Tribhuvan University

Institute of Science and Technology

Performance Analysis of Stream Cipher RC4 Variants: VMPC & SPRITZ

Dissertation

Submitted to:

Central Department of Computer Science and Information Technology

Tribhuvan University, Kirtipur, Nepal

In partial fulfillment of the requirements

For the Master's Degree in Computer Science & Information Technology

By

Santosh Sharma

Mar 18, 2018

Tribhuvan University

Institute of Science and Technology

Performance Analysis of Stream Cipher RC4 Variants: VMPC & SPRITZ

Dissertation

Submitted to:

Central Department of Computer Science and Information Technology

Tribhuvan University, Kirtipur, Nepal

In partial fulfillment of the requirements

For the Master's Degree in Computer Science & Information Technology

By

Santosh Sharma

Mar 18, 2018

Supervisor

Mr. Nawaraj Paudel

Tribhuvan University

Institute of Science and Technology

Central Department of Computer Science and Information Technology

Declaration

I, **Santosh Sharma**, declare that the thesis entitled "**Performance Analysis of Stream Cipher RC4 Variants: VMPC & SPRITZ"** contains no sources other than listed, this thesis is my own work.

……………………………………….

Santosh Sharma

Mar 18, 2018

Tribhuvan University

Institute of Science and Technology

Central Department of Computer Science and Information Technology

Supervisor's Recommendation

I hereby recommend that this dissertation prepared under my Supervision by **Santosh Sharma** entitled **"Performance Analysis of Stream Cipher RC4 Variants : VMPC & SPRITZ"** in partial fulfillment of the requirements for the Master's Degree in Computer Science & Information Technology be processed for the evaluation.

………………………………………………………………

Mr. Nawaraj Paudel

Head of Department (HOD)

Central Department of Computer Science & Information Technology

Kritipur, Kathmandu, Nepal

(Supervisor)

Date: …………………

Tribhuvan University

Institute of Science and Technology

Central Department of Computer Science & Information Technology

<u>LETTER OF APPROVAL</u>

We certify that we have read this dissertation and in our opinion it is satisfactory in the scope and quality as a dissertation in partial fulfillment of the requirements for the Master's Degree in Computer Science & Information Technology.

Evaluation Committee

……………………………	………………………….
Mr. Nawaraj Paudel	**Mr. Nawaraj Paudel**
Central Department of Computer Science and Information Technology	Central Department of Computer Science and Information Technology
Kritipur, Kathmandu, Nepal	Kritipur, Kathmandu, Nepal
(HOD)	**(Supervisor)**

…………………………..	………………………
(External Examiner)	(Internal Examiner)

Date: …………………

ACKNOWLEDGMENTS

It's a pleasure for me to thank my Supervisor, Mr. Nawaraj Paudel, Head of Central Department of Computer Science & Information Technology, Tribhuwan University, Kirtipur Kathmandu, Nepal, for his constant encouragement, support and advices.
I greatly acknowledge to respected Professors and Lecturers: Prof. Dr. Shashidhar Ram Joshi, Prof. Dr. Subarna Shakya, Mr. Dheeraj Kedar Pandey, Mr. Sarbin Sayami, Mrs. Lalita Sthapit, Mr. Jagdish Bhatta, Mr. Arjun Singh Saud, Mr. Bishnu Gautum, Mr. Bikash Balami, Mr. Yog Raj Joshi & Mr. Tej Bahadur Shahi of CDCSIT-TU, for providing valuable suggestions and huge knowledge as well as inspirations. I would like to thank my friends and family for their encouragement and support. I would like to give my special thanks to my friends Mr. Chhetra Bahadur Chhetri & Mr. Dil Bahadur Budhathoki for helping to provide necessary resources to complete this work.

ABSTRACT

Stream cipher algorithms are most powerful tools in symmetric cryptography. These algorithms perform either bit wise or byte wise encryption in a simple way just doing XOR operation between key and message (plain text). Stream cipher algorithms are about 5 to 10 times faster than AES, TDES (block cipher). In stream cipher, creating key stream by randomizing the bits is most important thing. These algorithms are useful normally in GSM mobile communication, Hard disk encryption, Multimedia encryption and fast Software encryption, standard web application, network protocols etc. In this thesis, stream cipher RC4 variants VMPC & SPRITZ are studied, analyzed the performance and implemented in Java Programming using NetBeans 8.0.2 considering their other parameters constant.

The empirical performance shows that VMPC cipher is found to be better if the message size is bigger. With small size of message stream cipher RC4 variants VMPC & SPRITZ shows worst performance. The speed of stream cipher SPRITZ is much slower due to cryptographic function "sponge function", where different functions are used for the additional layer of security in RC4 variants, although performance of SPRITZ also increases with increment of size of message. While the message size gets increased, performance of stream cipher VMPC is better in all size of messages .Therefore, while inputting different and big size of message, performance of stream cipher VMPC gets increased and it is found to be the better algorithm for the large size message in the targeted architecture computer.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF ABBREVIATIONS

AES	Advanced Encryption Standard
ANSI	American National Standards Institute
API	Application Programming Interface
CSPRNGs	Cryptographically Secure Pseudorandom Number Generators
DES	Data Encryption Standard
ECRYPT	European Network of Excellence in Cryptology
FSM	Finite State Machine
GSM	Global System for Mobile communication
IDE	Integrated Development Environment
IV	Initialization Vector
JVM	Java Virtual Machine
LANs	Local Area Networks
OTP	One Time Pad
PKI	Public Key Infrastructure
PRGA	Pseudo-Random Number Generation Algorithm
PRNGs	Pseudo Random Number Generators
RC4	Ron's Cipher Four
RSA	Rivest - Shamir - Adleman
SHA	Secure Hash Function
SSL	Secure Socket Layer
SQL	Structure Query Language
TDES	Triple Data Encryption Standard
TLS	Transport Layer Securities
TRNGs	True Random Number Generators
VMPC	Variably Modified Permutation Combination
WEP	Wired Equivalent Privacy
XOR	Exclusive OR

