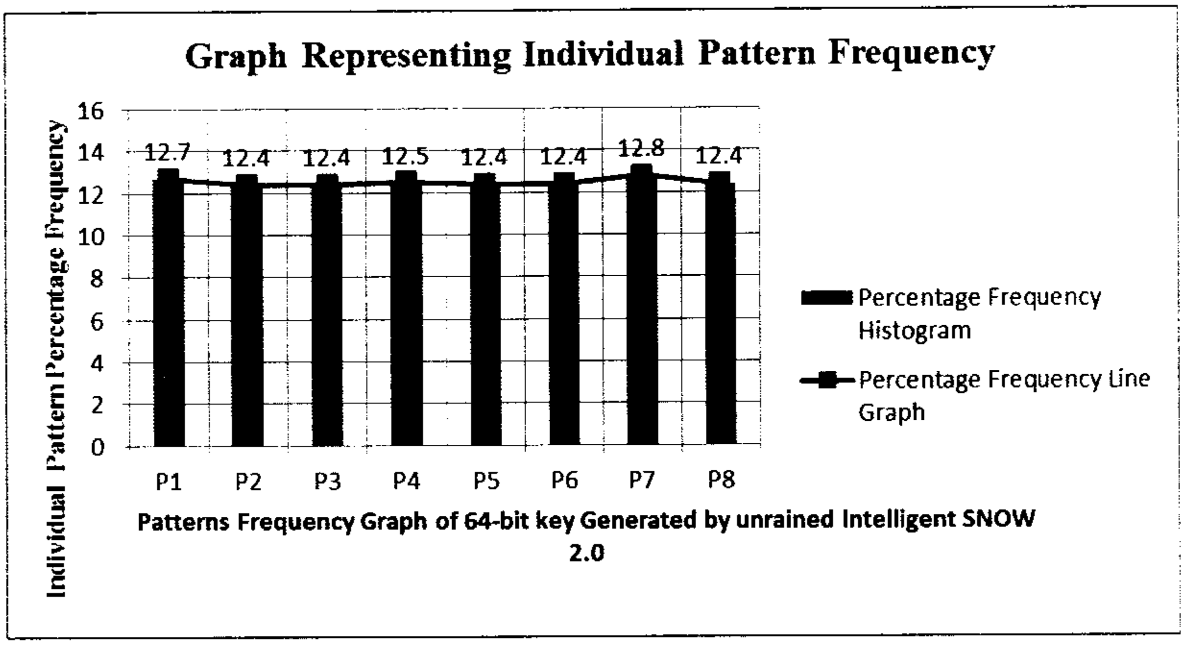
Determining Sub key Frequency

No. of Patterns (n)	Frequency	Percentage
Pattern size n = 1	47687	59.6
Pattern size n = 2	25478	31.8
Pattern size n = 3	5943	7.4
Pattern size n = 4	815	1.0
Pattern size n = 5	73	.1
Pattern size n = 6	4	.0
Total	**80000**	**100.0**



Determining Pattern Frequency

No. of Patterns (n)	Frequency	Percentage
P1	5111	12.7
P2	4987	12.4
P3	4969	12.4
P4	4998	12.5
P5	4958	12.4
P6	4988	12.4
P7	5153	12.8
P8	4957	12.4
Total	**40121**	**100.0**



ANNEXURE– B

PAPER- I

World Applied Sciences Journal 30 (4): 498-505, 2014
ISSN 1818-4952
© IDOSI Publications, 2014
DOI: 10.5829/idosi.wasj.2014.30.04.9

Intelligent Algorithm Design of the LFSR Based Stream Cipher

[1]Syed Irfan Ullah, [2]Muazzam A. Khan and [3]Malik Sikandar Hayat Khiyal

[1]Department of Computer Science, Faculty of Basic and Applied Sciences,
International Islamic University Islamabad, Pakistan
[2]EME College, NUST, H-11, Islamabad, Pakistan
[3]Preston University, H-8/1, Islamabad, Pakistan

Submitted: Nov 18, 2013; **Accepted:** Feb 3, 2014; **Published:** Feb 14, 2014

Abstract: Artificial Intelligence is one of the most prominent and dominant field of computer science. It is used in almost all area of computer fields. Very sophisticated intelligent algorithms have been developed and implemented in different applications and computer programming. We present our work to use it in byte oriented stream cipher algorithms, which are fast and comparatively less secure than those of the block ciphers, along with the intelligent algorithms so as achieve more secure algorithms along with the encryption processing speed, encryption algorithm, synchronization methods to improve the security of these comparatively less secure algorithms. in our work we propose a model based on Artificial Neural Networks to generate a pseudo random key intelligently for SNOW 2.0, so as to avoid the bit patterns traceable by the cryptanalysts using less computational power.

Key words: Stream Cipher · Linear FeedBack Shift Register · SNOW 2.0 · Modified SNOW 2.0 · Guess-and-Determine Attack · Artificial Neural Network

INTRODUCTION

Based on the neural structure of the brain, Artificial Neural Networks ANN are developed which are termed to be the crude electronic models. The human brain learns from the daily experience and once it is placed in the mind, is recalled whenever necessary or some situation related to that happens. There are a lot of problems in our daily life, which are beyond the scope of today's computers, because the computers makes decisions based on logics, already feed to it. This problem can be solved by using the artificial intelligence along with the logics those are feed to the current computers. This approach make it possible to solve complex problems with less overhead and provide a graceful degradation during system overload where very huge computations are involved in solving these problems.

The computers are best for keeping and maintaining records those are based upon some logics for example keeping and maintaining ledger and solving mathematical problems, but it faces trouble to maintain the patterns like humans and then to recognize those patterns and take some decision apart from the logics those are already feed to the system and take a proper decision at some accurate time.

Artificial Neural Network: It is a great mystery that how a human brain works, but some the aspects are explored, there are some cell in the human brain and spread in the entire body which jointly work, process and transmit information from one part of the body to another part of the body. These are the neurons about 100 billion in quantity, that makes the nervous system and makes a man able to think and makes the decisions [1].

Artificial Neural Network is an inspiration from the human nervous system that computes and behaves like human brain. Thousands of the neurons are spread over a parallel system, such that each individual neuron performs a very simple computation, such as xi = f(yi), where yi is a real valued input and the ith neuron output is xi and f is non linear function also called the node function. In ANN each neuron performs a specific

Corresponding Author: Syed Irfan Ullah, Department of Computer Science, Faculty of Basic & Applied Sciences, International Islamic University Islamabad, Pakistan.
E-mail: m.sikandarhayat@yahoo.com.

Fig. 1: Diagrammatic representation of SNOW 2.0

operation and accepts an input or a number of inputs which may be different from another neuron. The output of each neuron is a single value normally either 0 or 1.

Design and Structure of Stream cipher SNOW 2.0: Originally SNOW 1.0, which was modified to the new version SNOW 2.0. In the new version of SNOW:

- The word size in both versions of SNOW is same i.e. (32 bits) and
- Linear Feedback Shift Register (LFSR) length is again 16, but the feedback polynomial is different.
- The Finite State Machine (FSM) has two input words instead of one, taken from the LFSR i.e. Linear Feedback Sift Register and the pseudorandom key is generated by XOR between FSM output and the last entry of the (LFSR), as in first version of SNOW 1.0.

The operations of stream cipher are as follows:

- Key initialization is performed, which provides starting states to Linear Feedback Sift Register (LFSR) as well as to also give initial values to the internal Finite State Machine (FSM) registers R1 and R2.
- The operation of both the ciphers is slightly different, in SNOW 1.0, the model read the first symbol before ciphering was clocked after the initialization of the key but in SNOW 2.0, the first symbol was read out after the cipher is clocked once (Ekdahl and Johansson, 2002).
- In SNOW 2.0, two different elements involved in the feedback loop, i.e. α and α^{-1}.
- SNOW 2.0 takes two parameters as input value, the publicly known 128-bit initialization value IV and a secret key of either 128 or 256 bits.
- The IV, i.e. IV = (IV_3, IV_2, IV_1, IV_0) value is considered as a four word input, where IV_3 is the most significant one.
- The initial vector IV, possibly ranges from 0......2^{128}-1, means that

For a given key K, SNOW 2.0 implements a pseudorandom length increasing function from the set of IV values to the set of possible output sequences.

The new version is schematically a small modification of the original construction. Although SNOW 2.0 was appear to be more secure, but some weaknesses have been found in it [3], which results in proposal of Modified Version of SNOW 2.0 [4].

SNOW 2.0 is more secure, although it is slightly changed from SNOW 1.0, but still it has the cryptanalytic weakness addressed in [2-4, 7].

Linear Cryptanalysis of SNOW Family: Guess-and-Determine attack and correlation attacks are proven to be effective against word-oriented stream ciphers. In these attacks we initially Guess the contents of some of the cells and obtain the states of all the cells of the ciphering system and comparing the running key sequence with the resulting key. If the initial guess is proper one, then we may get that the sequences are same and otherwise we try another guess. The time complexity of the GD and exhaustive search is in the same order and depend on the guessed element, in spite of a long time devotion to Guess and Determine (GD) Attacks and correlation attacks on SNOW family stream ciphers improvements on word-oriented stream ciphers, they have often been implemented heuristically as on every new day some work is done on them to prove and justify the weaknesses in this family of stream ciphers [4].

This paper discusses the biological and mathematical design of the Artificial Neural Network in section 2, section 3 discusses the design and cryptanalysis of the ANN based stream cipher, section 4 discusses the Knowledge extraction from KB and back propagation of the Learning Algorithms, section 5 concludes the work done and its importance.

Biological and Computational Model of Artificial Neural Network: In this section we discuss the biological and computational model of the Artificial Neural Network:

Biological Model of ANN: Biological neurons are the foundation for the artificial neural networks, i.e. ANN functional model is made due to the inspiration from the biological neurons where they resemble in many ways. The nervous system contains neurons, the discrete cells and whose several processes arise from its cell body and generating and transmitting signals.

