

ANALYSIS AND IMPLEMENTATION OF RC4 ALGORITHM FOR STREAM CIPHERS

A DISSERTATION

Submitted in partial fulfilment of the requirements for the award of the degree of

MASTER OF TECHNOLOGY

in

COMPUTER SCIENCE AND ENGINEERING

by

SWAGATAM ROY ROCKY



DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

INDIAN INSTITUTE OF TECHNOLOGY, ROORKEE

Roorkee – 247667 (INDIA)

May, 2019

DECLARATION

I hereby declare that the thesis “Analysis and Implementation of RC4 Algorithm for Stream Ciphers” is my own work, done under the supervision of Professor Dr. Sugata Gangopadhyay. It is carried out for the degree of Master of Technology in Computer Science and Engineering. Resources of work used here found by other researchers are acknowledged by reference. This dissertation, neither in whole or in part, has been previously submitted for any degree.

Signature of Author

Date

CERTIFICATE

This is to certify that the above declaration by the student is true to the best of my knowledge.

Signature of Supervisor

Date

ACKNOWLEDGEMENT

I would first like to thank my thesis advisor Professor Dr. Sugata Gangopadhyay of the Department of Computer Science & Engineering at Indian Institute of Technology, Roorkee. The door to his office was always open for me. He consistently allowed this paper to be my own work, but steered me in the right the direction whenever he thought I needed it.

Finally, I am grateful for the influence and support of faculties, friends and my family members, who have been a great motivation for me throughout my work. This accomplishment would not have been possible without them.

SWAGATAM ROY ROCKY



TABLE OF CONTENTS

DECLARATION	i
ACKNOWLEDGEMENT	ii
TABLE OF CONTENTS	iii
LIST OF FIGURES	vii
LIST OF TABLES	viii
ABSTRACT	ix
Chapter 1	1
1 Introduction	2
1.1 History of RC4	3
1.2 History of Other Stream Ciphers	3
1.2.1 Salsa20 and ChaCha	4
1.2.2 SOSEMANUK	4
1.2.3 PANAMA	5
1.2.4 Chameleon	5
1.3 Thesis Outline	5
Chapter 2	6
2 A Deeper Look Into RC4	7
2.1 The RC4 Algorithm	7

2.2 General Description	7
2.3 The Internal State	8
2.4 The State Change Function	9
2.5 Output Function	10
2.6 KSA – The Key Schedule Algorithm	10
Chapter 3	11
3 Attacks and Weaknesses	12
3.1 Attack Categories and Types of Weaknesses	12
3.2 Key Schedule Biases	13
3.2.1 Roos' Class of Weak Keys	14
3.2.2 Wagner' Class of Weak Keys	14
3.2.3 Key Schedule Invariance	15
3.2.4 Weak Keys and Related Key Attacks	15
3.2.5 Grosul and Wallach Related Key Pairs	16
3.2.6 Matsui Key Collisions	16
3.2.7 Paul and Maitra Equation Solving Method	16
3.2.8 Attacks on WEP	17
3.2.9 Attacks on WPA	17

3.3 Predictive States and Distinguishers	18
3.3.1 Predictive State Applications	19
3.3.2 Distinguisher Using Digraphs	20
3.3.3 Broadcast Attack Using Second Byte Bias	21
3.4 State Recovery Attacks	22
3.4.1 Knudsen's Attack	22
3.5 IV weaknesses	24
3.6 Other Attacks	24
3.6.1 Mironov's Analysis	24
3.6.2 RC4's Special States – The Finney States	25
3.6.3 Golic's Distinguishers	25
3.6.4 Paul and Preneel's Distinguishers	26
Chapter 4	27
4 RC4 Enhancements	28
4.1 RC4A	28
4.2 RC4B	29
4.3 VMPC	30
4.4 NGG	32
4.5 GGHN	33

Chapter 5		36
5 Implementation		37
5.1 RC4		37
5.1.1 Basic Encryption		37
5.1.2 Processing Time on Varying Key Lengths		40
5.2 RC4 vs RC4 – Fact		42
5.2.1 Algorithm Modifications		42
5.2.2 Comparison Between RC4 and RC4 – Fact		44
Chapter 6		46
6 Conclusion		47
Chapter 7		48
7 References		49

LIST OF FIGURES

Figure 2.1: KSA - “Key Schedule Algorithm”	8
Figure 2.2: PRGA – “Pseudo Random Generator Algorithm”	8
Figure 3.1: Modified KSA	15
Figure 3.2: RC4 Branch and Bound attack	23
Figure 4.1: RC4A PRGA	28
Figure 4.2: RC4B KSA	30
Figure 4.3: RC4B PRGA	30
Figure 4.4: VMPC Key Scheduling Algorithm	31
Figure 4.5: VMPC PRGA	31
Figure 4.6: NGG KSA	32
Figure 4.7: NGG PRGA	33
Figure 4.8: GGHN KSA	34
Figure 4.9: GGHN PRGA	35
Figure 5.1: A Secret key and plaintext merged to create a ciphertext	37
Figure 5.2: Processing time versus increasing secret key length	41
Figure 5.3: Encryption using the maximum sized secret key – 256 bytes	42
Figure 5.4: Modified KSA for RC4 – Fact	43
Figure 5.5: Modified PRGA for RC4 – Fact	43
Figure 5.6: Comparison of processing time between RC4 and RC4 – Fact	45

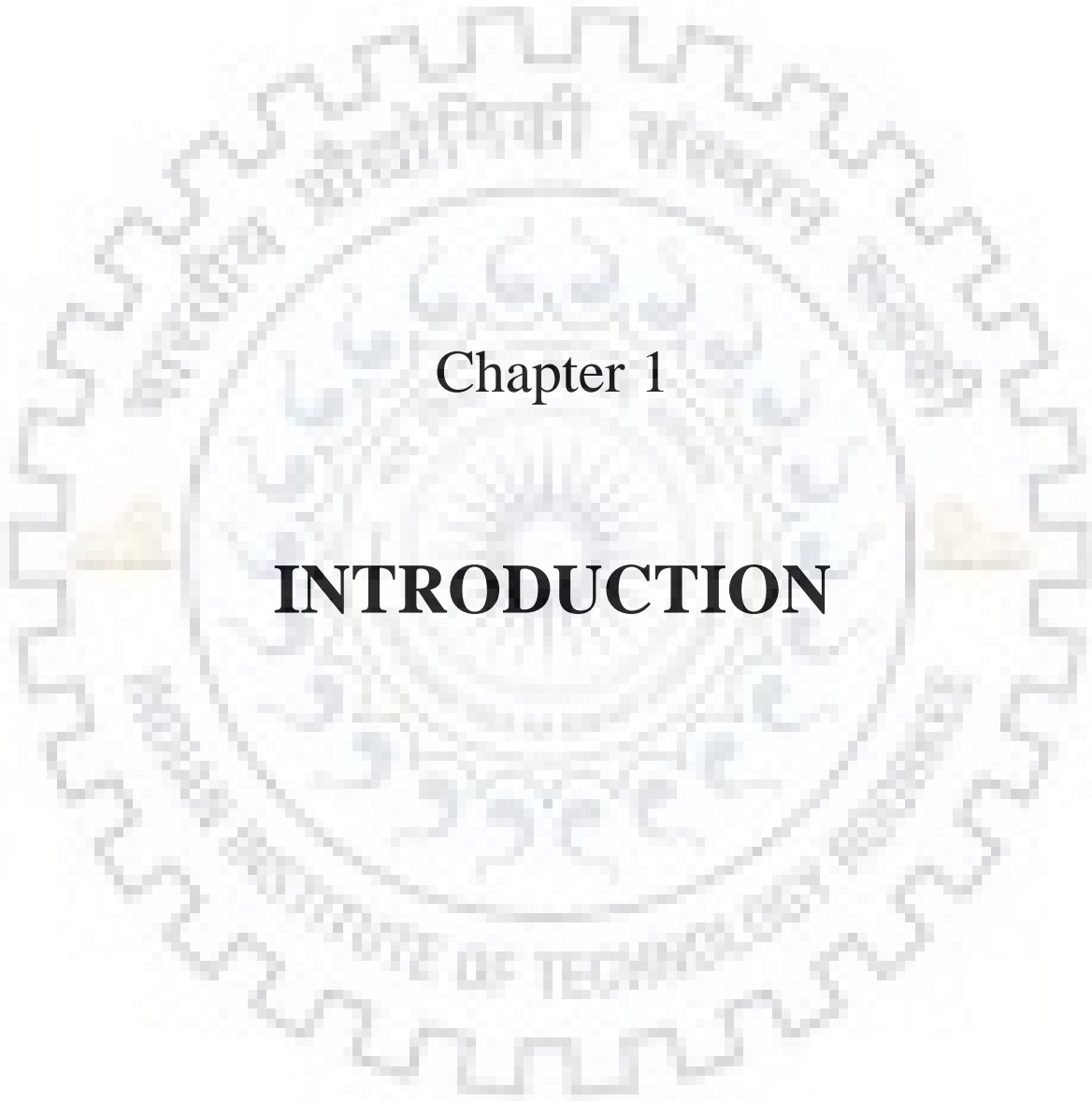
LIST OF TABLES

Table 5.1: Plaintext ASCII conversion	37
Table 5.2: Key ASCII conversion	38
Table 5.3: Repeating key patterns to fill 256 state spaces	38
Table 5.4: Keystream generated after key is permuted through KSA and PRGA	38
Table 5.5: Padded plaintext with pattern repetition to fill 14 bytes	38
Table 5.6: The padded plaintext is converted to ASCII code first and then binarized	39
Table 5.7: The keystream is converted to ASCII code first and then binarized	39
Table 5.8: Forming the final ciphertext	40
Table 5.9: The change in processing time based on increasing Secret Key Length	41
Table 5.10: RC4 runtime with varying key sizes	44
Table 5.11: RC4 – Fact runtime with varying key sizes	44

ABSTRACT

Since its public availability in 1994, RC4 has been extensively used for data – security measures. The specialty of RC4 lies within its formation – it is very easy to implement compared to other stream ciphers while providing excellent security. However, extensive research on the matter for more than two decades has exposed many frailties that lie within RC4 and how these frailties can be used to breach RC4. This has led to many modifications of the cipher along with the creation of various variants too. This dissertation focuses on analyzing the best of the heaps of research that has been done on RC4 since its birth and also to look into its computational efficiency compared to more modern versions of the cipher.





Chapter 1

INTRODUCTION

1. INTRODUCTION

Information security is becoming more and more important in our daily lives as technology becomes increasingly entangled in all aspects of our lives. From day to day conversations between friends to highly sensitive military data, everything is being passed around and this has created an ever-increasing necessity to keep this passage of data secured from unauthorized access to varying degrees. As a consequence, *Cryptology*, being an ancient study, is at its peak importance in our current society.

The science behind the application of information security is called *Cryptology*. In a more modern context, it is the study of safe keeping of data for the purpose of storage and/or transmission such that any unauthorized access to the data is blocked (or made extremely difficult). To do so, it is required to construct complex mathematical protocols which prevent third parties the access to the information unless they are able to meet certain strict criterion. The science of *Cryptology* is comprised of two studies namely, *Cryptography* – the study of creating ciphers and *Cryptanalysis* – the study of breaking ciphers.

Use of *Cryptology* has seeped into many corners – instant messaging, internet banking, using ATMs, satellite TVs, keyless access to cars, smart ID cards and many more. More sensitive use of cryptology is in modern military where it is used to protect strategical data such as guidance of missile, control systems of fighter jets, etc. It is also used for digital rights management and copyright infringement of digital media.

There are various cryptographic algorithms that are being used for different kinds of Information security, such as product ciphers developed by Claude Shannon[1], feistel network based DES developed by Horst feistel[2] and many others - one of which is RC4 which is very popular and is currently in use in various forms of security. This work focuses on understanding the popular stream cipher RC4 and applying cryptanalysis on it to check for frailties within its functions which can be used to breach RC4 by extracting information on its internal states giving away its mask of randomness enabling the attackers to differ an RC4 output stream from a genuine random stream. The computation prowess of RC4 is also evaluated compared to recent variants of the algorithm.

1.1 History of RC4

RC4 is the first known use of a dynamic permutation in stream cipher design, created by Ron Rivest in 1987, which is now a well-known cipher from the RSA Data Security, Inc. The algorithm was proprietary at the beginning but was leaked to an Internet mailing list in 1994 and since then people who knew about the real implementations of RC4 have passed it as authentic. Common use for the algorithm is in communications protocols like TLS (predecessor - SSL) and WEP, the IEEE 802.11 wireless networking security standard.

In general, RC4 can be described as a binary additive stream cipher which uses secret keys of varying sizes, usually 8 bits, in keystream. Its primary claim to fame being the elegantly simple way it works, allowing practical implementations be hassle free and less prone to mistakes compared to other ciphers of comparable security, such as DES [16]. Its use in the Wired Equivalent Privacy (WEP) protocol is well known, along with its use in Oracle, SSL (Secure Sockets Layer), TLS (transport Layer Security), SQL and Cellular Digital Packet Data specification among others [20]. However, it is also vulnerable to different security attacks where attacker's main goal is to find biases in the pseudo-random output keystream that it generates or in its internal state.

During the time when RC4 was developed, research on Stream ciphers were mostly focused on LFSRs – Linear Feedback Shift Registers [16]. From a mathematical standpoint, LFSRs are rather simple to study which made its security analysis a popular subject for research. However, LFSRs are rather inefficient and slow for the use of software implementation due to the use of numerous bit operations.

1.2 History of Other Stream Ciphers

RC4 has been a very popular cipher since its inception but it does come with a whole host of vulnerabilities. As research is getting more intensive on the topic, further biases and weaknesses are popping out. So, it would suffice to look into few other stream ciphers which are not as popular but have a good prospect of being used in various security measures if needed.

1.2.1 Salsa20 and ChaCha

Salsa20 is a stream cipher developed in 2005 by Daniel J. Bernstein who then submitted it to eSTREAM. Later, in 2008, he came up with a modification to Salsa20 which he called ChaCha. Both the ciphers share various similarities however ChaCha improves on it by increasing diffusion and performing better in some architectures [21].