Chapter 1
INTRODUCTION

1.1 RC4 Variants:

The cryptographic stream cipher RC4 was designed by Ronald L. Rivest in 1987 for RSA data security. RC4 was first trade secret but algorithm was leaked & published in 1994 [15]. It has been said that Rc4 was the most widely used stream cipher in the world. The simplicity in design & extremely fast has been hit in the software industry for decades & has been adopted as the core cipher in numerous web & software application like Microsoft Windows, Lotus Notes, Apple OCI, Oracle secure SQL, web & network protocol such as SSL (Secure Socket Layer)/TLS (Transport Layer Security) , WEP(Wired Equivalent Privacy) etc.
The main factor in RC4 [9] success over such a wide range of application have been its speed simplicity & efficient implement in both software & hardware were very easy to develop. Apart from its popularity in commercial uses it has also become one of the most involved topics of research for cryptologists because it felt to different cryptanalysis [8][1]. Then different stream cipher RC4 variants has been proposed such as RC4A [19], VMPC [3], RC4+ [18], & SPRITZ [14].

VMPC (Variably Modified Permutation Composition) is a stream cipher variant of RC4. The core of the cipher is VMPC one way function [3], which is a combination of triple permutation composition & integer addition. The cipher generate a stream of 8-bit value from an internal state consists of 256-byte permutation two 8-bit integer variable. The generated values should be X-ored with the plain text to derive the ciphertext. The initial value of cipher's internal state is determined by the VMPC Key Scheduling Algorithm. PRGA (Pseudo-Random Number Generation Algorithm) of VMPC has more resistance against the different attack applied to RC4.This encryption is suitable for encryption of files & folder, encrypted book, encryption of emails, easy encrypted on-line conversation, unrecoverable wiping of files from disk with up 99 rounds [21].

RC4 like stream cipher "SPRITZ" [14], variant of RC4, attempts to repair weakness of RC4 while remaining true to its general design principles. SPRITZ provides additional cryptographic capabilities, such as hashing, derived from its reformulation with sponge like construction [5].It is intended SPRITZ to be replacement of RC4,hence the design of SPRITZ was chosen with special attention given to the fact that known weakness of RC4[9][10],it can be used.

1.2 Motivation

RC4 stream is developed by Ronald - L- Rivest in 1987 for RSA data security. It was the most widely used stream cipher in the world .It is simplicity in design & extremely fast has attracted everyone towards this cipher. Apart from its popularity in commercial uses it has become one of the most involved topics of research for cryptologists because it felt cryptanalysis.[8][1].

Then the different RC4 variants' has been proposed such as VMPC, RC4+, & SPRITZ [14].Among them VMPC (Variably Modified Permutation Composition) is presented in 2004 at fast software conference [2] & SPRITZ is presented in ramp session of CRYPTO 2014[14].Performances of these algorithms are depending upon number of parameters. One can easily ask the question what the performance of these techniques will be if different sizes of message are input to them considering other factors constant. This question is the sign of motivation in this thesis that the performance of the algorithms is observed by inputting small to large (variable) sizes of textual message.

1.3 Problem Definition

Although weakness [9] [1] discovered in RC4 make it less ideal choice of new application, the novel internal structure, continue to make it interesting topic of research while RC4 is still usable(with care using variants). The algorithm is 25 years old & deserves retirement & replacement by newer, stronger variants.

After the disclosure of its algorithm in 1994, RC4 has attracted intensive cryptanalytic efforts over past 20 years. Finally, in 2013, practical plaintext recovery attacks on RC4 in SSL/TLS were proposed [9] [1]. In the response to these results, usage of RC4 has drastically decreased, especially in TLS, and major companies such as Google, Microsoft, and Mozilla announced that they will officially remove the RC4 from web browsers by early 2016.

At the same time, there has been extensive research in recent years to come up with RC4-like stream ciphers that while marginally slower in software, would fulfill the weakness of RC4. Many such stream ciphers like RC4A [19], VMPC [3], RC4+ [18], SPRITZ [14], variants of RC4 have been proposed.

Time is the key factor for encryption as well as decryption in cryptography. So that, the better & fastest algorithm among these two variants of RC4 stream cipher with good performance will be purposed in this study .For this performance analysis is required to select the better approach stream cipher.

1.4 Objective

The objective of this dissertation work is:

- ❖ To implement and analyze the performance of stream cipher RC4 variants VMPC & SPRITZ.
- ❖ To perform the Cycle/Byte calculations.

1.5 Thesis Organization

The rest of the content in this thesis is organized into subsequent five chapters. Chapter 2 provides background study required for dissertation. In this chapter the problem of different stream cipher algorithms are mentioned, problem statement is formulated and how this study response those issues is mentioned. Chapter 3 contains previous literature related to this work in detail under literature review.

Chapter 4 provides an implementation overview of different stream cipher RC4 variants in Java Programming language integrated in NetBeans 8.0.2 version. The implementation details with major coding functions are provided in this chapter. Chapter 5 includes the analysis of time required for creating keystream and encrypting messages and finally with the help of average time needed for encrypting for candidates algorithms, cycle per byte is calculated. The result of the study is shown in tabular form as well as in graphs. Finally, the concluding remarks and further recommendations are outlined in chapter 6.