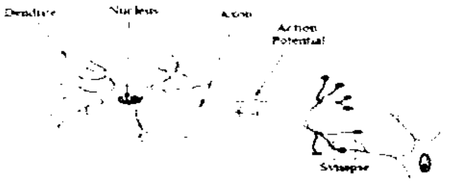


Fig. 2: General Structure of Biological Neuron

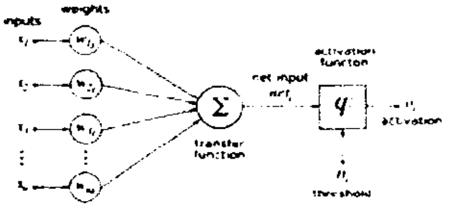


Fig. 3: Computational model of Artificial Neural Network.

The biological neuron has the following four regions i.e. the cell body also termed as soma which has two offshoots from it, the dendrites, the axon, which end in Presynaptic terminals.

All the four regions of the neurons has a specific task to perform and work as a unit[5], where it accept multiple inputs and output a single result, the functionality of the biological neurons is discussed briefly in the following points:

- The cell body or soma is the main part of the neuron cell, which contains protein synthesis and nucleus, which also works as storage and decision making body locally, such as reflex actions which are initiated as a result of these local decision makers inside the neuron.
- The dendrites which are the offshoots of the soma have branches in a treelike structure, which are used to receive signals from other neurons or the organs which senses the information from the outside.
- The axon that grows outside the cell body also termed as the hillock and conduct electric signals that grows from the cell body and ends at the presynaptic terminals. Only one axon grows from the cell which transfer signals to other neurons or brain. The action potential that carries the electric signals is identical to each other so the brains can easily judge, from the path it took that which type of signal has been received. The brain analyzes the signal patterns and issues orders where appropriate.
- The synapses are again the threads at the end of the axons which are connected to the dendrite of the other neurons where the information is passed from the synapse to the dendrites of the following neuron.

Computational Model of ANN: This model [6] accepts different inputs for training the learned algorithm those are kept in the database, which contain the rules for extracting the knowledge from the knowledge base, sensitivity of serious attacks, the stream ciphers and dozens of the weak keys, proved by the analysts.

$$y_k = \sum_{j=1}^{p} Wkj\ X_j$$

The activation function that gets output from the summing junction where the weight w_{kj} is multiplied with the x_j and are added together and are given as output to next neuron stored in the variable y_k

Knowledge Gain by Artificial Neural Networks: In Artificial Intelligence the Knowledge Acquisition is the bottleneck and most of the time is wasted in this phase. Because the system learn by induction, conduction analogy and require a long duration is this process. The expert systems those acquire knowledge in the form of rules or frames etc and processed whenever require. It is very difficult to formulate this knowledge and to understand the entire process. And thus in case of missing any of the steps will lead to erroneous knowledge gain by the system. In case of a proper acquirement of the knowledge that is stored in the knowledge base and gradually increase with process of experience and real world application and thus the decisions made by the system are more appropriate.

In order to get the experts knowledge into an expert system we propose to process these databases in the attempt to learn the particularities of the domain. Whatever is learned by this process can be discussed with the expert, who is now in the role of a supervisor and consultant that corrects and completes knowledge instead of a (often) unwilling teacher who has to express himself in some form he is not common and not comfortable with. Experts are required to describe their knowledge in form of symbolic rules, i.e. in a usually unfamiliar form. In particularly to describe knowledge acquired by experience is very difficult. Therefore KBS may not be able to diagnose cases that experts are able to. Some machine learning algorithms, for example ID3 [8], have the capability to learn from examples. We propose to use Artificial Neural Networks (ANN) as a first step of a machine learning algorithm. ANN claim to have advantages over these systems, being able to generalize and to handle inconsistent and noisy data. Interesting features of natural neural networks are their ability to build receptive fields in order to project the topology of the input space.

A high-dimensional input space is projected on a low dimensionality, usually a plane, conserving the topology of the input space. This is one of the advantages of unsupervised Neural Networks, like the Kohonen's Feature Map [9]: the internal structure of the ANN reflects structural features in the data without having any a-priori knowledge about their structure. However a good representation on this map has to be found in order to find the inherent structures of data, now represented on the map. The main idea therefore is to integrate both approaches, so that the advantages of ANN to generalize and to handle inconsistent and noisy data as well as to find the inherent structures in the data are combined with the ability of KBS to give explanations about the problem solving process using the rules of the knowledge base.

To realize the integration, an algorithm has to be constructed, that converts symbolic knowledge for the KBS out of the sub symbolic data of the ANN. In this work, we show how such a knowledge extraction was developed and tested on several data sets. Due to their inherent parallelism ANN are well suited to be mapped on massively parallel computer architectures like transputer clusters, which can be used as for hacking and breaking the security of the cryptographic algorithms. If enough knowledge is provided to these systems then, they can be used to improve the security of the system. For example in case of dictionary attacks where the system is enriched with the key words used in English and also with the keys those once used in past are avoided.

Our approach is different from this as it not only looking to the data dictionary with the exact match but it checks the probability to be broken down at run time so as to stop the generation and acceptance of the weak keys.

Intelligent Design of the Stream Cipher: To implement the biological neurons with stream ciphers it is necessary to define the working of the neural network with special perspective of the said area. For this purpose we first define the mathematical function of ANN and its different components.

Instance Representation by Attribute Value Pairs: The learning algorithm the educate the target function is defined over the instances, such that a vector of predefined features is described, such as past attacks those we successfully implanted, or the tendency towards the success, because in our case we do not only look at the intensity of fire, but the wound that may occur when a masquerade, tries to find out the bit patterns in the key. The input values can be any real values, which may be independent of one another or may highly correlate

The Output of the Target Function May Be a Vector of Several Real or Discrete-valued, Real Valued or Discrete Valued Attributes: If the output is a real number then it will range between 0 and 1, which in this case corresponds to the confidence in predicting the corresponding steering direction. We can also train a single network to output both the steering command and suggested acceleration, simply by concatenating the vectors that encode these two output predictions.

Errors May Occur in the Training Examples: If there is a noisy data with errors in training examples, then it depends on the learning methods that how they respond to these errors. In the other applications of the ANN, it is kept in mind that the training data must not contain errors, but in the ciphering algorithms we permit to larger instance, because here we are interested to maintain a training base and not in the quality of data.

Long Training Times Are Acceptable in Case of an Isolated System: To fully train an artificial neural network training algorithms requires longer times ranging from a few seconds to many hours and even days and weeks, depending on the factors such as the settings of various learning algorithm parameters, the number of training examples considered, the number of weights in the network. Keeping in view that in public area where the outer world has the access to the system, longer time is never permissible and in such a case the point iii, may be applicable for the time being.

The Learned Target Is Required to Be Evaluated with Fast Speed: Comparatively long time is required to train the ANN learning algorithms, but to evaluate the learned network, in order to apply a subsequent instance is very fast and the instantaneous action is taken while transferring data upon the network, whenever and attack is detected.

The Learned Target Function Need Not to Be Understood by the Humans: It is very difficult for the humans to clearly understand the complex nature of the weights learned by the ANN and is very difficult to interpret for them. The learned rules can easily be communicated to the learning rules, but it is very difficult to communicate it to

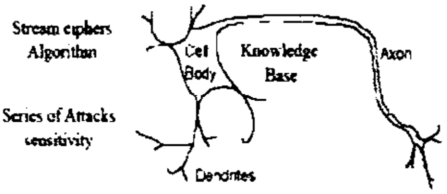


Fig. 3: Biological Neuron model with Stream Cipher

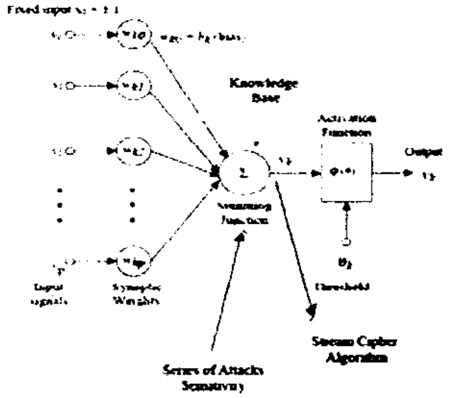


Fig. 4: Computational model of neural network for Stream cipher

human beings. So, the learned rules are made in such a way that if the data store, contain some data for what the rules have already been defined, then a human may ignore it, but the processing device cannot ignore it, which may cause to increase the security, i.e. to avoid the similar bit patterns in the key stream, produced by ANN pseudo-random number generator.

Mathematical Design of ANN for Stream Ciphers: The mathematical model of this model can be seen in Figure (4). This model accepts different inputs for training the learned algorithm those are kept in the database, which contain the rules for extracting the knowledge from the knowledge base, sensitivity of serious attacks, the stream ciphers and dozens of the weak keys, proved by the analysts.

$$y_k = \sum_{j=1}^{p} Wkj\, Xj$$

The activation function that gets output from the summing junction where the weight w_{kn} is multiplied with the x_i and are added together and are given as output to next neuron stored in the variable y_k.

To the summation function the information from the knowledge base, series of attack sensitivity and other rules and frames are provided, where based on these rules and frames the resultant vector is given to the activation which ultimately decides as the key generated is appropriate at this stage or a rejection is implanted and the search for another strong key is started.

Activation Function of the ANN, with the General Stream Ciphers: In the neural networks the neurons output is between 0 and1 or -1 and 1, usually. The activation function is denoted by Φ(.) and is of three types [6]:

- If the summed input is smaller than the threshold value, then threshold function taken on a value of 0 and if it is greater or equal to the threshold value then it is taken as 1.

$$\Phi(v) = \begin{cases} 1 & if\ v \geq 0 \\ 1 & if\ v < 0 \end{cases}$$

- The piecewise-linear function, taken on the values of 0 or 1 and can be taken on the amplification factor in a certain region of linear operation.

$$\Phi(v) = \begin{cases} 1 & v \geq \frac{1}{2} \\ v & -\frac{1}{2} > v > \frac{1}{2} \\ 0 & v \leq \frac{1}{2} \end{cases}$$

Success in Pattern Recognition: The pattern recognition in the field of cryptography is based upon the statistical tests, where the Guess and determine attack [3, 7] and correlation attacks find the correlation between the actual key stream and that of the best matching key and to apply exhaustive search for finding the exact key or the best one. The correlation attacks and the Guess and determine attacks those are based on the algebraic rules have been proven to be more efficient against stream ciphers. The key generated by the pseudo random numbers generators is used only once and for the second use we need to generate another key.