Both Salsa20 and ChaCha are assembled on a pseudorandom function based on ARX (Add, Rotate, XOR) operations. The function consists of 32-bit addition, bitwise addition (XOR) and rotate operations. Its central function is to map a 256-bit key, a 64-bit nonce, and a 64-bit counter to a 512-bit keystream block. This core functionality gives both these ciphers an excellent advantage over most – the user is able to effectively and swiftly seek any position in the keystream and that too in constant time. On x86 processors, Salsa20 clock in around 4 to 14 cycles per byte [22] with good hardware performance. With Bernstein’s belief in “Open Source”, neither of the ciphers are patented and the developer himself has worked on various public domain applications which are adjusted for various mutual architectures.

1.2.2 SOSEMANUK

Another stream cipher with promise is SOSEMANUK which was developed by Berbain C. et al. in 2008 [23]. It is, along with Salsa20, Rabbit, HC – 128, one of the final four Profile 1 ciphers chosen for the eSTREAM portfolio.

The name translates to “Snow Snake” in Cree Indian language as it was based on the stream cipher SNOW and the block cipher Serpent. Compared to the cipher SNOW, SOSEMANUK performs better due to having a more efficient initialization phase – using a 128-bit initialization vector. For this cipher, the key length may range from 128 to 256 bits, however, it only guarantees security if the length is 128 bits. It is worth mentioning that when this cipher was being evaluated in eSTREAM phase 1, it was found that an attack with 2224 cost can be applied on it, but it does not contradict SOSEMANUK’s security claim which it guarantees up to 128bits [23]. This cipher is also free to be used by anyone.

1.2.3 PANAMA

Panama was developed by J. Daemen and C. Clapp in 1998 presented in the paper “Fast Hashing and Stream Encryption with PANAMA”. It is a cryptographic primitive that can be used as a hash function as well as a stream cipher. However, with time, its hash function functionality has been cracked and now it does not serve any practical value. It can still be used as stream cipher which uses a 256-bit key and accomplishes impressive performance by reaching 2 cycles per byte [24].

1.2.4 Chameleon

Chameleon was developed by Matthew E. McKague in 2004 at the University of Regina for the purpose of creating a game cipher that uses a card deck as its sole apparatus while being able to compete with computerized ciphers in terms of security. After the development of Chameleon, it was found that it has a striking resemblance to the core functionality of RC4. This was purely coincidental and even so Chameleon does differ in various ways from RC4 like the use of a “Mutating S-Box” – A new cryptographic primitive. Also, it is not a random key generator but rather an autokey cipher [25].

1.3 Thesis Outline

The thesis is organized as follows:

- Chapter 1 Discussion on the history of RC4, other stream ciphers and stream ciphers in general.
- Chapter 2 Detailed dive in the description of RC4
- Chapter 3 Studying the previous attacks on RC4
- Chapter 4 Studying different variants of RC4
- Chapter 5 Implementation of RC4
- Chapter 6 Conclusion



Chapter 2

A Deeper Look Into RC4

2. A Deeper Look Into RC4

The design of RC4 is very simple compared to other stream ciphers and it is incredibly small which has made it such a popular choice from security since its birth. What is even more impressive is that even after years of intense analysis it is still secure enough for most applications. This chapter is completely dedicated to the meticulous study of the inner structures of the RC4 algorithm and breakdown of the functions of these structures.

2.1 The RC4 Algorithm

To start off, we shall describe the entire RC4 algorithm in general before getting in to the details of its internal structures and functions. Here, we will take two n -bit arguments, i and j , and an array, $S[]$, of size $N = 2^n$ where the n -bit words are indexed to 0 through $N - 1$. Any summation required are worked with respect to modulo N .

The opening stage of the algorithm is called the “Key Scheduling Algorithm” which takes a key $k[]$ that consists of l n -bit words as described in **Figure 2.1**. Afterwards, the second algorithm – “Round Algorithm”, also known as the “Pseudo Random Generator Algorithm”, is executed one time for each word output. This is shown in **Figure 2.2**.

For most practical use cases of RC4, the value of n is kept at 8 which strikes the best balance between speed of processing and complexity.

2.2 General Description

Among the stream ciphers, RC4 is rather exceptional in regards to the internal structure, specially, the internal state of RC4 makes it unique. It can be described as a finite state machine which contains information about the internal state, a state change function that is subject to change with respect to the current state and an internal state dependent output function. When it is run, it firstly performs the state change function after that the output function. In general, this outline of functions can be used to describe any stream ciphers.

Input - Key k of l words

i) for $i = 0$ through $N - 1$,

$S[i] = i$;

ii) $j = 0$;

iii) for $i = 0$ through $N - 1$,

a) $j = j + S[i] + k[i \bmod l] \bmod N$;

b) **Swap** - $S[i]$ and $S[j]$;

iv) $i = j = 0$;

Output - i, j, S

Figure 2.1: KSA - “Key Schedule Algorithm”

Input – Internal State S, i, j

i) $i = i + 1 \bmod N$

ii) $j = j + S[i] \bmod N$

iii) **Swap** - $S[i]$ and $S[j]$

iv) **Output** - $S[S[i] + S[j] \bmod N]$

Figure 2.2: PRGA – “Pseudo Random Generator Algorithm”

2.3 The Internal State

RC4 is a binary additive stream cipher that operates on binary words of length n . It produces output simultaneously, one word each time, each of which can be any of $N = 2^n$ values. The pseudo-random arrangement of these N values makes up the internal state. This collection of pseudo random values is denoted as S and can be thought of in several ways, but in the case of RC4 it is usually an array or a look-up table. This collection of random values, S , needs to be applied to a word and then be multiplied by a transposition. The internal state of RC4 only houses in S and the variables i and j . For complexities sake, the internal state space is a vital issue as it provides the

ceiling to a brute force attack effectiveness. If the internal state is huge, possible number of states may be $2^{(2n)(N!)}$ which results in $\log_2 N! + 2n$ bits of data. That means, for example, if $n = 8$, it results in about 1700 bits which makes it practically impossible to apply brute force to determine the internal state. Furthermore, the magnitude of the internal state also puts a ceiling on cycle length (number of unique outputs before it starts to repeat). The limit is that the cycle length cannot extend the possible number of states. Again, it shows the positives of having a huge internal state. The internal state also houses the variables i and j which are essential mechanics to manage the state change and output function by working as indicators in the table \mathbf{S} .

2.4 The State Change Function

When the state change function works, it modifies both the variables i and j and also the table \mathbf{S} . This occurs during the first three steps of the round algorithm mentioned earlier. Step 1, variable i is incremented simultaneously for N rounds, so it indicates each element of the table \mathbf{S} once. So, when the swap function occurs, after N rounds, all elements of \mathbf{S} are scrambled. The variable i is set as a constant in the beginning and is not tied to \mathbf{S} , but, in step 2, the variable j is added with $S[i]$ and is not independent and private. So, if \mathbf{S} is a function that is pseudo random, j is a variable that is pseudo random that “randomly” indicates to different elements of \mathbf{S} . Step 3 can be considered as the most important step of the function where $S[i]$ and $S[j]$ are swapped which is like shuffling the table \mathbf{S} with two elements at a time. By the time all the elements are traversed, the whole of table \mathbf{S} is completely randomized. The output sequence and the traversal of variable j are both tied to the table \mathbf{S} which makes the output sequence erratic too, in turn making it extremely difficult to practically predict the output stream.

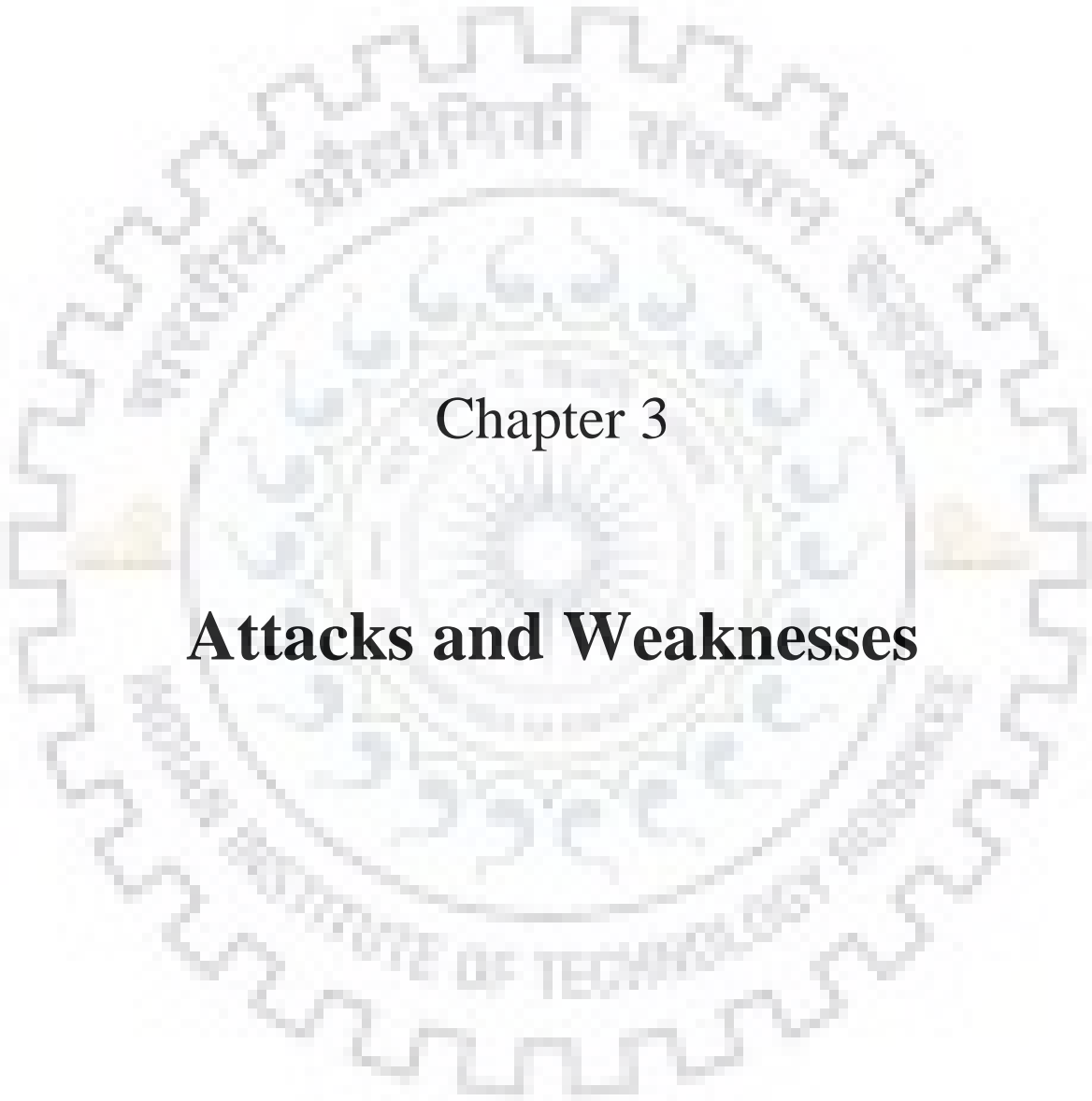
It is worth mentioning that the state change function is reversible, i.e., preceding states of the function are not lost and can be retrieved using the current state and the state change function. This attribute is useful to create longer cycle lengths as reversibility means all possible states of the function are available, which creates higher entropy in the output sequence. As the output stream can only use the information that are available, lack of reversibility would result in lost states causing an overall decrease in information and making the output sequence more predictable. Mathematically, the average cycle length is $\frac{2m}{2}$ for m bits of data in a randomly chosen state change function. If the function is reversible however, that cycle length averages at $2m - 1$ [7].

2.5 Output Function

The table \mathbf{S} is not truly random, which is why the two variables i and j are used to find two pseudo random numbers $S[i]$ and $S[j]$. These numbers are added together and mixed with \mathbf{S} once more to complete the procedure. These layers of operations do serve a purpose – as \mathbf{S} is not truly random, it harbors biases which are made discreet by the multi-level use of \mathbf{S} . Also, as the variable i is known publicly, it can be used to garner information about the table \mathbf{S} which is why both variables i and j are used. Finally, an addition operation is performed that blends the whole words.

2.6 KSA – The Key Schedule Algorithm

As the PRGA, Pseudo Random Generator Algorithm, has no dependencies on a key, it leads to inconveniences that can be solved by an algorithm that can create an initial state which is dependent on a key. This is done by the Key Schedule Algorithm – by initializing the identity permutation to the function \mathbf{S} and setting j to zero. Here, the contents of \mathbf{S} are scrambled in a similar way to the state change function. The incrementation of the variable j is different though as a key byte is added to it at each round which cycles around the key length. Using this process, key lengths can be varied wildly. The key schedule function ends by setting both i and j as zero which is only a convention as any value from 0 through $N - 1$ would be acceptable.



Chapter 3

Attacks and Weaknesses

3. Attacks and Weaknesses

There has been significant amount of research done on RC4 since its inception all intrigued by its high level of security provided its simplicity of functions. As such, many new weaknesses and attacks have been discovered throughout the years and researchers have also come up with potential modifications to counter some of these. In this chapter the most prominent attacks on RC4 are discussed.

3.1 Attack Categories and Types of Weaknesses

Being a stream cipher, the primary objective of RC4 is to generate an output keystream which is random. However, practical constraints make it so that the keystream is only pseudo-random. So, if an attack is able to find patterns in the RC4 keystream it is considered to be a weakness. Also, some attacks may be able to reveal data on the key used, partial or complete, is a weakness. However, these attacks are more of theoretical purpose as they are only possible if the key used is small, which is not the case in practical situations. In this chapter the described attacks have been thoroughly analyzed and are computationally feasible for practical use. The types of attacks described are:

- i) Key Schedule Biases
- ii) Predictive States
- iii) State Recovery Attacks
- iv) Initialization Vector Weakness
- v) Other Attacks

3.2 Key Schedule Biases

Even a cipher with adequate security can be attacked by analyzing its key shuffling functions for related or weak keys. These attacks can be divided into multiple categories:

- i) **Weak keys** – A specific class of keys of the RC4 cipher leaves certain “traces” in the internal state space or the output keystream bytes during key scheduling. Among the keys in this class, a few of them generates very distinct patterns in the internal state or the output stream which are called the *weak keys*.
- ii) **Related Key Attacks** - If there are two keys which have a certain public connection between them, the resulting outputs from using those keys should not share any known connection. As the known information on the outputs may lead to brute force attacks as it leads to a lesser number of keys to search, thus decreasing the computation time for the key search. These attacks in general are known as *related key attacks*.
- iii) **Key Collisions** –During the key scheduling algorithm, it is possible that two different keys produce a similar state which will also result in output of similar streams. These key pairs are known to be “related key pairs” or *Key Collisions*. The attacker’s motive here is to produce such key collisions.
- iv) **Key Recovery from the State** – The round algorithm or PRGA (Pseudo Random Generator Algorithm) is reversible by design and is also one to one which grants it the ability to revert back to the initial state from any other state of the PRGA. But, as the KSA is not reversible and one to one, reversing its state to figure out the secret key is not simple. However, if this can be done efficiently, a state recovery attack can be transformed into a key recovery attack from state.
- v) **Key Recovery from the Keystream** – Biases in the output stream of RC4 can easily lead to discovery of its secret key. These biases are exploited in WEP and WPA.