Chapter 2
BACKGROUND STUDY

Today, the internet has virtually become the way of doing business as it offers a powerful widespread medium of commerce and enables greater connectivity of disparate groups throughout the world. So, it may have many risks like loss of privacy, loss of data integrity, denial of service and identify spoofing. To the solution of these threads in internet many secure cryptographic algorithms are needed for providing services such as confidentiality, data integrity and authentication to handle packets which may vary in size over a large range. The size of the message has a significant impact on the performance of such algorithms. Hence the messages have to be prepared by padding the required amount of zero bits to get an integer number of blocks. This process becomes a considerable overhead when the short messages are more dominant in the message stream.

Since all the study require the basic terms and terminology related to that study. In this context, basic study related to this work is outlined in the following sections.

2.1 Cryptography

Cryptography is the art & science of making cryptosystem that is capable of providing information security. It is art of protecting information by encrypting it into an unreadable format, called cipher text. Only those who possess a secret key can decipher (or decrypt) the message into plaintext. Encrypted messages can sometimes be broken by cryptanalysis, also called code breaking, although modern cryptography techniques are virtually unbreakable. [5]
Cryptography enables one to store sensitive information or transmit it across insecure networks so that it cannot be read by anyone except the intended recipient. While cryptography is the science of securing digital data, cryptanalysis is the science of analyzing and breaking cipher text .It involves the study of cryptographic techniques to test their security strengths.
Until 1970's cryptography was considered the domain of military & government only [5]. However the worldwide use of computers & the rise of internet have made it an integral part of our daily lives .Today cryptography is at the heart of many secure applications such as

online banking, online shopping, online government services such as filling personal income taxes, cellular phones, & wireless LANs etc.

Cryptography is generally used in practice to provide four services: privacy, authentication, data integrity, & non-repudiation .The goal of privacy is to ensure that communication between two parties remain secret. This often means that the contents of the communication are secret; however in certain situation that very fact that communication took place must be a secret as well .Encryption is generally used to provide privacy in modern communication. Authentication of one or both parties during a communication is required to ensure that information is being exchanged with the authorized party. Passwords are common example of one-way authenticate them to gain access to a system.

Classical cryptanalysis involves an interesting combination of analytical reasoning, application of mathematical tools, pattern finding, patience, determination, and luck. Cryptanalysts are also called attackers and represented as Darth. Cryptology embraces both cryptography and cryptanalysis. The modern cryptography can be divided into two main branches [22]:

- Symmetric Cryptography, where the same key is used to encrypt a message and decrypt data.
- Asymmetric cryptography, where two different keys are used for encryption and decryption.

2.2 Cryptosystem

A cryptosystem [5] is an implementation of cryptographic techniques & their accompanying infrastructure to provide information security services. It is also known cipher system. The given figure is simple model of cryptosystem that provides the confidentiality to the information being transmitted. The basic model of cryptosystem is given below:

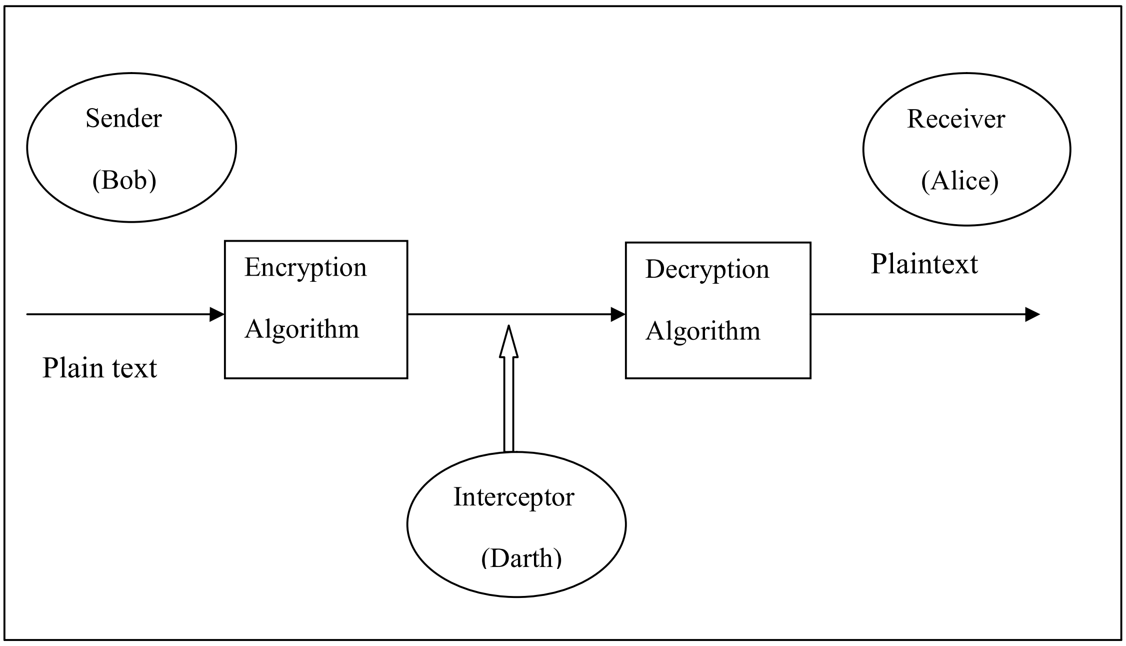


Figure-2.1: Cryptosystem [5]

The above figure shows a sender (Bob) who wants to transfer some sensitive data to receive (Alice) in such a way that any party intercepting or eves dropping on the communication channel cannot extract the data. An interceptor (or attacker) is an unauthorized entity who attempts to determine the plaintext. The objective of this simple cryptosystem is that at the end of the process, only the sender & receiver will know the plaintext.