The cryptanalyst have find out the techniques that when the new key generated by the LFSR based stream ciphers is traceable and the algebraic attacks are more successful. One alternative is to use NLFSR based keystream generators but it has its own implications and drawbacks and the other approach that we propose is to use Artificial Neural Networks along with the LFSR based pseudo-random key stream generators. The ANN is proved to be the best suited intelligent algorithm for cryptography, to avoid the traceability in the patters of the keystream of any of the algorithm specifically the SNOW 2.0. The process looks for the 0, 1, 00, 11, 000, 111, 01, 001, 10, 110 as assumed in the Guess and Determine attack on the stream ciphers, SNOW and make a hypothesis that the pattern at the lowest is been distinguished, in the universal approximation theorem.

Application of the Universal Approximation Theorem: The universal approximation theorem states that, "The simplest form of multilayer perceptron, a single hidden layer, in a feed forward network, containing a finite number of neurons is a universal approximator among continuous functions on the reduced form of Rn, where the activations functions are mould"[8].

The mathematical form of the Universal approximation theorem is as under:

Consider a bounded, non-constant and monotonically increasing continuous function $\varphi(\cdot)$,

Let I_m denote the *m*-dimensional unit hypercube $[0,1]^m$. The space of continuous functions on I_m is denoted by $C(I_m)$. Then, given any function $f \in C(I_m)$ and $\varepsilon > 0$, there exist an integer N and real constants $\alpha_i, b_i \in \mathbf{R}$, $w_i \in \mathbf{R}^m$, where $i = 1, \ldots, N$ such that we may define:

$$F(x) = \sum_{i=1}^{N} \alpha_i \varphi(w_i^T x + b_i)$$

as an approximate realization of the function f where f is independent of Φ; that is,

$$|F(x) - f(x)| < \varepsilon$$

for all $x \in I_m$. In other words, functions of the form $F(x)$ are dense in $C(I_m)$.

we apply the linear approximation theorem on SNOW 2.0, for the approximation of the Addition Modulo 2^n, so as to find the approximation to the probability of any of the keystream with some specific bit patterns.

Knowledge Extraction from KB and Back Propagation of the Learning Algorithms: The strategy adopted for the implementation if the artificial neural networks with the SNOW 2.0, we design a special environment so that the ANN and SNOW 2.0 both are kept in such a way that when a new key is established, it is compared with the Knowledge base of the neural network and if there exist bit pattern which may be vulnerable to the attacker, then that key is rejected, for this purpose. The key comparison is performed by using the following algorithm, which lies in the worst case of 2^n.

Attack Based on the Knowledge Base and Pattern Matching Algorithm:

```
KE_match(key, KB)

•   Set IV – input through keyboard
•   Key = snow2 ( IV )
•   Psize = 1,2,3,...., n (Pattern size, where initially 1 bit, 2 bits and so forth)
•   vcount –      0       (Initialize j to first instance in the dictionary)
•   j – 0
•   hcount – 0    (initialize to first bit in the key stream of dictionary and Original Key)
    •  Repeat step ii to vi n times
    •  if (key(i) = = dic_key(i,j))
    •  put 1 to file
    •  hcount – hcount++ (move pointer to next bit)
    •  else
    •  put 0 to file

•   j++
•   Vcount – vcount++(move pointer to next key in dictionary)
•   Repeat step 5 to 7 till end of file
•   End of function
```

If the knowledge base size is M and n is the key size, then the time complexity of the algorithm to compare all the patterns is illustrated in section 4.2

Time Complexity of the Pattern Matching Algorithm: If there is a finite set of n elements for s, then $|S| = n$, the subsets of this set S will be $|P(S)| = 2^n$, which is the motivation notation 2^A, is described as follows:

Let ωi, such that $1 \leq i \leq n$ and the subsets of S are $\{\omega 1, \omega 2, \omega 3, \ldots\ldots, \omega n\}$, where ωi can take values of 0 or 1.

If $\omega i = 1$, then any element lies at ith position will be the element in the subset and otherwiae it will not be the element of the subset. All the elements in the subset are distinct and non overlapping and will lie under the $O(2^n)$.

If m is the size of the Knowledge base then the time complexity for the above algorithm will be $M * 2^n$ and lies in the $O(2^n)$, in the worst case, which is too high to reduce the performance of the key generation of SNOW 2.0.

So. to minimize the time and computational complexity. we adopt an approach that instead of searching the each individual bit pattern. we make a group of 8 bits and look to these 8 bit jointly. which ultimately increase the performance of the algorithm, while still existing in the $O(2^n)$. but in this case n will be n/8, so. if the key size is 64 bits, then 64/8 = 8 and in case of 128 bit key size 128/8 = 16. so. the performance will improved. while using this algorithm but still existing in the $O(2^n)$. worst case time complexity. The bit patterns. while comparing each individual bit, is strictly discouraged. because the focus in this work is upon the bit patterns, so if 8 consecutive bits are similar. then we say that one pattern is similar. while rest of the 7 blocks. in the 64 bit (8 block). still require to be deduced. so if 8 consecutive bits are similar. then we say that one pattern is similar. while rest of the 7 blocks in the 64 bit(8 blocks). still require to be deduced. so if a series of 8-bit blocks consecutive to each other are targeted by the comparison algorithm. in response to the neural networks. then that key is treated as weak key and must be rejected.

To minimize the time complexity of the above approach we propose mining without minimum support threshold is the best suited approach for finding frequent patterns in the dataset [10]. "The prominent advantage of this approach is that it does not require user specified *min-sup* threshold and performs single database scan only. This technique can be applied to retrieve the *top most* and *top-k* maximal frequent item sets. In case of mining more than one maximal Frequent item set, the algorithm requires a more easily understandable parameter *k*, which refers to the desired number of maximal frequent item sets".

Frequent Patterns in Keystream and Knowledge Extraction from KB: The following algorithm will check the key strength by comparing with the knowledgebase in the ANN, to get the adjacent patterns in the key:

```
KEY_check(key, KB)

•   Set IV – input through keyboard
•   do
•   {
•   Key = snow2 ( IV )
•   for ( i=0; i<=n, i++)
•   {
•   for (j = i; j<=n; j++)
•   {
•   Vcount – 0
•   For(k = i; k<=j; k++;
•   {
       • repeat step ii to vi n times
       • if (keypattern(i) = = dic_key(i))
       • put 1 to datafile
       • else
       • put 0 to datafile

•   vcount – vcount++(move pointer to next key in
    dictionary)
•   }
•   }
•   }
•   While(!key-not-approved)
•   End of function
```

The algorithm works as follows:

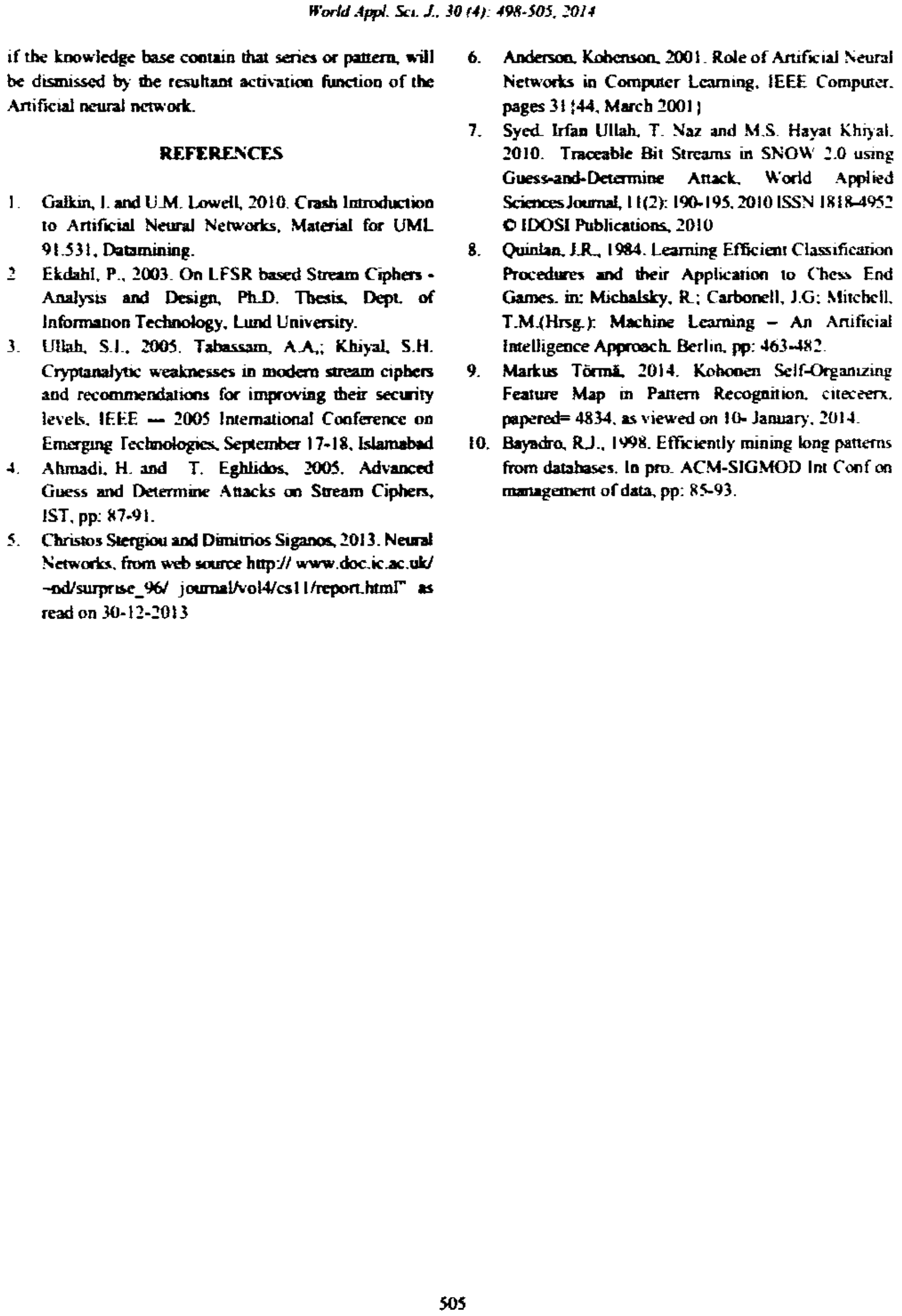
Table. Neural net setup keys from decision making

Index	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Value	v0	v1	v2	v3	v4	v5	v6	v7	v8	v9	v10	v11	v12	v13	v14	v15

If value at the index {0}, is taken and compared with the knowledge base and the value at the index {1,2} is taken and compared with the knowledge base and this process continues for values at the index {1,2,3},......,{1,2,3.....15}, in the first iteration, in the second iteration the value at the index {2},{2,3} and so forth, wherever the key match is successful unto some threshold level, the key is rejected and the algorithm generates the next key and the process continues until and unless a secure key is found. To find out the time complexity of the algorithm we first define the parameter used in the algorithm. n is the numbers of bytes in key stream and m is the size if the knowledge base. Then the worst case time complexity of the algorithm is m*(n(n+1)/2)

CONCLUSION

The model we present is more secure in a sense, that scanning the knowledge base it does not output the weak cryptographic keys and also the time complexity of the algorithm is too high, that any of known attack cannot succeed, because of the natural strength of the SNOW 2.0 and also if there is a probability of generating weak key.