3.2.1 Roos' Class of Weak Keys

Andrew Roos first publicly reported on a RC4 weakness on weak keys back in 1995 where he found multiple sets of weak keys which had very distinct patterns which resulted in specific weaknesses like biased output stream and output streams that leaked information about the used keys [3]. Roos analyzed the key scheduling algorithm to find a strong bias in the initial state that caused the the initial few output bytes to also be heavily biased. He computed his analysis considering $n = 8$ where the bias was evident for about $\frac{1}{256}$ of the keys which were the class of the weak keys.

The root of the bias is that there is a certain element in the state, which is indexed by the variable j during the KSA, has a probability of $\left(1 - \frac{255}{256}\right)^{256} = 0.63$, considering the values of the variable j are uniformly distributed. What this proves is that there is a single state element (index i) that has been swapped only a single time with the probability at 0.37. Roos showed that the probability decreases from 0.37 for values from values of i increasing from zero, however, compared to the uniform probability of only 0.004, these values show a huge bias. The solution to this weakness is to discard the first few output bytes.

3.2.2 Wagner' Class of Weak Keys

Inspired by Roos' work on weak keys, Wagner made his own research on the matter and extended the work by analyzing Roos' work and also discovering new weaks keys in 1995, the same year Roos' discovered the first weak keys [4]. Wagner used Roos' observation of one element i in a state table having 37% probability that it depends only on the elements of the key. Given the key $K[0], K[1] \dots K[255]$, there is one element i which is solely dependent on the key with a probability of 0.37. Wagner proposed a new problem – An attacker has public knowledge of the first 10 bytes, $K[0], K[1], \dots, K[9]$, of a key with 16 bytes. Is it possible for said attacker to guess the remaining bytes? Wagner's research showed that the knowledge of the first three bytes of the key, $K[0], K[1], K[2]$, enables the attacker to know about $K[10], K[11], K[12]$ with a very high rate of success. After which, the attacker can use brute force to recover the rest of the elements of the key with complexity now significantly down. However, this research is mostly of theoretical interest as the specific conditions for the test makes it improbable in real life.

3.2.3 Key Schedule Invariance

There has been more sophisticated work done on weak keys by Fluhrer et. Al. [14]. For this work they have modified the key schedule algorithm slightly. The figure below denotes the modified algorithm. This weakness is dependent on the transformation of the variable j in the algorithm.

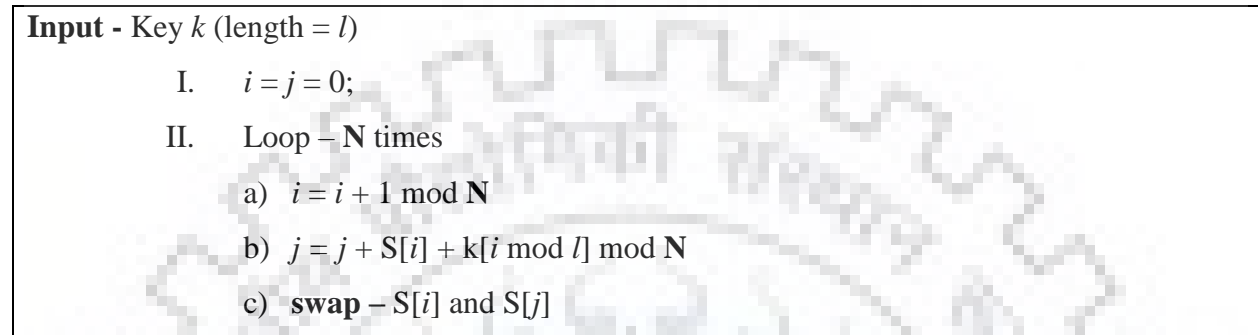


Figure 3.1: Modified KSA

The only difference with the modified algorithm to the original key schedule algorithm is that the variables i and j are not equal. This is repaired using a different key described by Fluhrer et al. as, “Let l and b be integers and let \mathbf{k} be a key of length l . If $\mathbf{k}[0] = 1$, the most significant bit of $\mathbf{k}[1]$ is 1, and $\mathbf{k}[i] \equiv i \pmod{b}$ for all $i \neq 0$, then \mathbf{k} is called a special b -exact key.” [14].

Using their method, they predicted outputs that were highly correlated with the real output for the initial bytes. They further tested the precision of the predicted outputs compared to the actual one for varying values of b . For $b = 2$, they predicted 20 bits of output with a probability of $2^{-4.2}$ which is significantly higher than the usual 2^{-20} . With b as 16, they predicted 40 bits of output with a probability of $2^{-2.3}$ compared to the usual 2^{-40} .

3.2.4 Weak Keys and Related Key Attacks

Fluhrer et. Al. [14] also worked on a related key attack based on a very specific problem. The problem states that if we are to implement RC4 in a black box which stores a secret key \mathbf{k}_s and it allows for only two operations. Operation 1: Requests the following word of the output stream. Operation 2: Reset machine with new key \mathbf{k}_n where $\mathbf{k}_n = \mathbf{k}_s + \Delta$, where Δ denotes the input. The goal is to attack in a way to extract information about the secret key. Fluhrer et. Al. [14] were able to

develop an attack like that which only required 2^{40} operations compared to 2^{256} for a brute force attack.

Given the test case, this attack is mostly of theoretical value as a practical situation like that is not likely to occur. Nonetheless, the weakness can be cured substantially by discarding the initial few output bytes.

3.2.5 Grosul and Wallach Related Key Pairs

The first discovery of key collisions was done by Grosul and Wallach in their paper “A Related Key Cryptanalysis of RC4” in 2000 where it was observed that when RC4 is initiated with a single 2048 bit key, there exists related keys pairs for which the output streams of the first hundred bytes are well correlated after which they deviate. However, when they tried their experiment with keypairs of 256 bytes, the attack was not useful which made them conclude that their attack is of only theoretical significance as usage of a practical key length in RC4 makes the attack moot.

3.2.6 Matsui Key Collisions

Matsui [12] was the first to propose a RC4 key collision which had practical implications in 2009, unlike preceding works of Grosul and Wallach [11] and others. In his method, the selected keys had a difference of a single byte which made the opening disparity in the internal state vanish by the end of the key schedule. He calculated the “key-pair distance” which is the byte difference between two states in a given round of the key schedule algorithm. Then an algorithm was designed to track the “key-pair distance” throughout the run time of the Key scheduling process and simultaneously build related key pairs such that the “key-pair distance” does not exceed two at any round of the KSA. Following this process, he was able to generate practical key collisions with the colliding key pairs had lengths of 24, 43 and 64 bytes.

3.2.7 Paul and Maitra Equation Solving Method

The concept of key recovery from state has been a relatively new topic and research on it was not considered for a long time after the inception of RC4. Paul and Maitra [13], in 2007, kickstarted the research interest in this area when they started working on key recovery by solving system modular equations. This study was built upon the work done by Roos [3] where the associations between key bytes and the bytes in the initial phase of the Round algorithm are shown. The results

found by Paul and Mitra [13] create an arrangement of integrated equations which are able to relate summations of successive key-bytes to each individual byte of the initial state of the Round algorithm. However, the results of Roos's work are probabilistic in nature which makes it so that the accuracy of the results for the bytes in the secret key have probabilistic dependency on the accuracy of the selected equations. Their work differs in this area as it does not depend on majority of the chosen equations being correct. They claim, based on results, that enough correct equations are obtained in most cases to find the correct key.

3.2.8 Attacks on WEP

At the beginning, the IEEE 802.11 standard used the WEP protocol which was based on RC4. As such, there has been thorough analysis on the topic which has inevitably exposed some weaknesses in the protocol.

The WEP protocol is designed to be resilient against packet loss during transmission so it encrypts each packet discretely. As the RC4 cipher does not support the use of an Initialization Vector (IV), for the WEP protocol to use it, it requires a 3-byte IV along with secret key to generate session keys of 64 to 128 bits for every packet. As a large percentage of the plaintext in the WEP protocol is constant and public along with various predictable bytes, a lot of plaintext-ciphertext pairs are available for a potential attacker. So, the most popular attack on WEP is to recover the secret key through the information obtained from the plaintext-ciphertext pairs.

In 2001, Fluhrer et. Al. [14] first conceived the idea of a related key attack on WEP. Their work was theoretical where they described that a success rate for WEP secret key recovery of 0.5 can be achieved working with around 4 million packets. However, this is based on the incremental Initialization Vector for consecutive packets. Later, a practical test was made on the topic was made by Stubblefield et. Al. [26] where they discovered the number of packets required was closer to 6 million.

3.2.9 Attacks on WPA

After the discovery of multiple weaknesses in WEP protocol, the IEEE802.11 standard switched to WPA protocol. In general, it is an embellishment on top of WEP as it has more elaborate key mixing features. To guard against the first attack on WEP by Fluhrer et. Al. [14], WPA was

equipped with key hashing function along with the old WEP design. It also avoids key reuse by using a “message integrity” feature alongside a scheme for key management. A temporal key is taken as input along with a transmitter address and a 48-bit IV which outputs the original RC4 key that does not repeat for 2^{48} packets. So, when this session key is obtained, the WEP algorithm is put in effect. This modification has made WPA lot more robust to the weaknesses of WEP.

Sepehrdad et. Al. and Vuagnoux [15, 16] worked on finding the first practical attack on WPA to find the secret key in 2011. They were successful by finding distinguishers with complexity of 2^{43} for WPA with a packet complexity of 2^{40} along with a probability of success of 50%. After which, they used a partial key recovery method to recover the complete temporal key used by WPA in the beginning using only 2^{38} packets. They calculated the time complexity to be around 2^{96} [15,16].

3.3 Predictive States and Distinguishers

The primary function of the RC4 cipher is to generate a random output keystream, however, practical constraints only allow for the cipher to generate “Pseudo-random” bytes of output keystreams only. As such, the output bytes often have patterns which can be detected with feasible computation prowess, and such patterns become the core weakness of the cipher as they can be used to significantly breach its security leaking important information.

Sometimes, the biases found in RC4 cipher is completely based on the keystream bytes which are known as distinguishers. The purpose of the attacks on this bias is to distinguish the RC4 generated “Pseudo-random” output keystream from a random keystream. R. J. Jenkins [5] found out that the output streams created by RC4 are slightly biased right after RC4 was publicly available in 1994. Mister and Tavares [6] later devised a gap length test in 1999 testing 230 elements of RC4 keystream where they also found biases.

More interesting and substantial work on the field was done by Mantin and Shamir in [10] and Fluhrer and McGrew in [8] where they have found new classes of streams that contain immutable biases in the output keystream. Their work area can be divided in two parts:

- i) Single key – Attacker is only aware of a single output keystream
- ii) Multiple key – The attackers knows the bytes in a fixed position for multiple keystreams

Working on single key distinguishers, J. Golic [7], in 1997, found the length of the keystream required to identify RC4 from a random keystream is about 2^{40} bytes. Later, Fluhrer and McGrew [8] improved on it by devising a method that requires only $2^{33.6}$ bytes of keystream to differentiate RC4 from a random sequence.

Mironov [9] worked on multiple key distinguishers in his paper “(Not so) Random Shuffles of RC4” where he calculated that if the attacked is aware of the first byte of multiple keystreams, it would require 1700 bytes to tell RC4 from a true random sequence. Mantin and Shamir [10] focused their work on the second byte position where they calculated that the value zero occurs twice as frequently that usual. Using that pattern, they devised a method which only required 200 bytes to be known in the second byte position to separate RC4 output from a random sequence of bytes.

Before describing their work, some technical definitions about RC4 predictive states coined by Mantin and Shamir [10],

Defintion 1: “An a -state is a partially specified state that includes values for i , j , and a -values in ‘ S ’ where values of ‘ S ’ does not need to be consecutive.”

Defintion 2: “If all states consistent with a given a -state cause the same output to be produced in the r^{th} position then the a -state is said to predict the r^{th} output.”

Defintion 3: “Let A be an a state. If there exists some r_1, \dots, r_b such that A predicts the outputs of rounds r_1, \dots, r_b , then A is called b -predictive.”

So, it can be said that when the b outputs are correctly projected using only a values in the state, it can said it is a b -predictive a -state. It does not require the b values to be consecutive or follow a -values for this hypothesis to be true. However, it only works when $b \leq a$.

3.3.1 Predictive State Applications

When state a and b are in close proximity, they cause biases to occur in the output. If RC4 is unbiased (which it is not) b outputs should occur with \mathbf{N}^{-b} probability in necessary positions. However, it is known that the states compatible with a -state produces the same result. There are $(a + 2)$ constraints in a -state which accounts for \mathbf{N}^{-a-2} of total states.

In their influential work, “A Practical Attack on Broadcast RC4”, Mantin and Shamir [10] describes “fortuitous attacks” as a subclass of predictive states. These attacks predict the first a -outputs and are known as a -predictive a -states. They assessed that given $n = 8$, the time needed for the attack to complete is around $2^{755.2}$ compared to the original attack time of 2^{779} which was predicted by Knudsen et al [18].

This attack was further analyzed and honed by Paul and Preneel [27], in 2003, where they were able to prove that the “Fortuitous States” account for most of the predictive states. For instance, given $n = 8$ there will be 297 3 -predictive 3 -states of which 290 will be fortuitous which is why modifying the attack to comply general predictive states would not result in noteworthy enhancement in the running time.