2.2.1 Component of Cryptosystem

- **Plaintext**
  It is the data to be protected during transmission.
- **Encryption Algorithm:**
  It is cryptographic algorithm that takes plain text & encryption key as input and produces cipher text.
- **Cipher text:**
  It is the scrambled version of the plaintext produced by encryption algorithm using a specific the encryption key. It flows in public channel, which is not guarded. It can be intercepted compromised by anyone who has access to the communication channel.

- **Decryption Algorithm:**
  It is a cryptographic algorithm that takes cipher text & a decryption key as input & output a plaintext. The decryption algorithm essentially reverses of the encryption algorithm.
- **Encryption Key**:
  It is a value that is known to the sender. The sender inputs encryption key into the encryption algorithm along with the plaintext in order to compute the cipher text.
- **Decryption Key:**
  It is a value that is known to be receiver. The decryption key is related to the encryption key, but is not always identical to it.

2.3 Type of cryptosystem

There are two types of cryptosystems [5] based on the manner in which encryption-decryption is carried out in the system.

- ➢ Symmetric Key Encryption
- ➢ Asymmetric Key Encryption

2.3.1 Symmetric Key Encryption

The encryption process where same keys are used for, encrypting & decrypting the information is known as symmetric key encryption. The study of symmetric cryptosystem is referred to as symmetric cryptography or secret key cryptosystem or private key cryptosystem.

In symmetric key encryption a secret key is shared between the sender & the receiver. The word “symmetric” refers to the fact that both sender & receiver use same key to encrypt & decrypt the information. The secret key must be shared over a secure channel before the communication. Symmetric cryptosystem was the only type of encryption technique in use prior to the development of public key cryptosystem. Which can be defined as: Let M denotes the set of all possible plaintext messages, C the set of all possible cipher text, K the set of all possible keys, $k: M \rightarrow C$ is the encryption function, and $k: C \rightarrow M$, is decryption function, such that $k(k(m)) = m$ for all $m \varepsilon M$ and $k \varepsilon K$. In this cryptosystem, sender and receiver have to initially agree upon a secret key $k \varepsilon K$. After that, whenever sender wishes to send a message m ε M to receiver, sender sends the cipher text C = k (m) to receiver, from which receiver can recover m by applying the decryption function as m = k(C) [7]. The notion of private key cryptosystem is depicted in Figure 2.1.

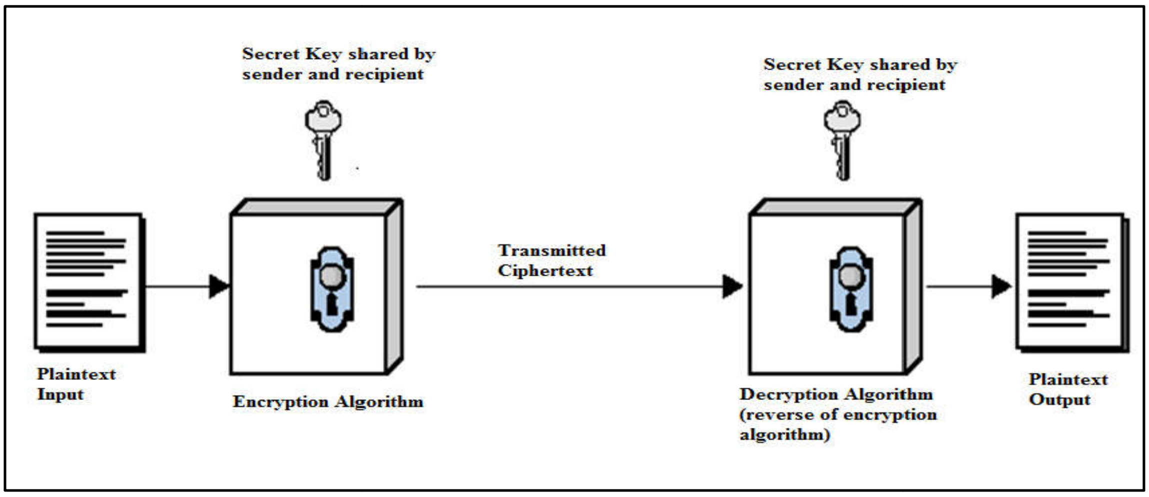

Secret Key shared by sender and recipient
Secret Key shared by sender and recipient
Transmitted Ciphertext
Plaintext Input
Encryption Algorithm
Decryption Algorithm (reverse of encryption algorithm)
Plaintext Output

Figure-2.2: Simplified Model of Symmetric Encryption [22].

The effectiveness of private key cryptosystems relies on the requirement of strong encryption algorithm which would be like the algorithm to be such that an opponent who knows the algorithm and has access to one or more cipher texts would be unable to decipher the cipher text or find out the key and another requirement is that sender and receiver must have obtained copies of the secret key in a secure fashion and must keep the key secure. Modern techniques used in private key cryptosystem are XOR Cipher, Rotation Cipher, Substitution Cipher: S-box, Transposition Cipher: Data Encryption Standard (DES), Advanced Encryption Standard (AES) and so on. The silent features of cryptosystem based on the symmetric key encryption are:

- Persons using symmetric key encryption must share a common key encryption must share a common key prior to exchange of information.
- Keys are recommended to be changed regularly to prevent any attack on the system.
- Length of key (number of bits) in this encryption is smaller & hence process of encryption- decryption is faster than asymmetric key encryption.
- Require less processing power of computer system to run symmetric algorithm.
- In a group of n people, to enable two-party communication between any two persons, the number of keys required for group is n*(n-1)/2.