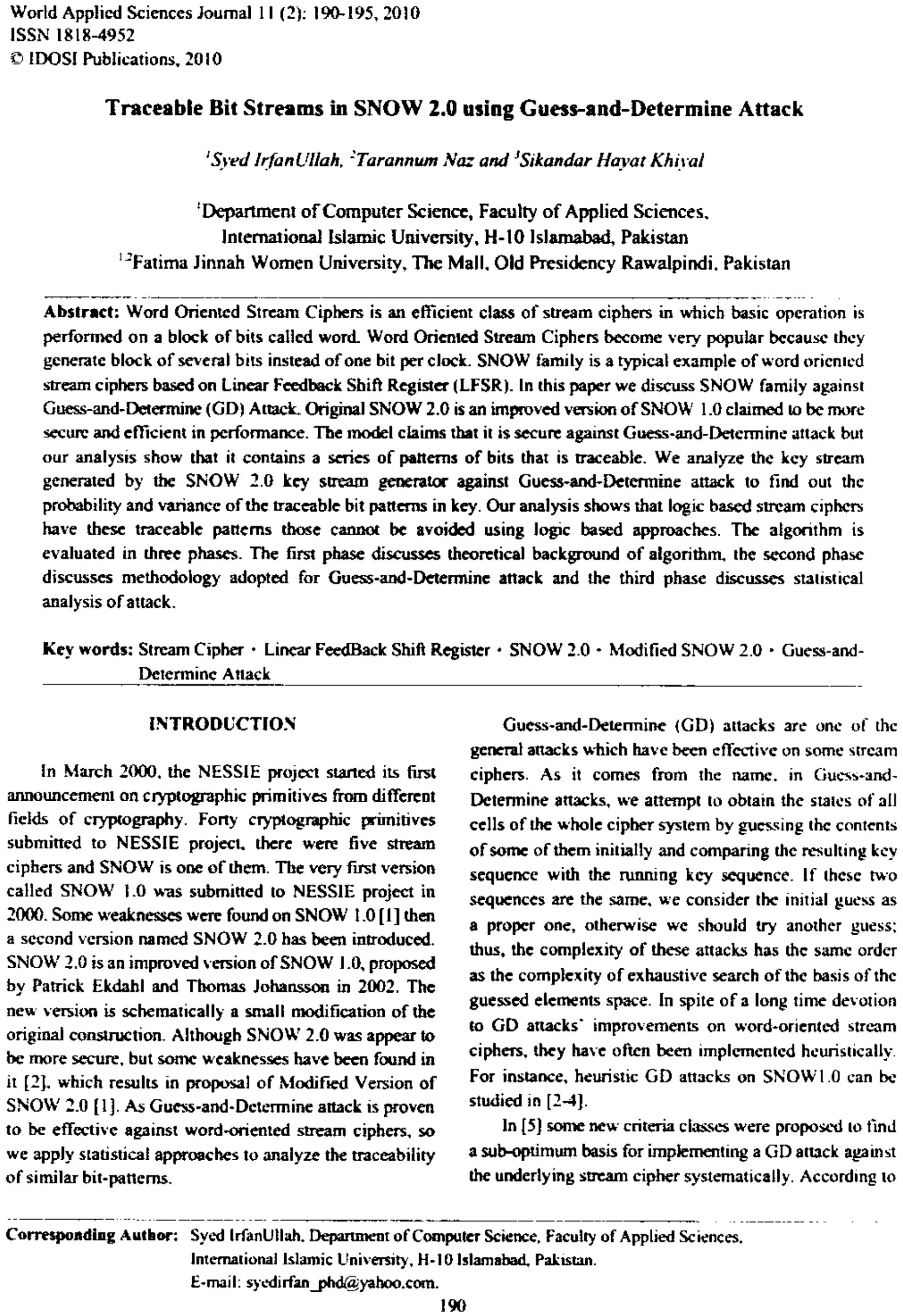


if the knowledge base contain that series or pattern, will be dismissed by the resultant activation function of the Artificial neural network.

REFERENCES

1. Galkin, I. and U.M. Lowell, 2010. Crash Introduction to Artificial Neural Networks, Material for UML 91.531, Datamining.
2. Ekdahl, P., 2003. On LFSR based Stream Ciphers - Analysis and Design, Ph.D. Thesis, Dept. of Information Technology, Lund University.
3. Ullah, S.I., 2005. Tabassam, A.A,; Khiyal, S.H. Cryptanalytic weaknesses in modern stream ciphers and recommendations for improving their security levels, IEEE — 2005 International Conference on Emerging Technologies, September 17-18, Islamabad
4. Ahmadi, H. and T. Eghlidos, 2005. Advanced Guess and Determine Attacks on Stream Ciphers, IST, pp: 87-91.
5. Christos Stergiou and Dimitrios Siganos, 2013. Neural Networks, from web source http:// www.doc.ic.ac.uk/ ~nd/surprise_96/ journal/vol4/cs11/report.html" as read on 30-12-2013
6. Anderson, Kohenson, 2001. Role of Artificial Neural Networks in Computer Learning, IEEE Computer, pages 31{44, March 2001}
7. Syed Irfan Ullah, T. Naz and M.S. Hayat Khiyal. 2010. Traceable Bit Streams in SNOW 2.0 using Guess-and-Determine Attack, World Applied Sciences Journal, 11(2): 190-195, 2010 ISSN 1818-4952 © IDOSI Publications, 2010
8. Quinlan, J.R., 1984. Learning Efficient Classification Procedures and their Application to Chess End Games. in: Michalsky, R.; Carbonell, J.G; Mitchell, T.M.(Hrsg.): Machine Learning – An Artificial Intelligence Approach. Berlin, pp: 463-482.
9. Markus Törmä, 2014. Kohonen Self-Organizing Feature Map in Pattern Recognition, citeceerx, papered= 4834, as viewed on 10- January, 2014.
10. Bayadro, R.J., 1998. Efficiently mining long patterns from databases. In pro. ACM-SIGMOD Int Conf on management of data, pp: 85-93.

PAPER- II



World Applied Sciences Journal 11 (2): 190-195, 2010
ISSN 1818-4952
© IDOSI Publications, 2010

Traceable Bit Streams in SNOW 2.0 using Guess-and-Determine Attack

[1]*Syed IrfanUllah*, [2]*Tarannum Naz and* [3]*Sikandar Hayat Khiyal*

[1]Department of Computer Science, Faculty of Applied Sciences,
International Islamic University, H-10 Islamabad, Pakistan
[1,2]Fatima Jinnah Women University, The Mall, Old Presidency Rawalpindi, Pakistan

Abstract: Word Oriented Stream Ciphers is an efficient class of stream ciphers in which basic operation is performed on a block of bits called word. Word Oriented Stream Ciphers become very popular because they generate block of several bits instead of one bit per clock. SNOW family is a typical example of word oriented stream ciphers based on Linear Feedback Shift Register (LFSR). In this paper we discuss SNOW family against Guess-and-Determine (GD) Attack. Original SNOW 2.0 is an improved version of SNOW 1.0 claimed to be more secure and efficient in performance. The model claims that it is secure against Guess-and-Determine attack but our analysis show that it contains a series of patterns of bits that is traceable. We analyze the key stream generated by the SNOW 2.0 key stream generator against Guess-and-Determine attack to find out the probability and variance of the traceable bit patterns in key. Our analysis shows that logic based stream ciphers have these traceable patterns those cannot be avoided using logic based approaches. The algorithm is evaluated in three phases. The first phase discusses theoretical background of algorithm, the second phase discusses methodology adopted for Guess-and-Determine attack and the third phase discusses statistical analysis of attack.

Key words: Stream Cipher • Linear FeedBack Shift Register • SNOW 2.0 • Modified SNOW 2.0 • Guess-and-Determine Attack

INTRODUCTION

In March 2000, the NESSIE project started its first announcement on cryptographic primitives from different fields of cryptography. Forty cryptographic primitives submitted to NESSIE project, there were five stream ciphers and SNOW is one of them. The very first version called SNOW 1.0 was submitted to NESSIE project in 2000. Some weaknesses were found on SNOW 1.0 [1] then a second version named SNOW 2.0 has been introduced. SNOW 2.0 is an improved version of SNOW 1.0, proposed by Patrick Ekdahl and Thomas Johansson in 2002. The new version is schematically a small modification of the original construction. Although SNOW 2.0 was appear to be more secure, but some weaknesses have been found in it [2], which results in proposal of Modified Version of SNOW 2.0 [1]. As Guess-and-Determine attack is proven to be effective against word-oriented stream ciphers, so we apply statistical approaches to analyze the traceability of similar bit-patterns.

Guess-and-Determine (GD) attacks are one of the general attacks which have been effective on some stream ciphers. As it comes from the name, in Guess-and-Determine attacks, we attempt to obtain the states of all cells of the whole cipher system by guessing the contents of some of them initially and comparing the resulting key sequence with the running key sequence. If these two sequences are the same, we consider the initial guess as a proper one, otherwise we should try another guess; thus, the complexity of these attacks has the same order as the complexity of exhaustive search of the basis of the guessed elements space. In spite of a long time devotion to GD attacks' improvements on word-oriented stream ciphers, they have often been implemented heuristically. For instance, heuristic GD attacks on SNOW1.0 can be studied in [2-4].