3.3.2 Distinguisher Using Digraphs

In 2000, Fluhrer and McGrew [8] developed a distinguisher based on biases in consecutive pairs of bytes called digraphs. They did so by calculating amount of consistent states with specific digraphs. This test was only possible if the value of N is small, but the results indicated biases that are independent of N . They found 12 digraphs that continue till the end whose probabilities differ from the usual probability. These digraphs occur irrespective of N . Of the 12 digraphs found, 2 were predicted by partial states which had dependencies on i and j and had 3 specified values in S and were 2 -predictive 3 -state. Further 8 digraphs were found to have a positive bias which belonged to class 2 -predictive 2 -state and were also shown to have dependency on i and j . The final 2 digraphs occur with less frequently and so are not considered predictive states but rather they belong to a general pattern which produces a particular output.

Using a similar technique, Golic [7] was able to calculate the output stream length to be known to successfully distinguish RC4 output from a random sequence to be around $2^{44.7}$ for $n = 8$. Fluhrer and McGrew [8] were able to decrease that complexity to around $2^{30.6}$ output bytes for $n = 8$. They tried to further improve this value but results based on smaller values of n suggest that the improvements would be insignificant. For instance, with n valued at 5, it would require $2^{18.76}$ keystream bytes using only the 12 digraphs which improves to $2^{18.62}$ by using all the digraphs. They also contemplated on working with trigraphs but the huge computational load made it impractical at the time.

3.3.3 Broadcast Attack Using Second Byte Bias

One of the most influential work on RC4 distinguishers has been done by Mantin and Shamir [10] in their paper “A Practical Attack on Broadcast RC4” in 2001. Here they found the strong bias of having almost double the occurrence of zero in the second position (Z_2) of the output stream. This bias is simple in nature but the observation required elegance and after its discovery it has led to a new dimension in the analysis of security of RC4. Using this bias, RC4 output stream can be told apart from a random sequence with a very competent complexity of $O(N)$.

Before explaining the bias in detail, a description of Theorem 1 in the words of Mantin and Shamir would suffice:

Theorem 1: “(Bias in second output word). *If the initial state of RC4 satisfies $S[2] = 0$ and $S[1] \neq 1$, then the second output word will be 0.*”

From the theorem, it is obvious the bias described is not a predictive state as the state $\mathbf{S}[1]$ is not specified as a value but rather just a constraint on which value it cannot be. The bias on $\mathbf{S}[2]$ can be calculated as there is only a single restriction on the state. So, for $\frac{1}{N}$ states, the output is 0 and for the other $1 - \frac{1}{N}$ states, the output will be zero with a probability of $\frac{1}{N}$. Combining the probabilities, it can be seen that it accumulates to $\frac{2}{N}$ which is double than the usual occurrence chances.

Using this bias initial state information can be extracted by guessing predictive states with an of accuracy of $\frac{1}{2}$. In this way, about n bits of data can be collected which is not the most effective attack but can be used in conjunction with Knudsen’s attack [18] to speed it up.

It has a far more effective usage as a “strong distinguisher” where Mantin and Shamir [10] claims that by analyzing the second byte for about 200bytes of the output stream, they are able to reliably separate RC4 output from a random sequence which is leaps ahead Fluhrer and McGrew’s [8] complexity of $2^{30.6}$ in the first byte position.

This bias enabled Mantin and Shamir to attack RC4 on a broadcast level. The exact plaintext was broadcasted to multiple receivers using various random keys for encryption. During this, the second byte of the plaintext was correctly retrieved with the knowledge of only $\Omega(N)$ ciphertexts.

This attack has practical implications, like email broadcasting, where encryption of the same exact message may be done with multiple keys.

3.4 State Recovery Attacks

When considering a brute force attack on RC4, it is wise to consider that the state space of RC4 is about $N! \times N^2$. $N!$ is the N Byte S array permutation space and N^2 is the all possible pairs of values. So, if for instance, $N = 256$, the state space of RC4 becomes $256! \times 256^2$ which is equal to 2^{1700} . This incredible size requires incredible amount of computation power, far out of the reach of practical computational power currently and this is the reason state recovery attacks are a very challenging task.

3.4.1 Knudsen's Attack

Knudsen et al [18] developed an algorithm that significantly decreases the required time to predict the internal state of RC4. This algorithm is based on two observations.

Observation 1: In RC4's internal state S , which contains N bytes, indices i and j are fixed at zero. The index i indicates to all the elements of the internal state in the first N outputs which makes the output reliant on each of the elements of the internal state. As such, this attribute can be used to devise a method to recover the internal state given there are N words of output.

Observation 2: The pseudo random generator algorithm of RC4 does not use complete knowledge of the internal state at any one point. At max, in any given round, it is able to use only three elements of the internal state – $S[i]$, $S[j]$ and $S[S[i]+S[j]]$. So, it is not required to process the complete internal state at once for any given output.

The algorithm developed is a branch and bound type which, with partial or no information about the internal state, guesses the remaining elements of the state until the values correspond with the output stream. It uses an output stream, c_t , as its input.

Input – Partial output stream c_0, c_1, c_2

- I. Initialize elements of S as unassigned
- II. $i = j = z = 0$;
- III. Loop
 - a. $i = i + 1$;
 - b. If, $S[i] =$ not assigned –
 - i. do, branch all values of $S[i]$
 - c. $j = j + S[i]$;
 - d. If, $S[j] =$ not assigned –
 - i. do, branch all values of $S[j]$
 - e. **Swap:** $S[i] \& S[j]$;
 - f. $t = S[i] + S[j]$;
 - g. If, $S[t] =$ not assigned & c_z is = not assigned in S
 - i. set $S[t] = c_z$;
 - h. If, $S[t] \neq c_z$, (Contradiction)
 - i. Close branch;
 - i. $z = z + 1$;
 - j. If, $z =$ output length;
 - i. Terminate operation
 - ii. Output S

Figure 3.2: RC4 Branch and Bound attack

Using this method, Knudsen et al.[18] were able to recover the internal state (for $n = 8$) in around 2^{779} steps. The typical brute force method would require 2^{1684} steps.

It is worth mentioning that Mister and Tavares [28] also developed a very similar model for the same purpose of recovering the internal state of RC4 during the same time but independently. They added kind of state tracker in their model which used the cycle structure of RC4. This structure worked better for smaller versions of RC4, say, for instance, when $N = 32$ it would recover the entire internal state in 2^{42} steps.

3.5 IV weaknesses

A special class of key schedule weaknesses in RC4 are known as IV (Initialization Vector) weakness. These are known information strings which can be used to extract a session key from a secret key. A session key makes it so that a keystream is not used twice. This weakness has led to various practical attacks on RC4 and is the only weakness that allows for the secret key to be extracted when n is set at 8.

There are many ways to connect two information strings to gain another string some of which expose relation between the session and secret keys. One of the ways is to concatenate the initialization vector (IV) with the secret key. Alternative way is to XOR the information strings with each other. These key relations require strong key schedule application for information not to be leaked, but, the RC4 key schedule is not up to the mark for this task. This weakness in the Key schedule in RC4 is used by Fluhrer et al. [14] to extract the secret key in their paper “Weaknesses in the key scheduling algorithm of RC4”. As such, a better method to keep the key relations from extracting information is to use cryptographic hash functions, also suggested by Fluhrer et al. [14] who worked out that using this technique leads to complete immunity from information leakage.

3.6 Other Attacks

Few other prominent attacks on RC4 are described in this section.

3.6.1 Mironov’s Analysis

After Mantin and Shamir’s [10] broadcast attack on RC4 on the second byte bias, it has become common practice to discard the initial few output bytes. In the paper, “(Not so) Random Shuffles of RC4”, Mironov [9] analyses exactly how much data need to be dropped for RC4 to be not vulnerable against broadcast attacks.

For the analysis, Mironov uses the usual key schedule setup for RC4 that contains an exchange shuffle which can be used to generate a random permutation if a random source of information is present. It works almost like a normal KSA only difference being with the element in the second byte position, which is actively chosen based on the key and the current state.

He describes two distinguishers using this “exchange shuffle”.

Distinguisher 1: Uses a concept called “Sign of a permutation” which is described as, “The parity of the number of transpositions in a representation of the permutation as a product of transpositions” by Mironov. In the exchange swap, each swap is transposition unless two indices are the same, chances of which are next to nothing. This creates a bias that can be quite significant reaching a probability of 0.05 for $N=256$. However, for this to work the complete information on the permutation is required beforehand which makes it difficult to use.

Distinguisher 2: A more useful weakness, to figure it out, Mironov first found that the chances of a specified value popping up in a specific place is not consistent for the “exchange shuffle” as it should be if it were truly random. This bias is calculated to be accurate to 60% for P_{256} . Mironov uses this bias to prove the bias in the first byte position of the output stream of RC4. The chances that a word is actually part of the output depends on $S[S[1] + S[S[1]]]$ being that word. This value is biased due to positional biases in the initial state which results in this distinguisher. For $N = 256$, this distinguisher can differentiate RC4 output from a random sequence by using the first 1600 bytes of the output stream in the first position. Based on this calculation, Mironov suggests the initial $3N$ bytes should be discarded from the output stream to avoid this weakness.

3.6.2 RC4’s Special States – The Finney States

Immediately after the public disclosure of RC4 in 1994, Finney [29] worked out a special class of states in his work titled, “An RC4 cycle that can’t happen”. Generally, RC4 has a huge internal state which corresponds to it having large cycle lengths as well. The Finney states are special because a bunch of short cycles are attached with them.

If a partial state is defined by, $i = a, j = a + 1$ and $S[a + 1] = 1$, the next state after one round would be $i = a + 1, j = a + 2$ and $S[a+2] = 1$. So, it can be seen that if the progression is allowed naturally, all the following states would be of same arrangement. For each subsequent round, all values are incremented except the $S[a+1]$ which becomes $S[a]$. As such, after $N - 1$ operations, all the entries return to their original positions, which creates a short cycle recurring after $N(N - 1)$ rounds.

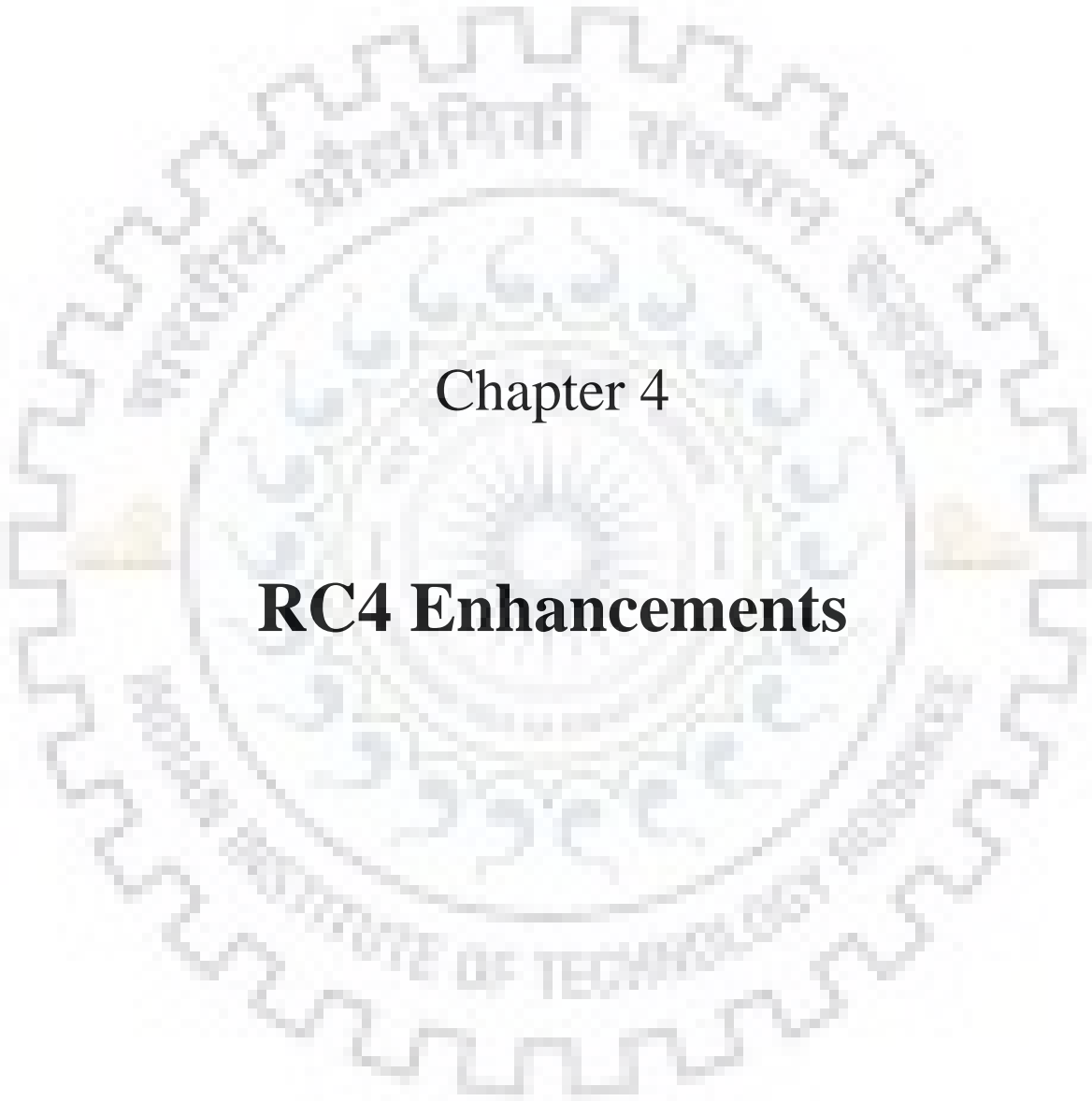
3.6.3 Golic’s Distinguishers

Golic [7] published a different kind of attack on RC4 early in its life. This technique uses linear analysis which says that it was not built for RC4 specifically but rather for attacking LFSRs and

block ciphers. The attack is based the attribute of RC4 that its S permutation progresses gradually where each new step of **S** is closely approximated by S immediately before the swap. Golic uses linear equation to predict the most insignificant bit of S. These two processes tie in to relate the least significant bit of an output byte and the successive byte. Using the bias, Golic requires to know 240 bytes of the output sequence to differentiate RC4 output from a random byte sequence.