As mentioned in [22], private key cryptosystems have numerous limitations which are outlined below:

- **Key distribution problem:** Two parties that want to communicate each other need to set up a shared secrete key before starting communicate over an insecure channel.
- **Key management problem:** Every pair of users must share a secret key leading to a total of n*(n-1)/2 keys. Where n be the users in a network. If n is large, then the number of keys become unmanageable and traffic in network may be increased. It makes difficult to manage key.
- **No signatures possible:** A digital signature is an authentication mechanism that enables the creator of the message to attach a special token that acts as a signature. A digital signature allows the receiver of a message to convince any third-party that the message in fact originated from the sender.

2.3.2 Asymmetric key Encryption

The encryption process where different keys are used for encryption and decryption the information is known as asymmetric key encryption. Encryption and decryption are performed using the different key- one a public key and one a private key. It is also known as public-key encryption. Asymmetric encryption transforms plaintext into cipher text using a one of two keys and an encryption algorithm. Using the paired key and a decryption algorithm, the plaintext is recovered from the cipher text.
Asymmetric encryption can be used for confidentiality, authentication, or both. The most widely used public-key cryptosystem is RSA. Public-key algorithms are based on mathematical functions rather than on substitution and permutation. More important, public-key cryptography is asymmetric, involving the use of two separate keys, in contrast to symmetric encryption, which uses only one key. The use of two keys has profound consequences in the areas of confidentiality, key distribution, and authentication.

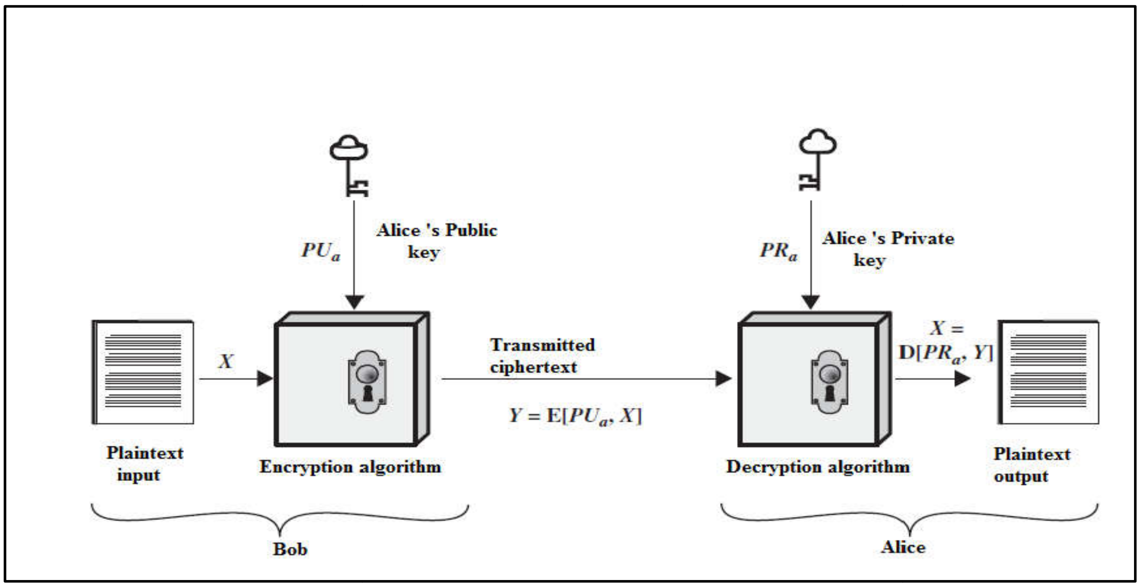


Figure-2.3: Encryption with Public Key [22].

Chapter 3
LITERATURE REVIEW

3.1 Modern Symmetric Key Encryption

Digital data is represented in strings of binary digits (bits) unlike alphabets. Modern cryptosystems need to process these binary strings to convert in to another binary string. Based on how these binary strings are processed, a symmetric encryption schemes can be classified into:

- Stream Cipher
- Block Cipher

3.1.1 Stream Ciphers

Stream ciphers are an important class of encryption algorithms and are defined as the ciphers in which plaintexts are encrypted by XORing between secret key and plaintexts to obtain ciphers. They encrypt individual characters (usually binary digits) of a plaintext message one at a time. Stream ciphers are generally faster than block ciphers in software as well as hardware applications [4]. They are more appropriate in some telecommunications applications, where buffering is limited or characters must be individually processed as they are received. A prominent example for a stream cipher is the A5/1 cipher, which is part of the GSM mobile phone standard and is used for voice encryption. However, stream ciphers are sometimes also used for encrypting Internet traffic, especially the stream cipher RC4.

Stream ciphers encrypt individual digits of plaintext using a time-varying transformation. Fig .3.1 shows the general structure of a stream cipher. The size of the digit can vary depending on the design constrains, application & underlying platform. A stream cipher usually consists of pseudo-random sequence of digits known as running keystream .The keystream digits are X-ORed with the plaintext digits to obtain ciphertext digits.