In [5] some new criteria classes were proposed to find a sub-optimum basis for implementing a GD attack against the underlying stream cipher systematically. According to

Corresponding Author: Syed IrfanUllah, Department of Computer Science, Faculty of Applied Sciences, International Islamic University, H-10 Islamabad, Pakistan.
E-mail: syedirfan_phd@yahoo.com.

these criteria classes a new algorithm, called *Chess Algorithm*, was introduced. Based on the novel idea [5], the Advanced GD attacks were introduced [6], which lead to the improvement of many previously heuristic GD attacks on some stream ciphers. For example, the computational complexities of the attacks on SNOW1.0 and SNOW2.0 were reduced from $O(2^{224})$ and $O(2^{264})$ to $O(2^{205})$ and $O(2^{267})$ respectively. It is worth mentioning that there are also better attacks on these ciphers, e.g., a distinguishing attack on SNOW2.0 with computational complexity of $O(2^{202})$ has recently been proposed in [7] which is the best published one so far [8].

SNOW 1.0 was captured by GD attack in a way that, Finite State Machine (FSM) has only one input function s(1) to the FSM. It enables an attacker to invert the operations in the FSM to derive more unknowns from only a few guesses [1]. SNOW 2.0 was captured by correlation attack.

Description of SNOW 2.0: The word size of SNOW 2.0 remains same (32 bits) as of SNOW 1.0 and length of LFSR is again 16, but the feedback polynomial is different. The Finite State Machine (FSM) has two input words instead of one, taken from the LFSR and the running key is generated by XOR between FSM output and the last entry of the LFSR, as in SNOW 1.0

The operations of cipher are as follows, first of all key initialization is performed. This operation provides starting states to LFSR and to give initial values to the internal FSM registers R1 and R2. There is a small difference in the operation of the cipher. In the first version, after the key initialization, the first symbol was read out before the cipher was clocked but in second version it is read out after the cipher is clocked once [9].

In SNOW 2.0, two different elements involved in the feedback loop, α and α^{-1}. SNOW 2.0 takes two parameters as input value, a secret key of either 128 or 256 bits and a publicly known 128-bit initialization value IV. The IV value is considered as a four word input IV = (IV_3, IV_2, IV_1, IV_0) where IV_0 is the least significant one. The possible range for IV is thus 0......2^{128}-1. This means that for a given key K, SNOW 2.0 implements a pseudorandom length increasing function from the set of IV values to the set of possible output sequences.

Key Initialization Process: SNOW 2.0 takes two parameters as input values; a secret key of either 128 or 256 bits and a publicly known 128 bit initialization variable IV. The IV value is considered as a four word input IV = (IV_3, IV_2, IV_1, IV_0), where IV_0 is the least significant word. The possible range for IV is thus 0...2^{128} – 1. This means that for a given secret key K, SNOW 2.0 implements a pseudo-random length-increasing function from the set of IV values to the set of possible output sequences.

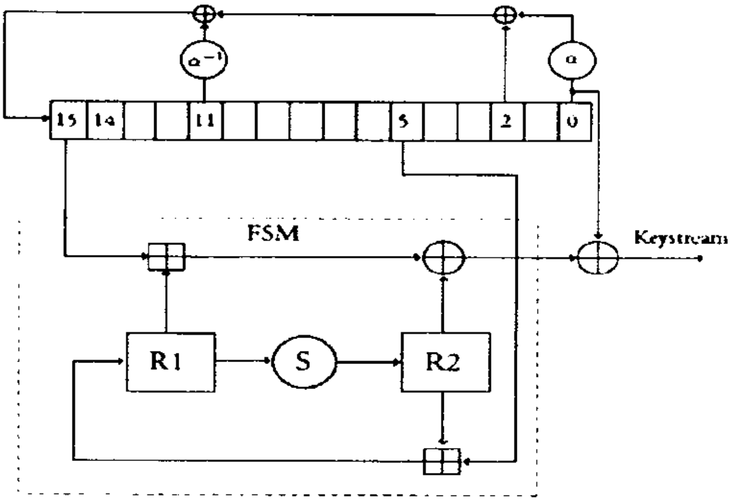


Fig. 1: A schematic model of SNOW 2.0

The model of SNOW 2.0 is shown in Figure 1, where we denote addition in F_2^{32} by the symbol ⊕, addition modulo 2^{32} by the symbol ⊞, multiplication with α by ⊙ and S-Box operation is denoted by ⓢ.

The key initialization is done as follows. Denote the registers in the LFSR by (S_{15}, S_{14}, ..., S_0) from left to right in Figure 1.

Thus S_{15} corresponds to the element holding S_{t+15} during normal operation of the cipher. Let the secret key be denoted by K= (K_3, K_2, K_1, K_0) in the 128 bit case and K = (K_7, K_6, K_5, K_4, K_3, K_2, K_1, K_0) where each K_i is a word and K_0 is the least significant word [9]. First, the shift register is initialized with K and IV as follows,

s15 = k3⊕IV0, s14 = k2, s13 = k1, s12 = k0 ⊕ IV1, s11 = k3 ⊕ 1, s10 = k2⊕IV2, s9 = k1⊕1⊕IV3, s8 = k0⊕ 1,
and for the second half of the register,
s7 = k3, s6 = k2, s5 = k1, s4 = k0, s3 = k3 ⊕ 1,
s2 = k2 ⊕ 1, s1 = k1 ⊕ 1, s0 = k0 ⊕ 1,

where 1 denotes the all one vector (32 bits).

After the LFSR has been initialized, R1 and R2 are both set to Zero. Now the cipher is clocked 32 times without producing any output symbol. Instead, the output of the FSM is incorporated in the feedback loop as shown in Figure 2. Thus during the 32 clocks in the key initialization, the next element to be inserted into the LFSR is given by $S_{t+16}=\alpha^{-1} S_{t+11} \oplus S_{t+2} \oplus \alpha S_t \oplus F_t$ [9].

After the 32 clockings the cipher shifted back to normal operation as shown in Figure 1 and is clocked once before the first keystream symbol is produced [9].

Design of Dictionary Through Guess-and-determine Attack: After the designing of SNOW 2.0, the next step is to design an attack for SNOW 2.0. Reliability of SNOW 2.0 has to be checked against Guess-and-Determine (GD) Attack, due to that reason Guess-and-Determine (GD) Attack has been designed. It is designed in a way that guess has been made on secret key and initialization values and determines running keystream on basis of these guesses. Guess-and-Determine (GD) Attacks can be considered as general attacks on stream ciphers. In the proposed system the Guess-and-Determine (GD) Attack is used to guess the secret key and initialization values. It is described below.

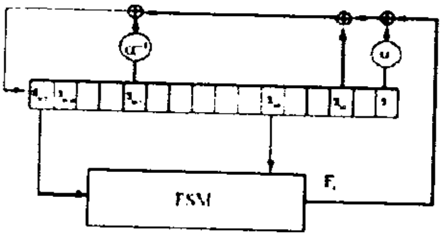


Fig. 2: Structure of SNOW 2.0 during Key Initialization

Step I: Make guess of secret key and IV values. For this purpose we have used the random function, which will take random values from given range.

Step II: Convert the guessed key into binary form because SNOW is additive Stream Cipher (i.e. a cipher in which plain text, cipher text and secret key must be in binary form).

Step III: Transform binary form of guessed key into 32-bits because 32 bit registers are used for storage and operations.

Step IV: Now secret keys and IV values are ready to initialize the cipher.

Step V: Proper working has been started and keystream are generated and stored in an attacking file.

Comparison Algorithm: Read the next keystream from original file when first keystream of original file is compared with all attacking keystream.

```
Key_comparison
    {
    Set IV - input through keyboard
    Key = snow2 ( IV )
    vcount = 0
    j = 0
    do{
    hcount = 0

    for i - 0 to n
    {
    if (key(i) = = dic_key(i,j))
    {
    put 1 to file
    hcount = hcount++
    vcount(i,j)++
    }
else
{
put 0 to file
}

}
j++
}
while (!eof)
;
```

The algorithm executes in this fashion to compare the original key with each and every keystream of attack file so that to locate the specific byte positions in key, those can be traced by Guess-and-Determine (GD) attack.

Experimental Analysis: A large number of experiments were conducted and its data was analyzed through statistical tools to find out the frequency of each byte position, so as to achieve the goal of finding traceable bit patterns in key. For this purpose we compare the original keystream with attacking keystream through comparison algorithm. Different data sets were taken to minimize biasness in the population; therefore the experiment was conducted on small datasets as well as large data sets. The complete analysis includes two phases.

In phase I, we determine the traceable bit patterns in key, a number of experiments were conducted on comparatively small randomly generated dictionaries and we find the following facts:

The GD attack with low intensity i.e. only with 500 guess keys, it was found that the frequency of 0,1,2,3 was 64.2, 29.2, 5.6 and 1.0 respectively, i.e. out of 500 guess keys 64.2% keys were not matching at byte position, but 1.0% of the keys had the successful match at 3 locations, i.e. 5 out-of 500 keys has 3 byte similarities which is too less if we guess the entire key.

$$f(x_i) \in f(x'_i),$$

where i is the byte position where one byte is equal to 8 bits, so if we compare it one key only which has 3 matches has 24 bits similarity with a specific position, which means that 24 bits have been traced in a pattern with a population of size 500. Figure 3 shows the frequency of byte similarity.

Figure 4 represents a graph shows the similarity of the key with the dictionary bit position wise which may give clue to traceable bit stream patterns.

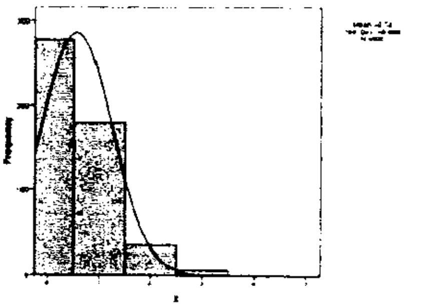


Fig. 3: Frequency graph for dictionary of 500 elements

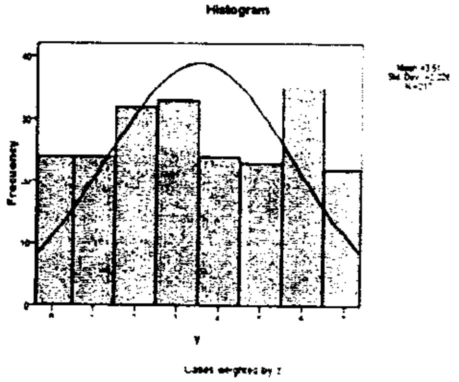


Fig. 4: Frequency graph for bit stream patterns for population of size 500

The graph in Figure 4 shows that byte positions 2, 3, 6 are more frequent as compared to the others, the dictionary is tested for a number of data sets and it was shown that these byte positions are more vulnerable to the cryptanalyst.