3.6.4 Paul and Preneel's Distinguishers

Influenced by the Mantin and Shamir on the second byte bias [10], Paul and Preneel [27] analyzed the digraph frequencies of the first two bytes of RC4 output streams. They identified a bias against equal bytes. The premise of the bias is that if in the initial state $S[1]$ is equal to 2, then the first two output bytes are never same. So, the first 2 bytes of the output stream cannot be equal for $\frac{1}{N}$ of the time and not equal the rest of the time. So, it can be calculated that the probability is about $\frac{(1-\frac{1}{N})}{N}$, less than the usual probability of $\frac{1}{N}$. The authors found that to detect the bias, it takes around 2^{24} streams.



Chapter 4

RC4 Enhancements

4. RC4 Enhancements

In this chapter various new variants of RC4 are discussed which are produced by researchers over the years with enhancements for specific weaknesses.

4.1 RC4A

S. Paul and B. Preneel described a new variant of RC4 in 2004 called RC4A in their paper “A New Weakness in the RC4 Keystream Generator and an Approach to Improve the Security of the Cipher”. They studied the weaknesses of RC4 cipher and to augment the security, they proposed the use of two S-boxes, S_1 and S_2 , instead of one, and called the new model – **RC4A** [30].

In RC4A, the key scheduling algorithm is as in the usual RC4 cipher except that the use of two S-boxes now requires the use of two separate secret keys. In the pseudo random generator algorithm, the variables j_1 and j_2 are used to update S_1 and S_2 respectively, throughout the shuffle exchanges. The alterations done is in the index $S_1[i] + S_1[j]$ where it is evaluated on S_1 but generates output from S_2 , and vice-versa.

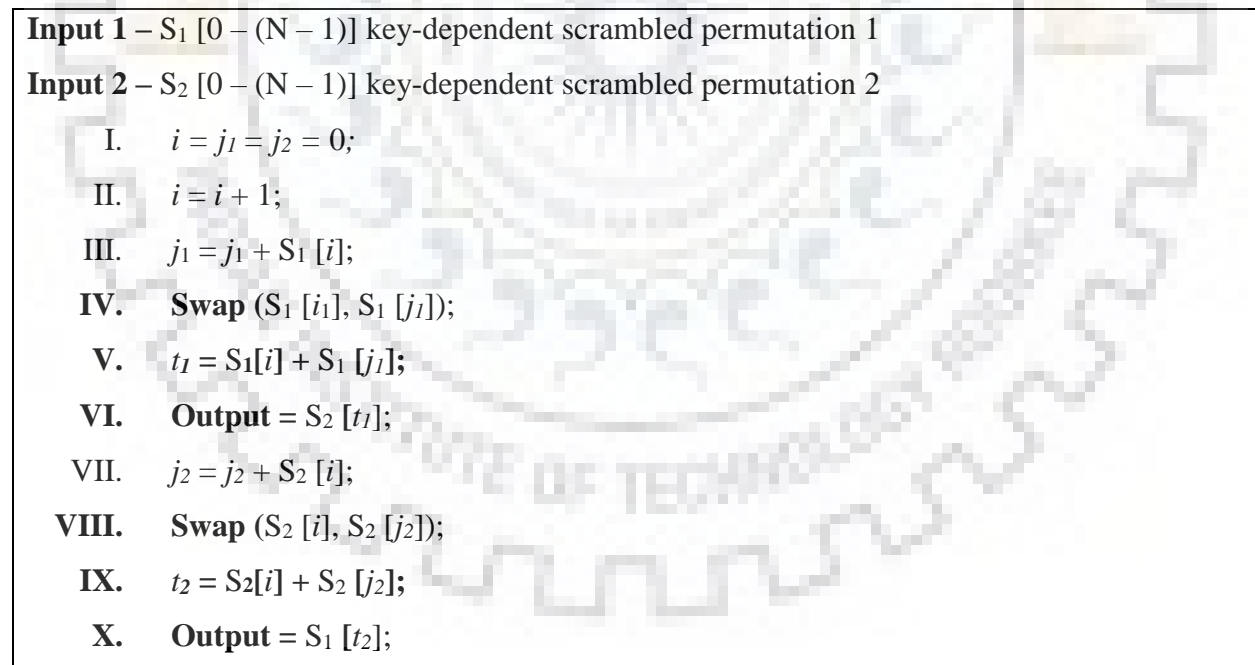


Figure 4.1: RC4A PRGA

RC4A improves on RC4 by requiring less CPU cycles to generate the output as it only needs to increment the index i once to produce two successive output bytes.

Distinguishing attacks have been produced for RC4A by A. Maximov [31] which requires to know 2^{58} output keystream bytes to be able to tell RC4 output from a random string. This was subsequently improved on by Y. Tsunoo et al. [32] who were able to decrease the length needed to decipher RC4 to 2^{23} bytes only.

4.2 RC4B

Developed by M. E. McKague in his master's thesis "Design and Analysis of RC4-like Stream Ciphers" in 2005 to increase the overall security of RC4 and called it RC4B. The concept is to replace one of the input streams of RC4 and change it to something else. For RC4B, McKague used an N-cycle to replace the variable i . It constituted an algorithm which had a state change function that is dependent on the key which makes it more secure as it is now able to hide every information on its internal state.

With the change in input, RC4B requires a modified Key schedule algorithm as to RC4. The algorithm is given in the below figure. The algorithm was modified not only to be able to adjust for the change in input, but also to be more secure. The internal state in the new KSA is shuffled two times to decrease chances of creating "Fortuitous States" as described by Mironov [9].

Input 1 - Key k (Length l_1)

Input 2 - IV v (Length $l_2 = l_1 - 1$)

- I. for, $i = 0$ through $N - 1$,
 - a. $S[i] = i$;
- II. $j = 0$;
- III. for, $i = 0$ through $N - 1$,
 - a. $j = j + S[i] + k[i \bmod l_1] \bmod N$
 - b. **Swap** ($S[i]$ & $S[j]$);
- IV. $t = S[0]$;
- V. for, $i = 1$ through $N - 1$,
 - a. $K[t] = S[i]$;
 - b. $t = S[i]$;
- VI. $K[t] = S[0]$;
- VII. $j_0 = S[j]$;

```

VIII.  for,  $i = 0$  through  $N - 1$ ,
        a.  $j_0 = S[j]$ ;
        b.  $j = j + S[j_0] + v[i \bmod l_2] \bmod N$ ;
        c. Swap  $S[j_0]$  &  $S[j]$ ;
IX.     $i = S[j]$ ;

```

Figure 4.2: RC4B KSA

The second function used in RC4B is almost identical to RC4 with the difference being the change of incrementing i to incrementing N -cycle K in its place.

```

I.      $i = K[i]$ ;
II.     $j = j + S[i] \bmod N$ 
III.   Swap ( $S[i]$  &  $S[j]$ )
IV.    Output  $S[S[i] + S[j] \bmod N]$ 

```

Figure 4.3: RC4B PRGA

4.3 VMPC

The VMPC cipher, which can be described a generalization of RC4, was developed by Zoltak [34] at FSE in 2004 at the same time as RC4A. In general, it changes the output function and the j update to essentially increase security. The name VMPC comes from “Variably Modified Permutation Composition”. The KSA of VMPC converts a secret key into a permutation of S and initializes a variable j . The VMPC function used for the cipher can be defined as:

$$S[S[S[x] + 1]].$$

Input 1 – Array of Secret key, $K[0...(l-1)]$

Input 2 – Initialization Vector Array, $IV[0...(l-1)]$

```

I.     for  $i = 0$  through  $N - 1$ 
        a. do,
        b.  $S[i] = i$ ;
        c.  $j = 0$ ;
II.    end;
//Key Scrambling Phase

```



```

III.  for  $m = 0$  through  $3N - 1$ 
      a.  do,
      b.   $i = m \bmod N$ ;
      c.   $S[j + S[i] + K[m \bmod l]]$ ;
      d.  Swap ( $S[i], S[j]$ );
IV.   End;
      // Scrambling with IV
V.    for  $m = 0$  through  $3N - 1$ 
      a.  do,
      b.   $i = m \bmod N$ ;
      c.   $j = S[j + S[i] + IV[m \bmod l' ]]$ ;
      d.  Swap ( $S[i], S[j]$ );
VI.   End;

```

Figure 4.4: VMPC Key Scheduling Algorithm

This function is used in VMPC PRGA to output keystreams and update S . The goal here is to restrict information leakage of the internal state S . The way the algorithm is set up creates interruptions in the exchange cycles forcing attackers to predict every state of the internal state in order to extract information.

Input 1 - $S[0 \dots (N - 1)]$ // A key-dependent scrambled permutation

Input 2 – Variable j //Key Dependent

```

I.     $i = 0$ ;
II.   Loop //output key generation
      a.   $j = S[j + S[i]]$ ;
      b.   $t = S[i] + S[j]$ ;
III.  Output  $z = S[S[j] + 1]$ ;
IV.   Swap( $S[i], S[j]$ );
V.     $i = i + 1$ ;

```

Figure 4.5: VMPC PRGA

VMPC is resilient to first byte bias and digraph frequencies. However, distinguisher attacks are possible and was tested by Maximov [31], who calculated it requires 2^{54} known keystreams of

output to properly identify VMPC output from a random sequence. Later, Tsunoo et al. [32] devised a more proficient distinguisher attack that only needed 2^{38} known keystreams to decipher VMPC from a random stream.

4.4 NGG

NGG, named after the initials of the creators, was developed to build RC4 to 32 or 64 bits while making the state space much smaller than 2^{32} or 2^{64} [35]. At first, the algorithm was known as RC4(n, m) where N is the size of the state array and is 2^n and m is the word size in bits. NGG and RC4 internal functions, KSA and PRGA, has similarities such as the increment process of the variables i and j . The difference in KSA being the internal state S is initialized to a preprocessed array a which is random. Later, $S[i]$ and $S[j]$ are swapped and the sum of these elements are assigned to $S[i]$.



Figure 4.6: NGG KSA

There are differences in PRGA of both ciphers as well. In the NGG PRGA, a pseudo random element is sent to the output and then it is modified by the summation of two other elements of the internal state S .

Input 1 - $S[0 \dots (N - 1)]$ // A key-dependent scrambled permutation

- I.** $i = j = 0;$
- II. Loop** //output key generation
 - a. $i = (i + 1) \bmod N;$
 - b. $j = (j + S[i]) \bmod N;$
- III. Swap**($S[i], S[j]$);
- IV. Output** $z = S[(S[i] + S[j]) \bmod N];$
- V.** $S [(S[i] + S[j]) \bmod N] = (S[i] + S[j]) \bmod N;$

Figure 4.7: NGG PRGA

4.5 GGHN

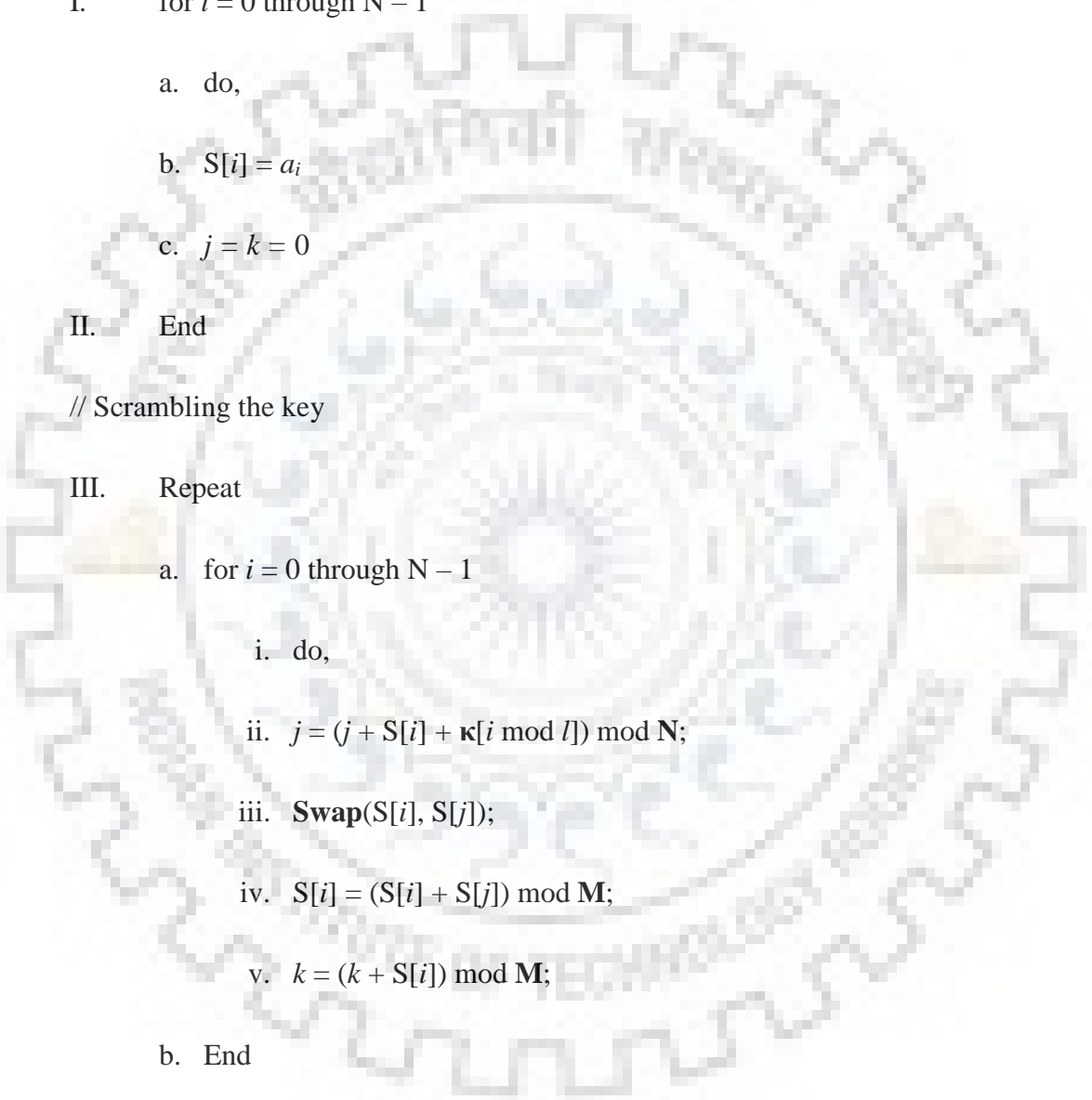
GGHN, which is also named after the initials of its creators, is another version of the previously described NGG cipher. The main difference between them being an added index, denoted by k , is used to enhance the security levels. This new index is initialized in the key schedule function of the algorithm which is also made to be key dependent. The key schedule function loops a certain number of time which is denoted by r which is dependent of certain parameters (n, m) . So, for instance, if n is 16 and m is 64, then r would be 48.