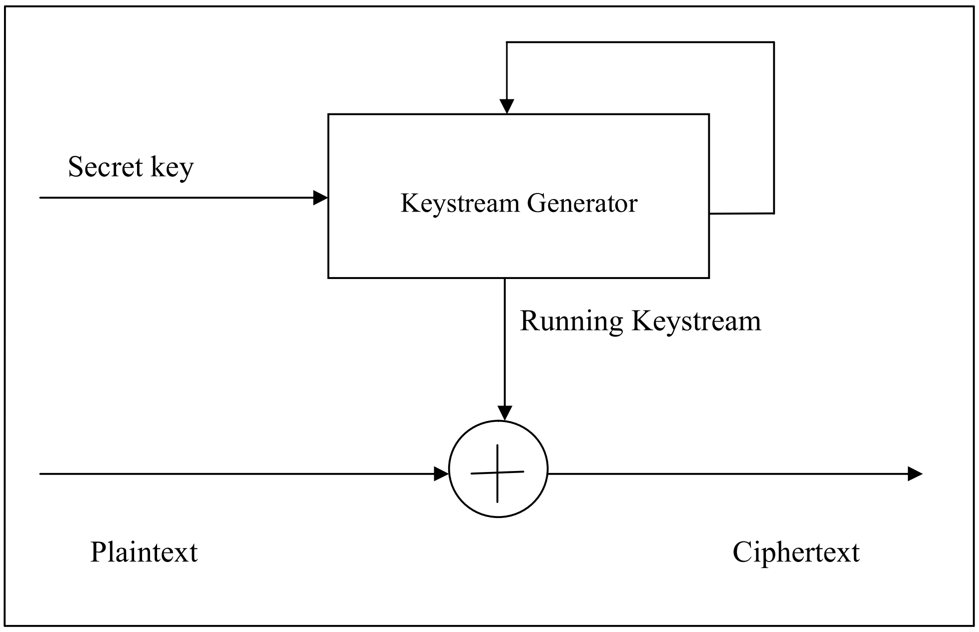


Figure-3.1. General Structure of a Stream Cipher [23]

If it is look at the types of cryptographic algorithms that exist in a little bit more detail, it can be seen that the symmetric ciphers can be divided into stream ciphers and block ciphers, as shown in Fig-3.2.

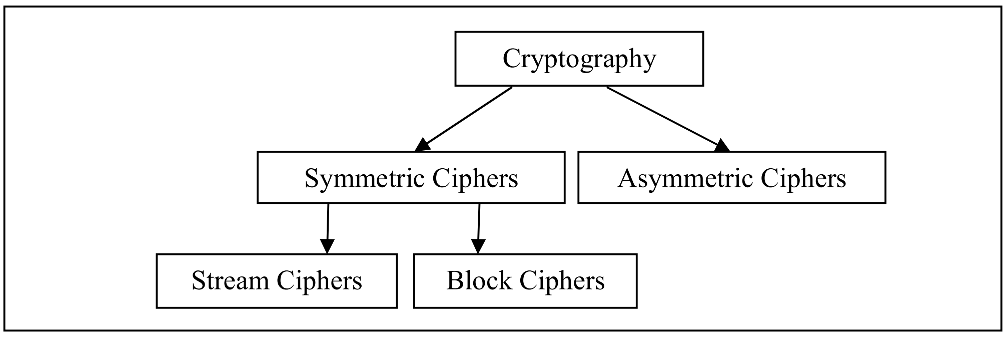


Figure-3.2: Showing Cryptographic Branches [4].

3.1.2 Stream and Block Ciphers

Symmetric cryptography is split into block ciphers and stream ciphers, which are easy to distinguish. Figure 3.3 depicts the operational differences between stream [Fig. 3.3(a)] and block [Fig. 3.3(b)] ciphers when we want to encrypt *b* bits at a time, where *b* is the width of the block cipher.

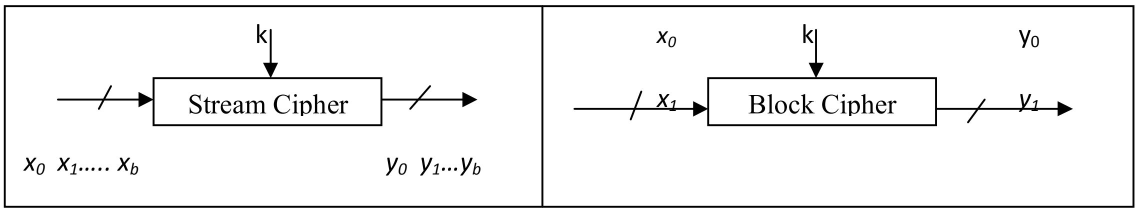


Figure-3.3: Principles of Encrypting *b* bits with a stream (a) and a block (b) cipher [4].

Stream ciphers encrypt bits individually. This is achieved by adding a bit from a keystream to a plaintext bit. There are synchronous stream ciphers where the key stream depends only on the key, and asynchronous ones where the key stream also depends on the cipher text.

If the dotted line in Fig. 3.4 is present, the stream cipher is an asynchronous one. Most practical stream ciphers are synchronous ones. An example of an asynchronous stream cipher is the cipher feedback (CFB) mode [4].

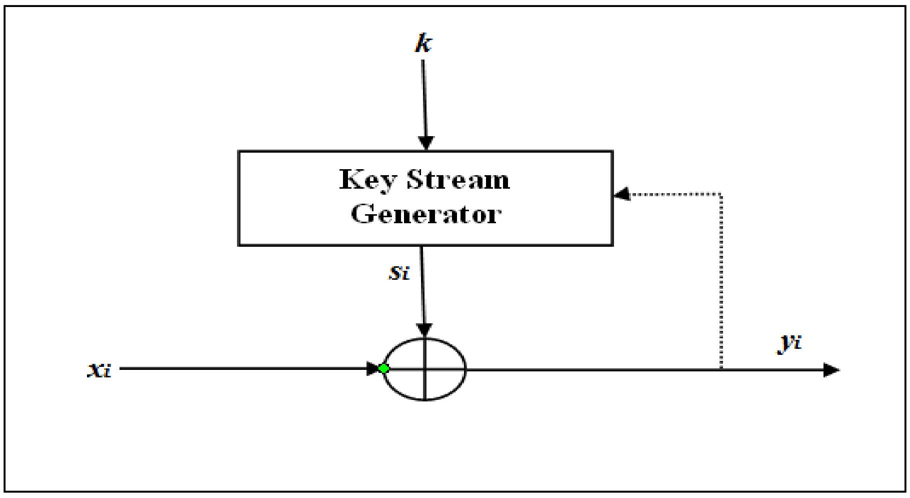


Figure-3.4: Showing Asynchronous Stream Cipher Generation [4].