When the population is increased to 200000, it is found that more than 60% of the key is traced; by having 5 byte similarities in 8 byte long key.

In a population of 200000 guess keys, it was found that the frequency of 0, 1, 2, 3, 4, 5 was 59.756, 31.75, 7.43, 0.9775, 0.0835 and 0.0030 percent respectively, i.e. out of 200000 guess keys 59.756% keys were not matching at byte position, but 0.0030% of the keys had the successful match at 5 locations, i.e. 6 out-of keys 200000 has 5 byte similarities which is too less if we guess the entire key.

The graph in Figure 5 represents the similarity of keys with the dictionary byte position wise which may give clue to traceable bit stream patterns.

Table 1: Frequency table for a dictionary of 200000 attacking keys

Row-wise		
	Frequency	
Percent	Valid Percent	
Cumulative Percent		
Valid 0	119512	59.
59.6	59.8	
1	63500	31.
31.8	91.8	
2	14860	7.
7.4	98.9	
3	1955	0.
1.0	99.9	
4	167	1.
.1	100.0	
5	6	0.
.0	100.0	
Total	200000	100.
100.0		

Table 2: Byte position frequency for dictionary of 200000 attacking keys

Position		
	Frequency	
Percent	Valid Percent	
Cumulative Percent		
Valid 0	12343	12.
12.4	12.4	
1	12516	13.
12.5	24.9	
2	12569	12.
12.6	37.5	
3	12463	13.
12.5	50.0	
4	12522	13.
12.5	62.5	
5	12583	13
12.6	75.2	
6	12405	12
12.4	87.6	
7	12382	12.
12.4	100.0	
Total	99783	100.
100.0		

Table 3: Cumulative frequency for data of different sizes

	Byte Position								
Sample Size	0	1	2	3	4	5	6	7	Total
500	11.1	11.1	14.7	15.2	11.1	10.6	16.1	10.1	100
1000	12.4	10.4	13.8	13.4	11.6	12.2	14.1	12.0	99.9
2000	11.5	12.2	12.5	13.5	13.3	10.4	13.2	13.5	100.1
5000	12.4	13.8	12.2	12.3	13.4	12.4	12.7	10.8	100
10000	12.7	12.7	12.7	13.0	12.2	12.3	12.4	12.0	100
20000	12.3	12.1	12.7	12.9	12.1	12.9	12.2	12.8	100
30000	12.6	12.6	12.2	13.0	12.1	12.1	12.5	12.9	100
40000	12.6	12.6	12.4	12.1	12.4	12.9	12.4	12.6	100
50000	12.6	12.4	12.5	12.5	12.3	12.6	12.5	12.6	100
60000	12.3	12.2	12.5	12.6	12.5	12.8	12.4	12.7	100
70000	12.4	12.3	12.3	12.3	12.5	12.9	12.8	12.4	99.9
80000	12.7	12.4	12.4	12.5	12.4	12.4	12.8	12.4	100
90000	12.4	12.1	12.7	12.4	12.6	12.5	12.7	12.6	100
100000	12.7	12.7	12.3	12.3	12.6	12.4	12.4	12.6	100
200000	12.4	12.6	12.5	12.5	12.6	12.5	12.4	12.4	99.9
C. %age	12.34	12.28	*12.69333*	*12.83333*	12.38	12.26	*12.88571*	12.29333	

The following table represents frequency of each byte position and its percentage which represents that positions 2, 5 are more frequent as compared to other byte positions so, we can see that data at position 2 and 5 are more vulnerable to the cryptanalyst.

The graph in Figure 6 shows the similarity of the key with the dictionary byte position wise which may give clue to traceable bit stream patterns.

The graph shows that byte positions 2, 6 are more frequent as compared to the others, the dictionary is tested for a number of data sets and it was shown that these byte positions are more vulnerable to the cryptanalyst.

This graph represents that there is uniformity in each byte position and the frequency is also approximately same, but if we attempt even more clever attacks then we see that some of the byte positions are vulnerable to the cryptanalyst. For this purpose we collect data from different population sizes and we pass it through statistical tests it is observed that data at location 2, 3, 6 is comparatively more frequent.

Cumulative Analysis: The following table contains location based frequency for all experiments and its relative frequencies:

From this graph it is clear that byte position 2, 3 and 6 are more frequent as compared to the other ones.

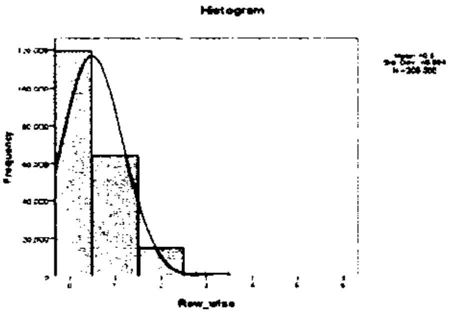


Fig. 5: Frequency graph for dictionary of 200000 elements

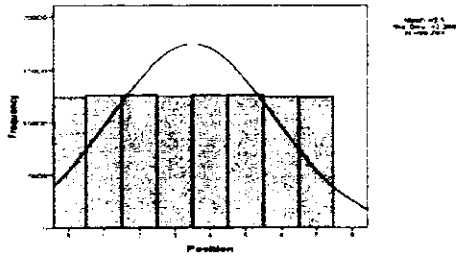


Fig. 6: Frequency graph for bit stream patterns for population of size 200000

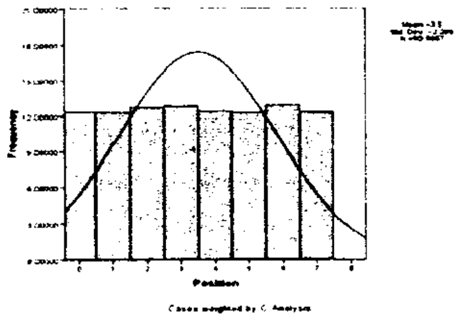


Fig. 7: Cumulative Frequency graph for bit stream patterns based on data in table 3

On comparing the original key with the dictionary we find that in 8 byte long key 5 bytes were traced to be similar with six different attacking keys in a population of 200000 attacking keys which is 0.0030% of the entire population, where as 4 bytes were traced to be similar with 167 different attacking keys which is 0.0835% of the entire populations. On choosing more clever initial guesses.

CONCLUSION

In this paper we have introduced the statistical analysis of the Guess-and-Determine attack on the stream cipher SNOW 2.0, based on advanced Guess-and-Determine attacks. The study is limited to the byte tracing instead of bit tracing, so as to justify the bulk amount of similar bit patterns in the original key. This study opens a way that in Guess-and-Determine attacks if some portion of the key is traced then it is preserved and recycled for tracing other similar bit patterns.

REFRENCES

1. Ullah, S.I., A.A. Tabassam and S.H. Khiyal, 0000. "Cryptographic weaknesses in modern stream ciphers and recommendations for improving their security levels".
2. Ekdahl, P., 2003. "On LFSR based Stream Ciphers-Analysis and Design", Ph.D. Thesis, Dept. of Information Technology, Lund University.
3. Hawkes, P. and G. Rose, 2002. Guess and Determine Attacks on SNOW. Qualcomm Australia, SAC 2002, LNCS., 2595: 37-46.
4. De Canniere, C., 2001. Guess and Determine Attack on SNOW. NESSIE Public Document," NES/DOC/KUL/WP5/011/a.See http://www.cryptonessie.org.
5. Mohammadi Chambolbol, A., 2004. Cryptanalysis of Word-Oriented Stream Ciphers, MSc. Thesis, Sharif University of Technology, Sept. 2004.
6. Ahmadi, H. and T. Eghlidos, 2005. "Advanced Guess and Determine Attacks on Stream Ciphers", IST 2005, pp: 87-91.
7. Maximov, A. and A. Johansson, 2005. "Fast Computation of Large Distributions and Its Cryptographic Applications". Lecture Notes in Computer Science ASIACRYPT, pp: 313-332.
8. Ahmadi, H. and Y.E. Salehani, 0000. "A Modified Version of SNOW2.0"
9. Ekdahl, P. and T. Johansson, 2002. "A new version of the stream cipher SNOW", Proceedings of Selected Areas in Cryptography (SAC) 2002, Springer, 2002. Available at, citeseer.ist.psu.edu/ekdahl02new.html

PAPER- III

IEEE — 2005 International Conference on Emerging Technologies
September 17-18, Islamabad

Cryptanalytic Weaknesses in Modern Stream Ciphers and Recommendations for Improving their Security Levels

Syed Irfan Ullah Ahmad Ali Tabassam Sikandar Hayat Khiyal
Department of Computer Science, Faculty of Applied Sciences
International Islamic University, H-10 Islamabad, Pakistan
E-mail: {syedirfan_phd, ahmadthe8}@yahoo.com, hdcs@iiu.edu.pk

Abstract

In this paper we discuss security problems in some modern stream ciphers. As we observe some times that a designer claims that the algorithm designed is more secure but when it comes to open literature we find a number of problems. We discuss SNOW, Scream and Rabbit. Some efforts have been made to overcome the problems those were pointed out in these cryptosystems by different cryptanalysts. The stream ciphers are faster and efficient than block ciphers but comparatively less secure. Our emphasis in this paper is to make some compromise on efficiency but to get more security.

Index Terms *- Stream ciphers, Block ciphers, SNOW, Scream, Rabbit, chaotic.*

I. Introduction

First we discuss SNOW 1.0. The idea for its design is taken from the classical summation generator [1]. The design of the cipher is quite simple, consisting of LFSR i.e. linear shift feedback register feeding a finite state machine. The new version of this stream cipher is SNOW 2.0[2], which is more secure and a bit faster as compared to SNOW 1.0. Although some minor changes have been made in SNOW 1.0 as the word size has not been changed (32 bit) and the LFSR length is again 16 [2].