Input 1 – Array of Secret key, $\mathbf{K}[0\dots(N - 1)]$

Input 2 – Random Permuted Array, $\mathbf{a}[0\dots(N - 1)]$

Output 1 – A scrambled array $\mathbf{S}[0\dots(N - 1)]$

Output 2 – A secret variable \mathbf{k} which is key dependent



```
I.   for  $i = 0$  through  $N - 1$ 
      a. do,
      b.  $S[i] = a_i$ 
      c.  $j = k = 0$ 
II.  End
// Scrambling the key
III. Repeat
      a. for  $i = 0$  through  $N - 1$ 
          i. do,
          ii.  $j = (j + S[i] + \kappa[i \bmod l]) \bmod N$ ;
          iii. Swap( $S[i], S[j]$ );
          iv.  $S[i] = (S[i] + S[j]) \bmod M$ ;
          v.  $k = (k + S[i]) \bmod M$ ;
      b. End
IV.  Repeat for  $r$  iterations.
```

Figure 4.8: GGHN KSA

The PRGA is used identically as like NGG cipher – to update the internal state S and to disguise the output.

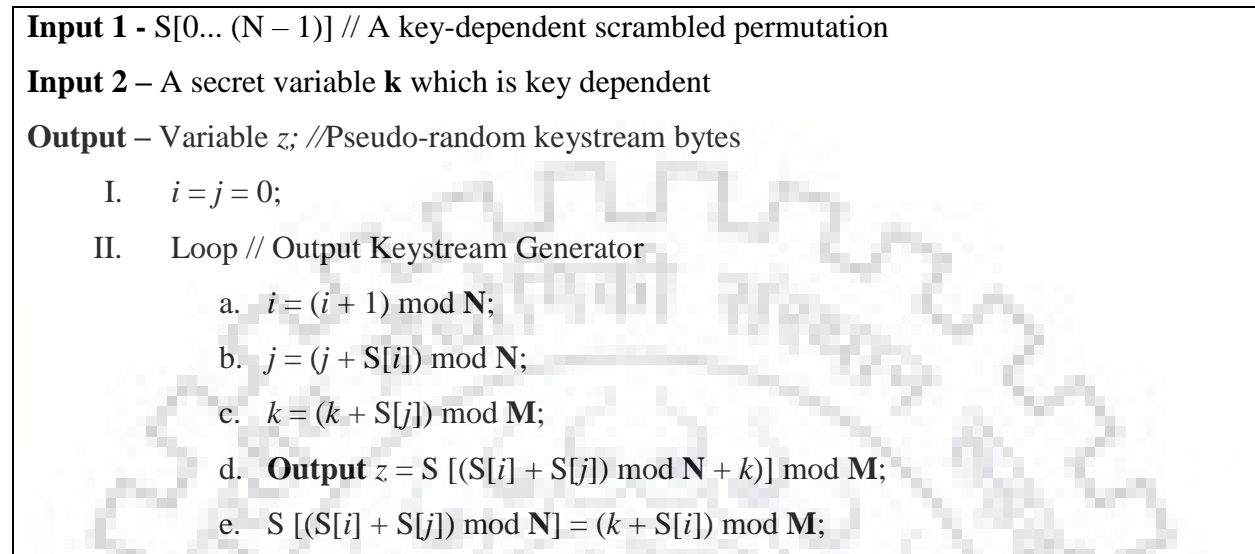
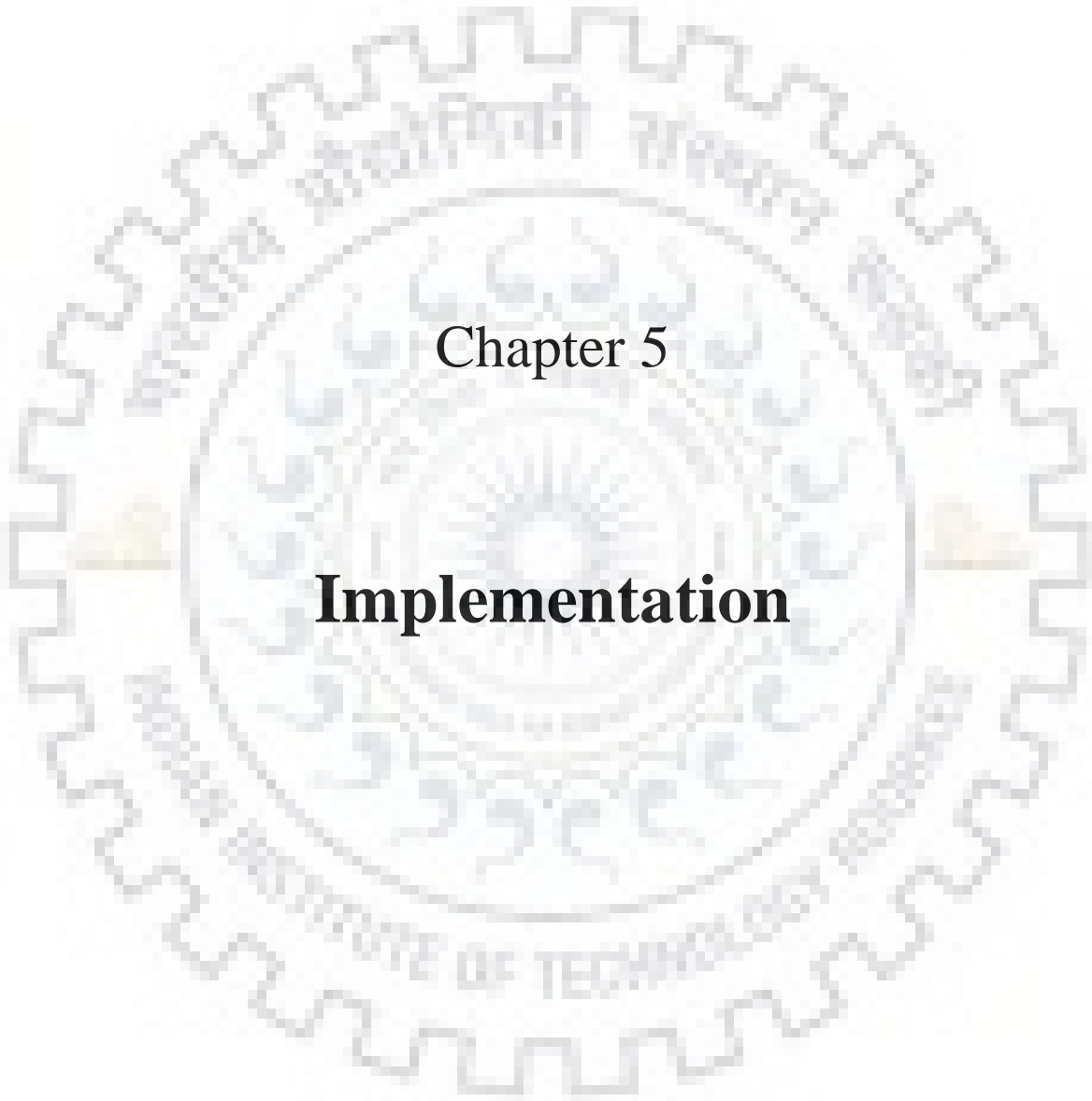


Figure 4.9: GGHN PRGA

H. Wu, in his work, “Cryptanalysis of a 32-bit RC4-like Stream Cipher”, devised a distinguisher attack on GGHN cipher which only required the information on 100 initial keystream words [37]. Y. Tsunoo et al. also devised a distinguisher attack on GGHN based on biases in the first- and second-byte positions, which required the information of around 2^{30} keystream bytes [32].



Chapter 5

Implementation

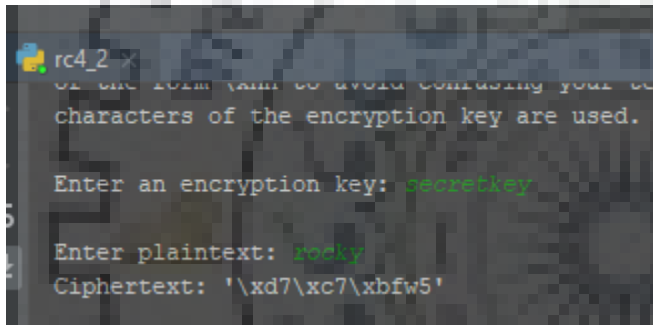
5. Implementation

In this chapter we show some implementations on the RC4 cipher and its variants. A general implementation of the basic RC4 code and how its encryption and decryption works is shown in the beginning, followed by a comparison study between RC4 and a modified version of it is shown.

5.1 RC4

5.1.1 Basic Encryption

Given below is a basic encryption function using RC4. Here the key K used is “secretkey” and the plaintext P is “rocky”. Using these two inputs, the algorithm is able to generate a ciphertext “\xd7\xc7\xbfw5”.



```
rc4_2
of the form (aim to avoid confusing your
characters of the encryption key are used.

Enter an encryption key: secretkey
Enter plaintext: rocky
Ciphertext: '\xd7\xc7\xbfw5'
```

Figure 5.1: A Secret key and plaintext merged to create a ciphertext

For this implementation, the word size n is set at 8. This results in an internal state space of $2n$, i.e., 256. So, the S-box array $S[]$ is set at 256 state spaces – $S[0], S[1], S[2], \dots, S[253], S[254], S[255]$.

By tradition, the S-box array is initialized with $S[n] = n$, so $S[13] = 13, S[253] = 253$ and as such.

Then the plaintext and secret key are transformed in to ASCII form and if the length of secret key is less than the state space, then the key is padded by repeating itself until it is of length 256 bytes.

index	0	1	2	3	4
plaintext	r	o	c	k	y
ASCII	114	111	101	107	121

Table 5.1: Plaintext ASCII conversion

Index	0	1	2	3	4	5	6	7	8
key	s	e	c	r	e	t	k	e	y
ASCII	115	101	99	114	101	116	107	101	121

Table 5.2: Key ASCII conversion

index	0	1	2	3	4	5	6	7	8	9	10	11	12..
key	s	e	c	r	e	t	k	e	y	<i>s</i>	<i>e</i>	<i>c</i>	<i>r..</i>
ASCII	115	101	99	114	101	116	107	101	121	115	101	99	114..

Table 5.3: Repeating key patterns to fill 256 state spaces

After the S-boxes are initialized and the key is converted and padded, the permutation part of the algorithm takes over. The key first goes through the Key Scheduling Algorithm for the first set of swapping. Then it goes in to the Pseudo Random Generator Algorithm where it is permuted again and then generated as a pseudo random keystream.

index	0	1	2	3	4	5	6	7	8	9	10	11	12	13
Pseudo Random key	“	Ć	e	o	}	z	o	g	Ć	y	r	o	w	o
ASCII	126	127	101	111	125	122	111	103	127	121	114	111	119	111

Table 5.4: Keystream generated after input key is permuted through KSA and PRGA

For this test, the output stream length was set to 14, so the plaintext had to be padded too.

Index	0	1	2	3	4	5	6	7	8	9	10	11	12	13
Padded plaintext	r	o	c	k	y	r	o	c	k	y	r	o	c	k
ASCII	114	111	101	107	121	114	111	101	107	121	114	111	101	107

Table 5.5: Padded plaintext with pattern repetition to fill 14 bytes.

After the pseudo random keystream is generated, the ASCII Codes of the keystream and padded plaintext are generated. These ASCII codes are then binarized. Then the binary values of the plaintext and keystream are XORed to find the ciphertext as a binary value. That binary value is transformed into decimal to find the ASCII Code of the ciphertext. Then gathering the corresponding symbols of the decimal values from the ASCII table generates the Ciphertext.

Index	Plaintext	ASCII	Binary
0	r	114	01110010
1	o	111	01101111
2	c	101	01100101
3	k	107	01101011
4	y	121	01111001
5	r	114	01110010
6	o	111	01101111
7	c	101	01100101
8	k	107	01101011
9	y	121	01111001
10	r	114	01110010
11	o	111	01101111
12	c	101	01100101
13	k	107	01101011

Table 5.6: The padded plaintext is converted to ASCII code first and then binarized.

Index	Pseudo-Random Keystream	ASCII	Binary
0	“	126	01111110
1	ć	127	01111111
2	e	101	01100101
3	o	111	01101111
4	}	125	01111101
5	z	122	01111010
6	o	111	01101111
7	g	103	01100111
8	ć	127	01111111
9	y	121	01111001
10	r	114	01110010
11	o	111	01101111
12	w	119	01110111
13	o	111	01101111

Table 5.7: The pseudo random keystream is converted to ASCII code first and then binarized.

Plaintext (Binary)	XOR	Pseudo Random Keystream (Binary)	Ciphertext (Binary)	Ciphertext (ASCII-decimal)	Ciphertext
01110010	⊕	01111110	01011000	92	\
01101111	⊕	01111111	01111000	120	x
01100101	⊕	01100101	01100100	100	d
01101011	⊕	01101111	00000111	7	7
01111001	⊕	01111101	01011100	92	\
01110010	⊕	01111010	01111000	120	x
01101111	⊕	01101111	01100011	99	c
01100101	⊕	01100111	00000111	7	7
01101011	⊕	01111111	01011100	92	\
01111001	⊕	01111001	01111000	120	x
01110010	⊕	01110010	01100010	98	b
01101111	⊕	01101111	01100110	102	f
01100101	⊕	01110111	01110111	119	w
01101011	⊕	01101111	00000101	5	5

Table 5.8: Forming the final ciphertext.

This is a simple encryption function in RC4 and the decryption function can be done by XORing the ciphertext with the keystream.

5.1.2 Processing Time on Varying Key Lengths

The RC4 cipher's primary function is the randomization of its secret key. The two internal functions both serve this purpose. The secret key goes through KSA for the sole purpose of being scrambled. Then, the scrambled value goes to PRGA where it goes through another level of randomization before the pseudo random keystream is generated. So, this processing of the secret key takes the highest level of computation power for RC4. So, this test was devised to check the difference in processing times for increasing key sizes.