Block ciphers encrypt an entire block of plaintext bits at a time with the same key. This means that the encryption of any plaintext bit in a given block depends on every other plain text bit in the same block. In practice, the vast majority of block cipher, either have a block length of 128 bits (16 bytes) such as the advanced encryption standard (AES), or a block length of 64 bits (8 bytes) such as the data encryption standard (DES) or triple DES (3DES) algorithm.

A block cipher is the symmetric key cryptography primitives which take as input n –bit block of cipher text using a fixed transformation. Fig 3.5 shows the general structure of block cipher. The common block sizes are 64-bit, 128 bit & 256 bits. For a fixed key the block cipher defines a permutation on the n-bit input. The building blocks of modern blocks are substitutions & permutations. A substitution is generally performed to provide non linearity in the output & permutation are used to diffuse the input and the key. These substitution & permutation operations in cipher are repeated several times to obtain highly non linear transformation of output bits.

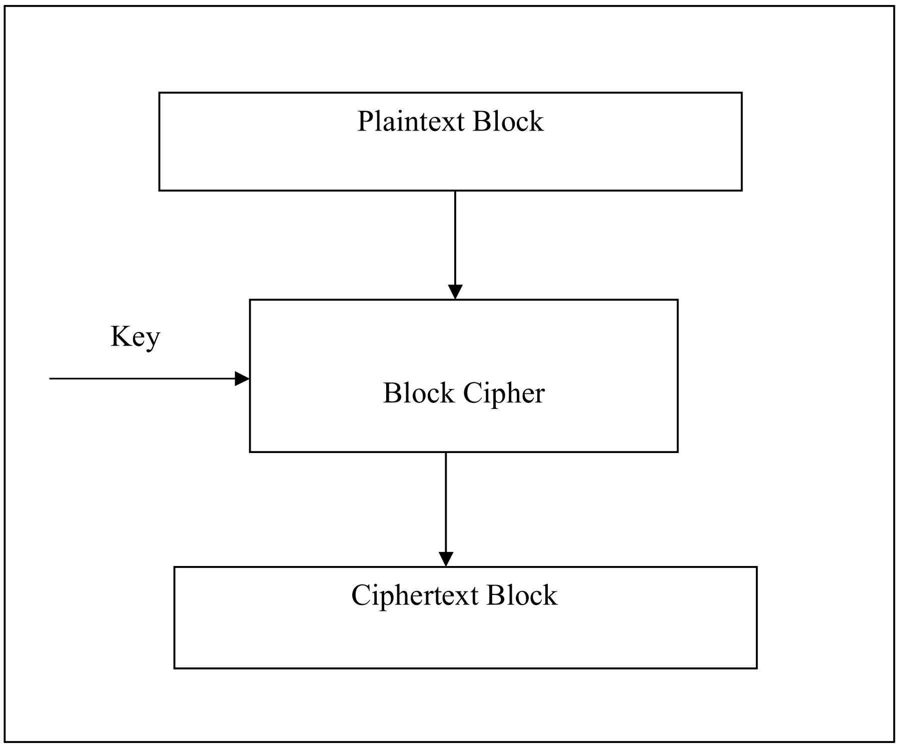


Fig-3.5: General Structure of Block Cipher [23]

The most famous block ciphers DES (Data Encryption Standard) & AES (Advanced Encryption Standard). DES was designed in 1970 & has been extensively used in the past three decades. It has a block size of 64 bits & secret key of 56 bits. AES was standardized in 2002 to replace DES since 56 bit key is too small for the computing power of today computer.AES has an increased key length of 128,192,256 bits and block size of 128 bits.

Some differentiation points between stream ciphers vs. block ciphers are as below:

1. In practice, particularly for encrypting computer communication on the Internet, block ciphers are used more often than stream ciphers.

2. Because stream ciphers tend to be small and fast, they are particularly relevant for applications with little computational resources, e.g., for cell phones or other small embedded devices.
3. It is assumed that stream ciphers tended to encrypt more efficiently than block ciphers. Efficient for software-optimized stream ciphers means that they need fewer processor instructions (or processor cycles) to encrypt one bit of plaintext.

3.1.3 Encryption and Decryption in Stream Ciphers

Example: Alice wants to encrypt the letter A, where the letter is given in ASCII code. The ASCII value for B is $66_{10} = 1000010_2$. Let's furthermore assume that the first key stream bits are ($s0, \ldots, s6$) = 0101100.

Sender: - Alice, Receiver: - Bob

$x0, \ldots, x6 = 1000010 = B$

$\oplus$

$s0, \ldots, s6 = 0101100$

$y0, \ldots, y6 = 1101110 = m$

.

.

.

. m= 1101110 $\rightarrow$

$y0, \ldots, y6 = 1101101$

$\oplus$

$s0, \ldots, s6 = 0101100$

$x0, \ldots, x6 = 1000010 = B$

According to above mentioned example, bitwise XORing between message and key is carried out for encryption as well as decryption.

3.1.4 Random Numbers, Nonce and OTP in Stream Cipher

Random numbers and its generation play very important role in stream ciphers since generating key is major thing and encryption and decryption is simply the XOR operation. Random number can be generated by three ways: TRNG, PRNG and CSPRNG [4, 20].