It is claimed that SNOW 2.0 is more secure and has more resistance against guess and determine attacks. Although some minor changes have been made in original SNOW and we got its more secure version. So, by making very few but careful changes we can close the entry points for ciphers.

Then we discuss Scream, which is considered to be a more secure SEAL [9]. We have made some changes to make it even more secure as some of the analysts have found out weaknesses in the original Scream.

Then we discuss Rabbit, which is considered to be a high performance stream cipher [3]. We analyze the security of this model against different possible attacks and the resistance of the model against these attacks; we found out some weaknesses, those are removed by making some changes in the model.

The organization of this paper is as follows:

In section II we discuss the shortcomings in original cryptosystem and their resistance against different attacks. In section III we present our work that how these systems can be improved with less overhead and minimal changes. In section IV we analyze these improved models that why these are so logical and providing more strength to the cryptosystems. We conclude and summarize our work in section V.

II. Analyzing the models

In this section we discuss the shortcomings of the cryptosystems SNOW 1.0, Scream and Rabbit respectively.

A. SNOW 1.0

The SNOW 1.0 is a word oriented stream cipher with word size of 32 bits [1]. The cipher is described with two possible sizes i.e. 128, 256 bits. As usual encryption starts with key initialization, giving the components of the cipher their initial key value [1, 2]. The guess and determine attacks has the data complexity of 295 words and the process complexity of 2224 operations [4], if we have clever initial choices then of the complexity can be decreased even more. There are some weaknesses in SNOW 1.0, which also reduces the complexity of the attack below the exhaustive key search [2].

Finite State Machine (FSM) has only one input function s (1). It enables an attacker to invert the operations in FSM to derive more unknowns from only a few guesses. There is an unfortunate choice of

0-7803-9247-7/05/$20.00 ©2005 IEEE

feedback polynomial in SNOW 1.0. The linear recurrence equation is given by:

$$S_{t+16} = \alpha(S_{t+9} + S_{t+3} + S_t) \qquad (1)$$

There is a distance of 3 words between S_t and S_{t+3} and a distance of 6 = 2.3 between S_{t+3} and S_{t+9}. Thus by squaring

$$S_{t+32} = \alpha^2(S_{t+18} + S_{t+6} + S_t) \qquad (2)$$

We can see that ($S_{t+i} \oplus S_{t+i+6}$) can be considered as a single input to either equation. Hence, the attacker does not need to determine S_{t+i} and S_{t+i+6} explicitly but only the XOR sum to use in both ones [2].

The choice of the feedback polynomial emerges when considering bitwise linear approximations. Using the same techniques as in [11] we can take the 2^{32}th power of the feedback polynomial i.e.

$$P(x) = x^{16} + x^{13} + x^7 + \alpha^{-1} \in F_2^{32}[x] \qquad (3)$$

$$P(x) = x^{16.2^{\wedge}32} + x^{13.2^{\wedge}32} + x^{7.2^{\wedge}32} + \alpha^{-1.2^{\wedge}32} \in F_2^{32}[x] \qquad (4)$$

Since $\alpha^{1} \in F_2^{32}$ we have $\alpha^{-1.2^{\wedge}32} = \alpha^{-1}$ summation of p(x) and $p^{2^{\wedge}32}(x)$ yields

$$x^{16.2^{\wedge}32} + x^{13.2^{\wedge}32} + x^{7.2^{\wedge}32} + x^{16} + x^{13} + x^7 \qquad (5)$$

dividing this equation by x^7 we get linear recurrence equation satisfying

$$S_{t+16.2^{\wedge}32-7} + S_{t+13.2^{\wedge}32-7} + S_{t+7.2^{\wedge}32-7} + S_{t+9} + S_{t+6} + S_t = 0 \qquad (6)$$

We derive the linear recurrence equation that holds for each single bit position. Hence, any bitwise correlation found in the FSM can be turned into a distinguishing attack. For correlation and distinguishing attacks we need about 2^{95} words of output and the computational complexity about 2^{100} [6]. This discussion proves that SNOW 1.0 is a very weak cryptosystem and some changes were made to improve its quality; those were addressed in the design of SNOW 2.0 [2], which is also having some problems. We discuss it in section III-A

B. Scream: A software efficient stream cipher

Scream is a software efficient stream cipher that was designed to be a more secure SEAL [7, 9]. This model resembles in many ways a block-cipher design but it offers a significantly higher level of security. As SEAL is specialized in encrypting small messages and data authenticity, that has a number of weaknesses [7]. One of the proposed attack [8] that requires 230 "samples", each 4-words long to distinguish SEAL from random function. The new version SEAL 3.0 was proposed but it was also having some weaknesses, so based on the design of SEAL a new model Scream was proposed in different versions. The first version Scream that is so-called toy cipher due to its simplicity has many more weaknesses.

For scream family of cipher two distinguishing attacks are proposed. The best one has the complexity around 280 [8]. Although if we implement distinguishing attack on scream then the attack uses 2105 output words and has complexity of a similar size. We can decrease the complexity of the attack against scream by improving the model of attack [10]. Some other attacks proposed for scream are linear attacks and low diffusion attacks [9], show that what ever the claims are, we can distinguish the cipher from a truly random sequence which tends the system to cryptographic weaknesses and invite new attacks.

The main problem is in the for-loop where we find X block which is 16 bytes long, where it is XORed directly with Y and we send it to the function F ( ) as argument, that may allow the ciphers to distinguish the key from a truly random sequence.

C. Rabbit

The design of Rabbit was inspired by the complex behavior of real-valued chaotic maps. These maps are primarily characterized by an exponential sensitivity to small perturbations causing iterates of such maps to seem random and long-time unpredictable [11]. These properties have also previously leaded to suggestions that chaotic systems can be used for cryptographical purposes [12, 13]. Due to this modern method used for designing the model makes cryptanalysis difficult but even then efforts have been made to analyze it. The cryptanalysis of the Rabbit is resulted in the following:

To investigate the possibilities for Divide-and-Conquer and Guess-and-Determine types of attacks, an algebraic analysis was performed with special attention on the nonlinear parts of the next state function, as they are the main sources of mixing the input bits [14, 15]. Some of the attacks like correlation and distinguishing attacks have got some sort of success against it. The literature gives us some clue the there are some deficiencies in the model through which we can peep in to the interior of the model specifically due to some weaknesses in the key scheduling algorithm of the model.

There is a possibility of related key attacks that exploit the symmetries of the next state and key setup function. For instance consider two keys K and K' related by K[i] = K'[i+32] for all i. This lead to the relation, $X_{j,0} = X'_{j+2,0}$, and $C_{j,0} = C'_{j+2,0}$. In the same way this symmetry could lead to a set of bad keys, i.e. if K[i] = K [i+32] for all i, then $X_{j,0} = X_{j-2,0}$ and $C_{j,0} = C_{j+2,0}$ and in this way the key can be traced.

We need to take care of the brute force attack, and other classical and modern attacks. Due to advancement in mathematics we cannot guarantee that a cryptosystem will always be secure and foolproof but we can give more tough time to cryptanalysts.

III. Recommendations for improvements in models

In this section we present the changes those we have made in the original models. In section II we discussed the problems in the original models. To minimize their severity we make the following changes.

A. SNOW 1.0

In section II-A the design analysis of SNOW 1.0 has been discussed in more detail and it has been shown that SNOW 1.0 is a very weak stream cipher. So, these weaknesses were addressed and removed in new version of SNOW i.e. SNOW 2.0 [2], but very soon a number of weaknesses were addressed by different analysts [4, 6] and proved that this version is also not so secure.

So, we make some changes in the basic design of SNOW 2.0 as SNOW 1.0 has already been improved to release its next version. Both versions of SNOW have basically the same design but some minor changes, so the basic design remains same for SNOW.

a. Convert the LFSR property to partial NLFSR:
   i. Take circular left shift after XORing α (alpha) with S_{t-2}.
   ii. Take circular left shift again after XORing α^{-1} (alpha inverse) with S_{t-11}.

b. Before XORing with R_2 take circular left shift of ($R_1 \boxplus S_{t-15}$) once again.

We represent the model diagrammatically with the above improvements. The diagram clearly represents that we need memory buffers at three different places to store the bit stream on temporary basis.

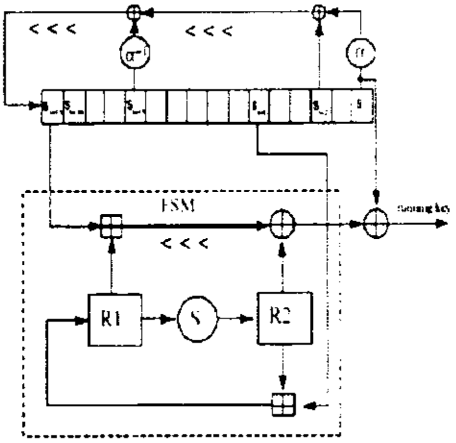


Figure 1: The new model of SNOW 2.0

B. Scream: A software efficient stream cipher

Some of the attacks although less efficient have been found, changes have been made in it [9] to improve the security but the efficiency have been lost by 10 % --15% although [10] analyze it for security. So, we make some changes as such to improve the security as well as less compromise on performance 2 % -- 5%.

The main weaknesses are in the for-loop where we find X block that is 16 bytes long, where it is XORed directly with Y and we send it to the function F ( ) as argument, that may allow the ciphers to distinguish the key from a truly random sequence. The changes that we have made are as follows:

a. Divide X block into 16 chunks, each one byte long.
b. Re-compute each byte by XORing it with the next byte.
c. Before XORing with the next byte, rotate it one bit to the left.
d. With the last byte the first recently computed byte is XORed.

e. Recombine all the bytes to get a final 16 bytes block.

f. Compute X = F(X ⊕ Y).

In this way we enjoy the stream cipher advantages 16 byte blocks. As stream ciphers are speedily computable, so the above computation takes less time as well as the non-linearity has also been maintained and thus security is improved.