Test	Secret Key Length	Processing Time (Seconds)
1	5	0.6987
2	10	0.7003
3	15	0.7080
4	20	0.7067
5	30	0.7131
6	40	0.7192
7	50	0.7281
8	60	0.7359
9	70	0.7431
10	80	0.7498
11	100	0.7612
12	120	0.7758
13	160	0.7913
14	200	0.8133
15	256	0.8338

Table 5.9: The change in processing time based on increasing Secret Key Length

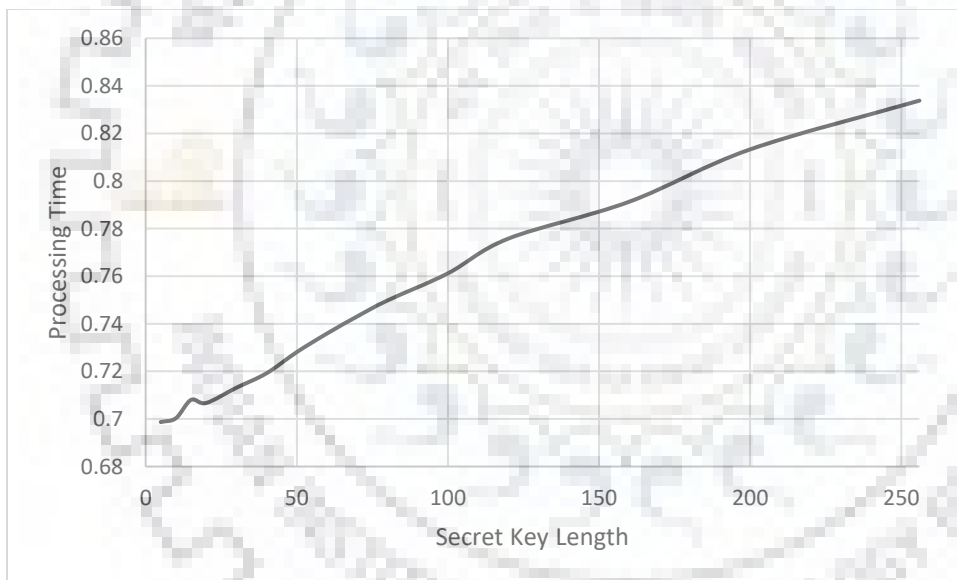


Figure 5.2: Processing time versus increasing secret key length.

```

Secret Key=
012345678910111213141516171819202122232425262728293031323334353637383940414243444546474849505152535455565758596061626364656667686970717273747576777879808182838485868788
899091929394959697989910010110210310410510610710810911011111211311411511611711811912012112212312412512612712812913013113213313413513613713813914014114214314414514614714
814915015115215315415515615715815916016116216316416516616716816917017117217317417517617717817918018118218318418518618718818919019119219319419519619719819920020120220320
42052062072082092102112121231241251261271281292202122223224225226227228229230231232233234235236237238239240241242243244245246247248249250251252253254255
-----
fc2f59ccfd18747213b0edc34f1e44e2db63d958fe01c8ee881ceeb5ad57a21f9b4e0f0a2e4ec5994916b25cebd90c70e8fc4c53a44c918cbb6dd6433da7082cb5efcfdde07a16e99d5a918d97cf26dab4661
971cf7323846d4aafba22fc390c48acb858797e69d941192f7e28ad1a636e0f4916f7bcf1d8dea346c8fc97edcebc7fa04a2837a447d1a4d8393d71a6d1e567dfe1b91a5a82533b175631bebb33c0fa668491ee7
537daf12cddb307c26cb6139df447f1933ce67adddf0598d873bc820e9a79e78ca70863f69efb44433839ae9c75bfcff8d47371c27c429a962042f7ba58bd8e1d9a0b73f8c94ca9c8e575771cbb7c79f419612
9783778c83b7f0ad9d30fe5ab66b534c2e185c8c30a4e44c4db52e482c09fc24bece92df81e939f58a4ee7bce64e78298ec5b4fedc2d341fc4ab4888a7d94980bd4d09f178a942341ec82a1564ee246e175f755c
e641196c609d6fa5d6bea21ca37c9a83a6adb62f92f45460c4fd8437b12e9fb86764dc1e1edcdd57199ddfbcf1f9ce4229e0ffe958d7374f52ef56b3d9afb28d565bdfb3aad126ef1b6ef2189e361e41d79c0e23
a5a5969ab99621cd6149862d192bf7cb114f4fc0f258c21726584a884dd1b24bf56ca1daf846e5ff7d72762f290f1492fe2a73c5f5dc51ba2a9c75a559fb643cc58af77af74f17438f9d201e46239e39ca34614
57d23e493c31a80cb7c47bf4da1da446c432a88d8916eb65e1156218f1d90dec9c328cf24527e6f3d437625b49277c2a77496f69a1d59c10d9acaf1bd9b38c6dd2cec2fc7d253e9cd969aadb5c3a2e6957bb12
c4bb7110c12df2a85bcd9c678f5cf9ea5233e9be927d4512c22b9cfaaed733f20ba75ff641efcb92366cc4579d3297027985be7af4aa65ec44c28ef3364bab1682ba2c7dbce4f047acfb041e5b9213e3fc7d7
62d22b9efaf69db67fd3747c9f4775bc5947fb83a6afc3ad7aee88a29df9204bf72efdaa89cea6d4a4b042f4f2d967e3f2ad0a1957d8246fbf17a3f9a1d1e322b523303757ab1529cd79423ab718ec346552260
983be366998a51a392fe74edb9a7b58dc583752e1fcffff136a87dc07ab089c46ec7bdcf2a4a2314fec4edd6d64ebce0daac469eadf33b49ed97fcfec467443a8c4920d4e4da899c5ab5b63fa9a4a3d63e0807c
35c998f660cedb77354b5e858de419e7da6b69ef475b426df65f4e127d17b75b68b2d68d1fd8b6dd84970597f47ff71e8f3da583665e4c6313aeb03db7327c108c9c71be228326fca05c9b5551b73b2f3f5ef3e65
2cdcacb648ade30661a79255814b26cc2aecd91d01e9ca9ec2f812d121698dfa9e49d7c421ca14575968873c0e2b3a16b335195f3f1233ca402a7298eba0c39c10fa
-----
Process exited after 0.8338 seconds with return value 1025
Press any key to continue . . .

```

Figure 5.3: Encryption using the maximum sized secret key – 256 bytes.

5.2 RC4 vs RC4 – Fact

As previously described in this thesis, there have been many researches done on RC4 to figure out its weaknesses and many people have found many weaknesses and corresponding attacks for it. As a consequence, there have been various modifications done on it as well, which have also been described previously. One of the more recent modifications done on RC4 was by A. M. Sagheer et al. [38] which is more subtle than others. Published on December 2016, the new model adds factorial functions in both KSA and PRGA to increase complexity by enhancing key randomization but also decreases computation time compared to RC4.

5.2.1 Algorithm Modifications

Here, the modified KSA and PRGA of RC4 – Fact with the factorial functions are shown.

INPUT – variable x //Key

- I. For $x = 0$ through 255,
 - a. $S[x] = x$;
- II. For $x = 255$ through 0,
 - a. For $c = 1$ through x ,
 - i. $S_Fact[x] = (S_Fact[x] * c) \bmod 256$
 - b. $y = 0$;
- III. For $x = 0$ through 255,
 - a. $y = (S_Fact[x] + S[x] + Key[x \bmod l]) \bmod 256$
 - b. **Swap** ($S[x], S[y]$)
- IV. **Output:** $S[x]$

Figure 5.4: Modified KSA for RC4 – Fact

INPUT 1 - $S[x]$

INPUT 2 – x

- I. $x = y = 0$;
- II. Loop //Output generation
 - a. $x = (x + 1) \bmod 256$
 - b. $y = (S [(y + S[x]) \bmod 256]) \bmod 256$
 - c. **Swap**($S[x], (S-Fact [y] \bmod 256)$)
 - d. $Z = (S[(x + y) \bmod 256] + S[(y + S[S[x]]) \bmod 256]) \bmod 256$
 - e. Key sequence = $S [Z]$
- III. Output: Key sequence

Figure 5.5: Modified PRGA for RC4 – Fact

5.2.2 Comparison Between RC4 and RC4 – Fact

The test devised is to check for the computation time required for both the algorithms to perform a complete encryption cycle with varying key lengths.

Test	Secret Key Length	Processing Time (Seconds)
1	5	0.7087
2	20	0.7203
3	40	0.7380
4	70	0.7467
5	100	0.7531
6	130	0.7692
7	160	0.7781
8	190	0.7959
9	220	0.8331
10	256	0.8598

Table 5.10: RC4 runtime with varying key sizes

Test	Secret Key Length	Processing Time (Seconds)
1	5	0.6298
2	20	0.6409
3	40	0.6577
4	70	0.6656
5	100	0.6847
6	130	0.6984
7	160	0.7198
8	190	0.7277
9	220	0.7356
10	256	0.7467

Table 5.11: RC4 – Fact runtime with varying key sizes

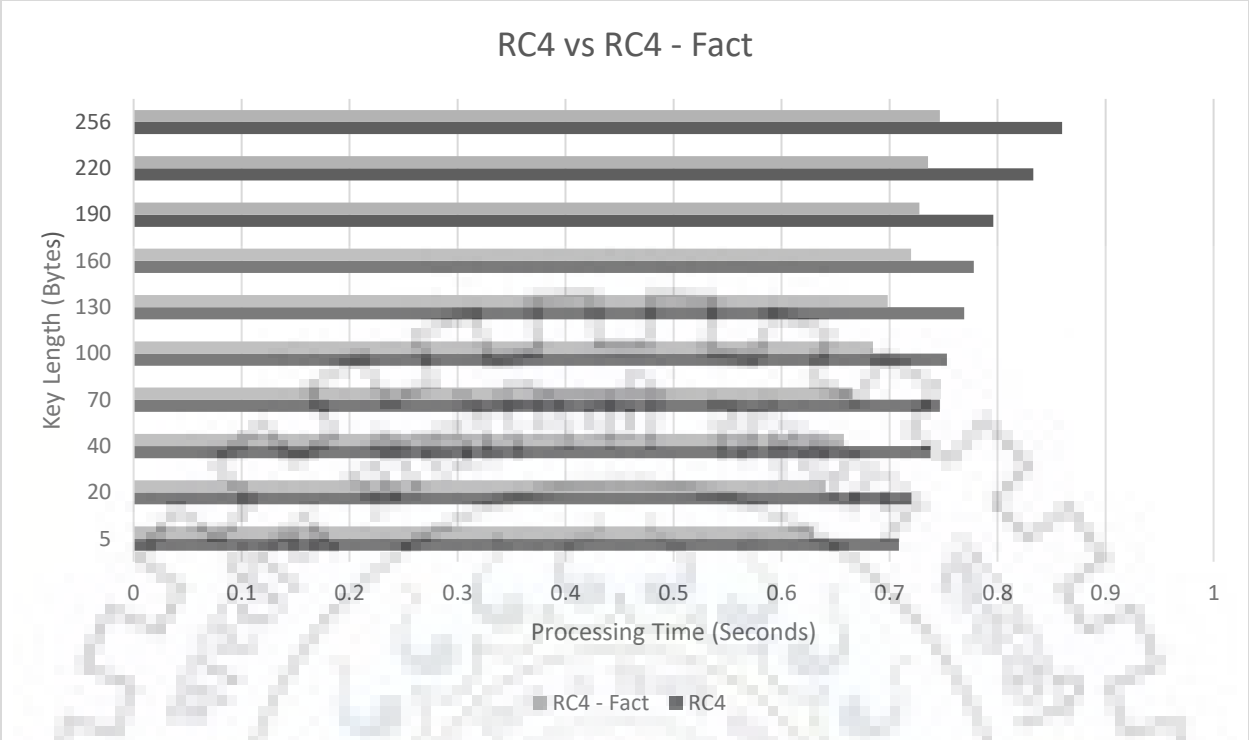
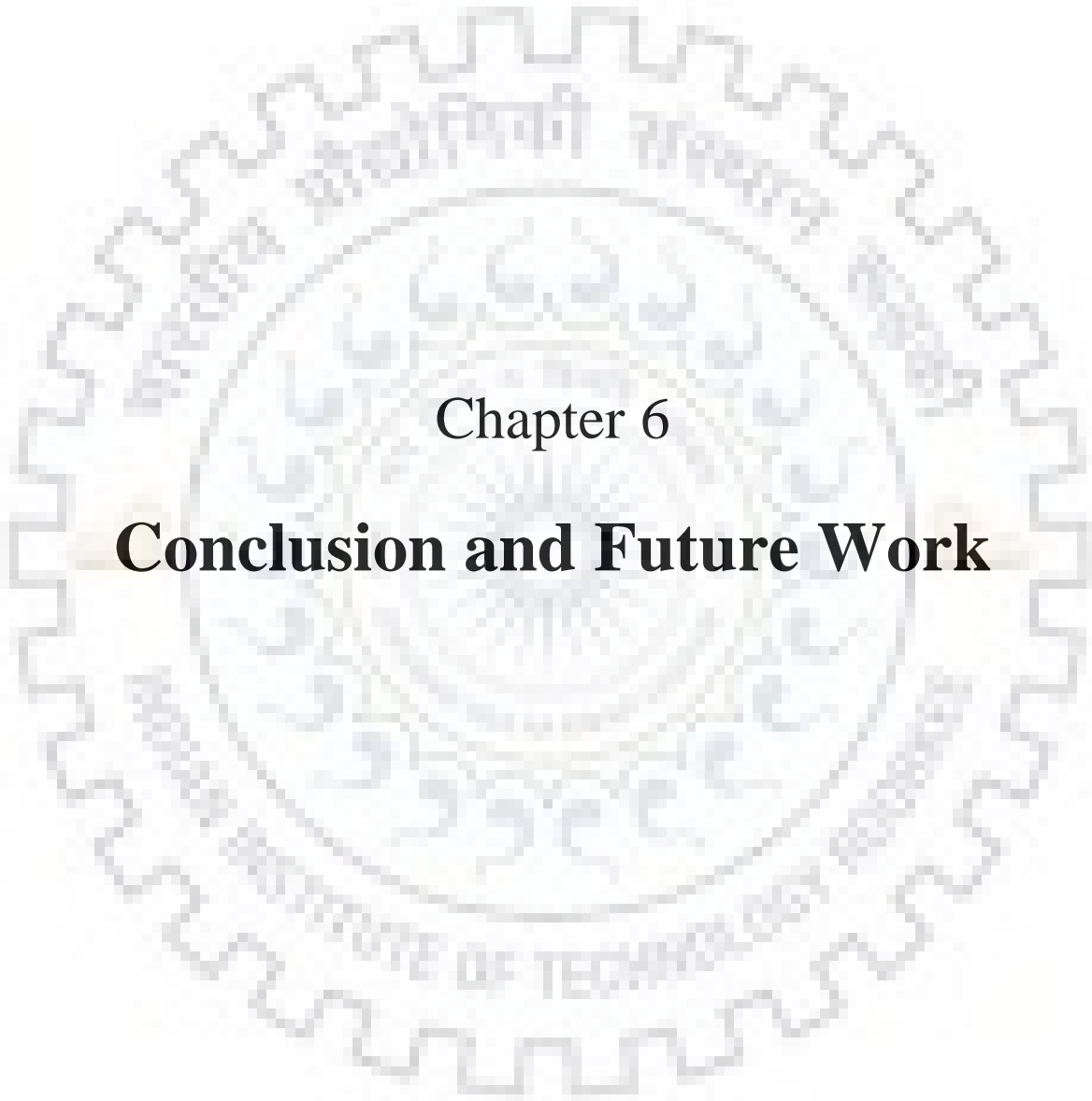


Figure 5.6: Comparison of processing time for a complete encryption cycle between RC4 and RC4 – Fact.