C. Rabbit

We see in section II-C that there are some weaknesses in the key scheduling of Rabbit as we make some changes in its design that generates the key stream as such to minimize the attacks on it.
In the next state functions every X affects the next second state which is easily traceable so we make the following changes:

a. Make the map as such that the current state X affects the next X state and that X affects the next second state and that affects the next third state. Thus there is no linear increase in the state scheduling. At the fifth state the process is repeated once again as for the first four states.

b. The process continues till all states change their values at least once.

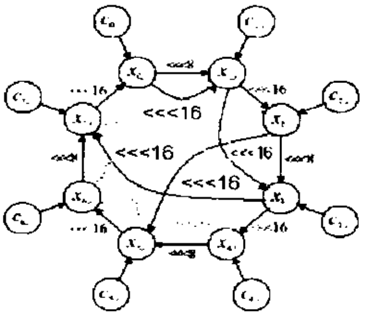


Figure 2: The New model of Rabbit

So, by making these few changes the attacks on key tracing become more difficult.
The nonlinearity is an important property of Rabbit, but we see that even then if one state is guessed we get information about the next state because the next state computation is based upon the previous state. The linearity problem is removed from the model by making these changes. The state diagram clearly specifies that by making these changes no one can determine the formula in a straight forward manner. So, a person who generates the differential equation after spending a lot of time can be discouraged just by increasing the number of states. Thus even more organized attacks may fail against this model. The changes that we have made are represented through diagram, which have a number of doubts that how we represent it mathematically, which is the actual achievement.

IV. Analysis of the improved models

In this section we prove that the changes we make are very logical and important. The security of the cryptosystems has been improved by making these efforts.

A. SNOW 1.0

Both versions of SNOW are based on LFSR so, the linear attacks are more efficient to analyze and crack them. We convert their design to Partial Nonlinear Feedback Shift Registers i.e. PNLFSRs, so the linear attacks are mostly discouraged. We give a partial touch of chaotic map [13] in our model. Chaotic maps give more strength to the security of cryptosystems. It also discourages the guess and determine attacks. The overhead that we make decreases the performance of the system. In the original model the linear recurrence equation is given by:

$$S_{t+16} = \alpha(S_{t+9} + S_{t+3} + S_t) \qquad (7)$$

There is a distance of 3 words between S_t and S_{t+3} and a distance of 6 = 2.3 between S_{t+3} and S_{t+9}. Thus by squaring:

$$S_{t+32} = \alpha^2(S_{t+18} + S_{t+6} + S_t) \qquad (8)$$

As in SNOW 2.0 α (alpha) and α^{-1} (alpha inverse) are XORed but even then they can be traced if we are able to succeed in finding α. So, left circular shift makes it difficult to trace the sequence as every time the sequence is circulated that is stored inside temporary buffer at three different places in our model. The input to the FSM is (S_{t+15}, S_{t+5}) but S_{t+15} is the sequence number after left circular shift, so the output of the FSM denoted by F_t, is computed as:

$$F_t = (\text{LeftCircularShift}\ (S_{t+15}) \boxplus R1_t) <<< n \oplus R2_t) \qquad (9)$$

239

where n is a chosen prime number.

The linear or distinguishing attacks do not affect the model even if we have more clever attacks. The key sequence can not be determined by linear sequence. The guess and determine attacks are also discouraged as these are based on initial guesses, but we see that due to repeated circular shift the guess is in contradiction with the exact value as the cipher is always confused about the exact location of the bit due to its circular shifts.

We do not make severe changes in the basic design of the cryptosystem to keep the thing same as in the original model. The model is so weak that by making changes we can decrease the severity of the attack but we can not fully stop attacks on it. The drawback in our model is that we store the key sequence at three different locations in temporary buffers that makes the model a little inefficient.

B. Scream: A software efficient stream cipher

It is an advanced stream cipher that provides better security. It is more efficient than SEAL, but even then we look in Scream-0 that some security flaws are there, specially in statement X := F(X ⊕ Y). As X and Y both are XORed in F ( ), which give way to ciphers for tracing the key.

The F ( ) function has linear approximations that approximate only three of the 8-by-8 S-boxes. Since the S-boxes in Scream-0 are based on the Rijndeal S-box, the best approximation of them has bias 2^{-3}. So, we can probably get a linear approximation of the F ( ) function with bias 2^{-9}.

For linear approximation we need to eliminate the linear masking, as each of these masks is used 16 times before it is modified. For each step the cipher looks a pair (X ⊕ Y ⊕ W[i], F (X) ⊕ Z ⊕ W[i+1]), where X is random. So if X is found once then we can trace the whole sequence. So in scream-F this problem was removed but it lost the efficiency by 10-15%.

So we make changes such that to improve the security as was challenged by direct XORing of X and Y. The basic quality of the model has also been restored. The theory of chaotic maps has been introduced in the model which provides better security by splitting the X in to 16 individual bytes and each one has been XORed with the next one. So, it makes it difficult for cipher to trace the sequence by applying the linear approximation and the (X ⊕ Y ⊕ W[i], F (X) ⊕ Z ⊕ W[i+1]) has been converted to even more complex shape i.e. ($(X_i \oplus X_{i+1} <<< 1)$ ⊕ Y ⊕ W[i], F (X) ⊕ Z ⊕ W[i+1]). The pair will never give the required result to cipher as it changes all the times. So, the cipher will not easily trace the sequence as well as the efficiency has also been restored up to some extant that was the basic purpose of the model to provide speedy encryption.

The basic purpose of the model has been restored along with the improved security to stop linear, differential and guess and determine attacks.

C. Rabbit

It is based on chaotic map but there are very little design weaknesses that allows cipher to peep in to the model. We see that every current state affect linearly the next second state. We can determine how this state affects the coming state as mathematically represented in the section II-C

In original model the g-function is computed by making some operation on the state of the map [11]. We see that every state is computed by XORing the previous two states as follows.

$$X_{0,j+1} = g_{0,j} + (g_{7,j} <<< 16) + (g_{6,j} <<< 6) \quad (10)$$
$$X_{1,j+1} = g_{1,j} + (g_{0,j} <<< 8) + g_{7,j} \quad (11)$$
$$X_{2,j+1} = g_{2,j} + (g_{1,j} <<< 16) + (g_{0,j} <<< 16) \quad (12)$$
$$X_{3,j+1} = g_{3,j} + (g_{2,j} <<< 8) + g_{1,j} \quad (13)$$

.

.

.

The g-function is the computed in straight forward mathematical function.

$$g_{j,i} = ((X_{j,i} + C_{j,i})^2 \oplus ((X_{j,i} + C_{j,i})^2 >> 32)) \bmod 2^{32} \quad (14)$$

This g-function can be summarized as follows.

$$g(Y) = (Y^2 \oplus (Y^2 >> 32)) \bmod 2^{32} \quad (15)$$

This g-function has a very huge complexity and only with very small word length it has been examined [14]. This discussion reveal that the non-linear order of this function is maximal and only through high-level differential it can be examined due to some weaknesses mentioned here.

In our model every current state affects the next states in different manner which makes it difficult to analyze it in a linear and even in differential way. The guess and determine attacks have also been discouraged.

The differential analysis is impossible because we are not always sure after how much iteration every state is affected, if we increase the number of states for achieving more security the mathematical representation is only possible by making some logics. So, even a clever attacker is always confused about the state of the art.

The speed of the cryptosystem is not affected as graphical and mathematical model is changed through logics only.

V. Conclusion

We discussed some of exemplary stream ciphers those are claimed by their designers to be more secure, but we saw that every model has been traced when they came to open literature. We observed that every model has left some very minor weaknesses those may be considered as ignorance of designers or there may be some other things that every designer had left some peeping point to make it always possible for them to crack their own models. We saw that with minor efforts those peeping points have been closed. The world is invited for any suggestion related to the paper will be encouraged to improve our models.

References

[1] P. Ekdalhl, T. Johansson, "SNOW- A New Stream Cipher" *Proceedings of first NESSIE Workshop, Heverlee, Belgium*, 2000.

[2] P. Ekdalhl, T. Johansson, "A new version of the stream cipher SNOW", Dept. *of Information Technology, Lund University, Sweden.*

[3] M. Boesgaard, M. Veserager, T. Pedersen, J. Christiansen, "Rabbit: A new high performance stream cipher" *Fast Software Encryption 2003, LNCS, Springer-Verlag*, www.cryptico.com.

[4] P. Hawkes, G.Rose, "The guess and determine attack on SNOW" *Proceedings of Selected Areas in Cryptography (SAC), St. John's Newfoundland, Canada August 2002.*

[5] P. Ekdahl, T. Johansson, "Distinguishing attacks on SOBER", *In Fast Software Encryption (FSE), 2002, Springer 2002.*

[6] D. Coppersmith, S. Halevi, C. Jutla. "Cryptanalysis of stream ciphers in linear masking". *In Advances in Cryptography – CRYPTO' 02, Springer-Verlag, 2002.* http://eprint.iacr.org/2002/020/

[7] P. Rogaway, D. Coppersmith, "A software optimized stream cipher SEAL". *In Journal of Cryptology, September 18, 1997.*

[8] H. Handschuh and H. Gilbert, "X^2 cryptanalysis of SEAL encryption algorithm", *In Proceedings of the 4th Workshop on Fast Software Encryption, Springer-Verlag, 1997.*

[9] S.Halevi, D. Coppersmith, C. Jutla , "Scream: A software efficient stream cipher", *In Fast Software Encryption (FSE) 2002, Springer 2002.*

[10] J. Thomson, A. Maximov, "A linear distinguishing attack on scream", *sDept. of Information Technology, Lund University, Sweden , 2004.*

[11] M. Boesgaard, T. Pedersen, M. Vesterager, J. Chriestiansen, O. Scavenius, "Rabbit: a new high performance stream cipher", *CRYPTICO A/S, Copenhagen, Denmark.*

[12] S. Wolfarm, "Cryptography with cellular automata", *Proceedings of Crypto '1985.*

[13] G. Jakmoski, L.Kocarev, "Chaos and cryptography: block encryption ciphers based on chaotic maps", *IEEE, Transactions on Circuits and Systems-I: Fundamental Theory and Applications, 2001.*

[14] M. Boesgaard, T.Pedersen, M. Vesterager, E. Zenner, "The Rabbit stream cipher: Design and Security Analysis", *CRYPTICO A/S, Copenhagen, Denmark.*

[15] V. Rijmen, "Analysis of Rabbit", *Crypromathic September 5, 2003.*