The results show a consistent decrease in the runtime of RC4 – Fact compared to RC4 over increasing key sizes indicating a more efficient model.



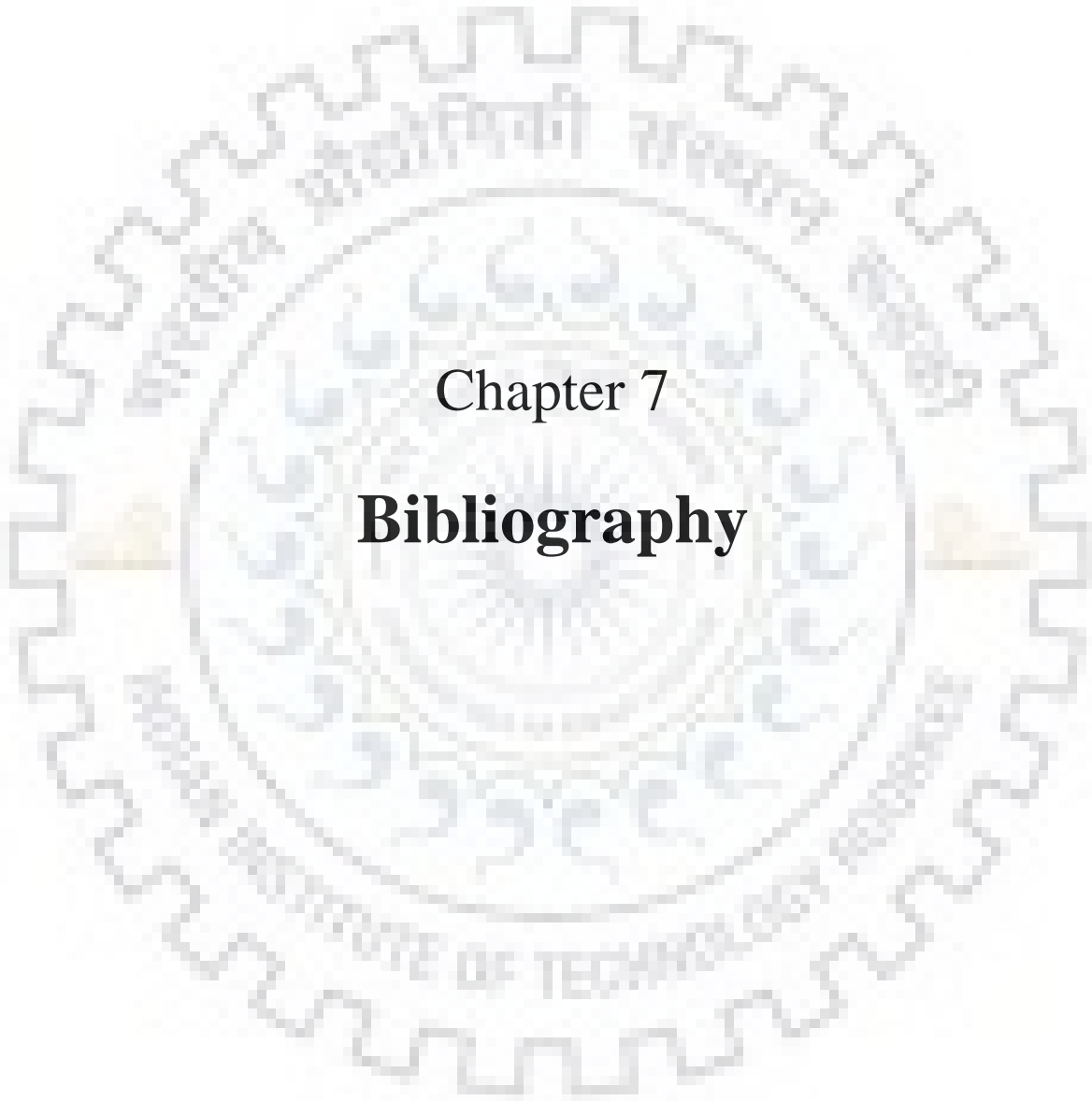
Chapter 6

Conclusion and Future Work

6. Conclusion and Future Work

RC4 has been in use for a long time and it is a testament to its accessibility and corresponding security. However, with time and ever-increasing devotion from researchers, multiple leaks have been found and the number seems to be increasing. As a result, there have been many modifications and variants proposed and produced till now. It is essential to keep up the cat and mouse game of finding weaknesses and its subsequent adjustment, as continuing this cycle would lead to more and more secure ciphers while keeping the elegantly simple core of RC4 alive.

However, something new is arising from the horizon of the field of computation that not only threatens RC4 but modern data security as a whole – Quantum Computing. This enables never before seen computation power which blows our current model of computing completely out of the water. Theoretical work has been going on for a while with recent researchers have found applied success as well, though superficial. So, it seems inevitable that the cryptographic model has to be revamped according to the conventions of quantum computing – the sooner the better.



Chapter 7

Bibliography

Bibliography

- [1] C. Shannon. *Communication Theory of Secrecy Systems*. Bell System Technical Journal. 28 (4): 656–715, 1949.
- [2] V. Tilborg, C. Henk. *Encyclopedia of Cryptography and Security*. Springer. ISBN 978-1-4419-5905-8., p. 455, 2011.
- [3] A. Roos, *Class of weak keys in the RC4 stream cipher*. Two posts in sci.crypt, 1995.
- [4] D. Wagner, *My RC4 weak keys*. Posting in sci.crypt, 1995.
- [5] R. J. Jenkins, *Re: RC4?*, posting to sci.crypt, 1994.
- [6] S. Mister and S. Tavares, *Cryptanalysis of RC4-like Ciphers*, Proceedings of SAC'98, LNCS vol. 1556, pp. 131-143, Springer-Verlag, 1999.
- [7] J. D. Golic, *Linear Statistical Weakness of alleged RC4 keystream generator*, Advances in Cryptology - Eurocrypt'97, LNCS vol. 1233, pp. 226-238, Springer-Verlag, 1997.
- [8] S.R. Fluhrer and D.A. McGrew, *Statistical Analysis of the Alleged RC4 Keystream Generator*, Proceedings of Fast Software Encryption 2000 FSE'00, LNCS vol. 1978, pp. 19-30, Springer-Verlag, 2001.
- [9] I. Mironov, *(Not so) Random Shuffles of RC4*, Advances in Cryptology, Lecture Notes in Computer Sci., vol. 2442, Springer, Berlin, pp. 304–319. 2002.
- [10] I. Mantin and A. Shamir, *A Practical Attack on Broadcast RC4*, Proceedings of Fast Software Encryption, FSE 2001, to appear in LNCS, Springer-Verlag, 2002.
- [11] A. I. Grosul and D. S. Wallach, *A Related Key Cryptanalysis of RC4*, Manuscript from Department of Computer Science, Rice University, June 2000.
- [12] M. Matsui. *Key collisions of the RC4 stream cipher*. In Orr Dunkelman, editor, FSE, volume 5665 of Lecture Notes in Computer Science, pages 38–50. Springer, 2009.

- [13] G. Paul and S. Maitra. *Permutation after RC4 key scheduling reveals the secret key*. Selected Areas in Cryptography, volume 4876 of Lecture Notes in Computer Science, pages 360–377. Springer, 2007.
- [14] S. R. Fluhrer, I. Mantin, and A. Shamir, *Weaknesses in the key scheduling algorithm of RC4*. Selected Areas in Cryptography, volume 2259 of Lecture Notes in Computer Science, pages 1–24. Springer, 2001.
- [15] P. Sepehrdad, *Statistical and Algebraic Cryptanalysis of Lightweight and Ultra-Lightweight Symmetric Primitives*. PhD thesis No. 5415, École Polytechnique Fédérale de Lausanne (EPFL), 2012.
- [16] P. Sepehrdad, S. Vaudenay, and M. Vuagnoux. *Statistical attack on RC4 - distinguishing WPA*. Eurocrypt, volume 6632 of Lecture Notes in Computer Science, pages 343–363. Springer, 2011.
- [17] J. Golic. *Iterative probabilistic cryptanalysis of RC4 keystream generator*. ACISP, volume 1841 of Lecture Notes in Computer Science, pages 220–233. Springer, 2000.
- [18] L. R. Knudsen, W. Meier, B. Preneel, Vincent Rijmen, and Sven Verdoolaege. *Analysis methods for (alleged) RC4*. ASIACRYPT, volume 1514 of Lecture Notes in Computer Science, pages 327–341. Springer, 1998.
- [19] A. Menezes, P. van Oorschot, and S. Vanstone, *Handbook of Applied Cryptography*, CRC press, 2001.
- [20] I. Mantin, *Analysis of the stream cipher RC4*. Master’s thesis, The Weizmann Institute of Science, 2001.
- [21] D. Bernstein, *ChaCha, a variant of Salsa20*, 2008
- [22] D. Bernstein, *Snuffle 2005: the Salsa20 encryption function*, 2005
- [23] C. Berbain, *Sosemanuk, a Fast Software-Oriented Stream Cipher*. New Stream Cipher Designs. Lecture Notes in Computer Science, vol 4986. Springer, Berlin, Heidelberg, 2008.
- [24] B. Guido, *The Road from Panama to Keccak via RadioGatún*, 2009.

- [25] R. Anderson, C. Manifavas, *Chameleon — A new kind of stream cipher*. Fast Software Encryption. FSE 1997. Lecture Notes in Computer Science, vol 1267. Springer, Berlin, Heidelberg, 1997.
- [26] A. Stubblefield, J. Ioannidis, and D. Rubin, *Using the Fluhrer, Mantin, and Shamir attack to break WEP*. In NDSS, The Internet Society, 2002.
- [27] S. Paul and B. Preneel, *Analysis of non-fortuitous predictive states of the RC4 keystream generator*. Progress in Cryptology— INDOCRYPT 2003, Lecture Notes in Comput. Sci., vol. 2904, Springer, Berlin, pp. 52–67, 2005.
- [28] S. Mister and S. Tavares, *Cryptanalysis of RC4-like ciphers*. Selected Areas in Cryptography, Lecture Notes in Comput. Sci., vol. 1556, Springer, Berlin, pp. 131–143, 1999.
- [29] H. Finney, *An RC4 cycle that can't happen*, Post in sci.crypt, September, 1994.
- [30] S. Paul and B. Preneel. *A New Weakness in the RC4 Keystream Generator and an Approach to Improve the Security of the Cipher*. FSE 2004, pages 245–259, vol. 3017, Lecture Notes in Computer Science, Springer, 2004.
- [31] A. Maximov. *Two Linear Distinguishing Attacks on VMPC and RC4A and Weakness of RC4 Family of Stream Ciphers*. FSE 2005, pages 342– 358, vol. 3557, Lecture Notes in Computer Science, Springer, 2005.
- [32] Y. Tsunoo, T. Saito, H. Kubo, M. Shigeri, T. Suzaki and T. Kawabata. *The Most Efficient Distinguishing Attack on VMPC and RC4A*. SKEW, 2005.
- [33] M. McKague. *Design and Analysis of RC4-like Stream Ciphers*. Master's Thesis, University of Waterloo, Canada, 2005.
- [34] B. Zoltak, *VMPC One-Way Function and Stream Cipher*. FSE 2004, pages 210–225, vol. 3017, Lecture Notes in Computer Science, Springer, 2004.
- [35] Y. Nawaz, K. Gupta and G. Gong. *A 32-bit RC4-like keystream generator*. Technical Report CACR 2005-19, Center for Applied Cryptographic Research, University of Waterloo, 2005.

[36] G. Gong, K. C. Gupta, M. Hell and Y. Nawaz. *Towards a General RC4Like Keystream Generator*. CISC 2005, pages 162–174, vol. 3822, Lecture Notes in Computer Science, Springer, 2005.

[37] H. Wu. *Cryptanalysis of a 32-bit RC4-like Stream Cipher*. IACR Eprint Server, eprint.iacr.org, number 2005/219, July 6, 2005.

[38] A. Sagheer, S. Searan, R. Alsharida. *Modification of RC4 algorithm to increase its security by using mathematical operations*. 2016.



RC4

ORIGINALITY REPORT



PRIMARY SOURCES

etd.uwaterloo.ca Internet Source	4%
Maitra, . "Variants of RC4", Discrete Mathematics and Its Applications, 2011. Publication	3%
www.jseis.org Internet Source	1%
www.cosic.esat.kuleuven.be Internet Source	1%
uwspace.uwaterloo.ca Internet Source	1%
Submitted to University Tun Hussein Onn Malaysia Student Paper	<1%
en.wikipedia.org Internet Source	<1%
Poonam Jindal, Brahmjit Singh. "Optimization of the Security-Performance Tradeoff in RC4 Encryption Algorithm", Wireless Personal	<1%