True random number generators (TRNGs) are characterized by the fact that their output cannot be reproduced. For instance, if we flip a coin 100 times and record the resulting sequence of 100 bits, it will be virtually impossible for anyone on Earth to generate the same 100 bit sequence. The chance of success is $1/2^{100}$, which is an extremely small probability. TRNGs are based on physical processes. Examples include coin flipping, rolling of dice etc [4].

Pseudorandom number generators (PRNGs) generate sequences which are computed from an initial seed value. Often they are computed recursively in the following way:

$$s0 = \text{seed}$$

$$s_{i+1} = f(s_i),\ i = 0,1,\ldots$$

A widely used example is the *rand()* function used in ANSI C. A common requirement of PRNGs is that they possess good statistical properties, meaning their output approximates a sequence of true random numbers. There are many mathematical tests, e.g., the chi-square test for PRNG [4].

Cryptographically secure pseudorandom number generators (CSPRNGs) are a special type of PRNG which possess the following additional property: A CSPRNG is PRNG which is unpredictable. Informally, this means that given n output bits of the key stream s_i, s_{i+1}, . . . , $s_{i+n\ 1}$, where n is some integer, it is computationally infeasible to compute the subsequent bits s_{i+n}, s_{i+n+1}, ... A more exact definition is that given n consecutive bits of the key stream, there is no polynomial time algorithm that can predict the next bit s_{n+1} with better than 50% chance of success.

Another property of CSPRNG is that given the above sequence, it should be computationally infeasible to compute any preceding bits $s_{i\ 1}$, $s_{i\ 2}$, ... Note that the need for unpredictability of CSPRNGs is unique to cryptography. Virtually, in all other situations where pseudorandom numbers are needed in computer science or engineering, unpredictability is not needed [20, 2].

Many aspects of cryptography require random numbers, for example: Key generation, nonce, one-time pads etc. The "quality" of the randomness required for these applications varies. For example, creating a nonce in some protocols needs only uniqueness. On the other hand, generation of a master key requires a higher quality. In cryptography, a nonce is an arbitrary number that may only be used once. It is similar in spirit to a nonce word. They can also be useful as initialization vectors and in cryptographic hash function [2].

In cryptography, the one-time pad (OTP) is an encryption technique that cannot be cracked if used correctly [13]. In this technique, a plaintext is paired with a random secret key (also referred to as *a one-time pad*). Then, each bit or character of the plaintext is encrypted by combining it with the corresponding bit or character from the pad using modular addition. If the key is truly random, is at least as long as the plaintext, is never reused in whole or in part, and is kept completely secret, then the resulting cipher text will be impossible to decrypt or break. It has also been proved that any cipher with the perfect secrecy property must use keys with effectively the same requirements as OTP keys [13].

3.2 Candidate Algorithms

3.2.1 VMPC

VMPC (Variably Modified Permutation Composition) is a stream cipher variant of RC4.It was designed by Bartosz Zoltak, presented in 2004 at Fast Software Encryption conference [3].The core of the cipher is VMPC one way function [3], which is a combination of triple permutation composition & integer addition. The cipher generate a stream of 8-bit value from an internal state consists of 256-byte permutation two 8-bit integer variable. The generated values should be X- ORed with the plain text to derive the cipher test. The initial value of cipher's internal state is determined by the VMPC Key Scheduling Algorithm.

Description of the VMPC Stream Cipher

The algorithm generates a stream of 8-bit values.

Variables:

P : 256-byte table storing a permutation initialized by the VMPC KSA

s : 8-bit variable initialized by the VMPC KSA

n : 8-bit variable

L : desired length of the keystream in bytes

The Core Algorithm of VMPC Stream Cipher [3]

```
1. n = 0
2. Repeat steps 3-6 L times:
3. s = P[(s + P[n]) modulo 256 ]
4. Output P [(P [P[s]] + 1) modulo 256]
5. Swap(P[n] , P[s])
   Temp = P[n]
   P[n] = P[s]
   P[s] = Temp
6. n = (n + 1) modulo 256
```

Description of the VMPC Key Scheduling Algorithm

The VMPC Key Scheduling Algorithm (KSA) transforms a cryptographic key and (optionally) an Initialization Vector into a 256-element permutation P and initializes variable s.
Variables as, with:

c : fixed length of the cryptographic key in bytes, 16 <= c <= 64
K : c-element table storing the cryptographic key
z : fixed length of the Initialization Vector in bytes, 16 <= z <= 64
V : z-element table storing the Initialization Vector
m : 16-bit variable

VMPC Key Scheduling Algorithm [3]

```
1. s = 0
2. for n from 0 to 255: P[n] = n
3. for m from 0 to 767: execute steps 4-6:
4. n = m modulo 256
5. s = P[(s + P[n] + K[m modulo c]) modulo 256 ]
6. Temp = P[n]
P[n] = P[s]
P[s] = Temp
7. If Initialization Vector is used: execute step 8:
8. for m from 0 to 767: execute steps 9-11:
9. n = m modulo 256
10. s = P[(s + P[n] + V [m modulo z]) modulo 256 ]
11. Temp = P[n]
P[n] = P[s]
P[s] = Temp
```

3.2.2 SPRITZ

SPRITZ is a stream cipher variant of RC4 proposed by Rivest and Schuldt at the rump session of CRYPTO 2014[16]. It is intended to be a replacement of the popular RC4 stream cipher.

The Core Algorithm of SPRITZ Stream Cipher [14]

```
1:   i = i + w
2:   j = k + S[j + S[i]]
2 a):  k = i + k + S[j]
3:  SWAP(S[i],S[j])
4: z = S[j + S[i + S[z + k]]]
5: Return z
```

S is an 8-bit permutation. In theory, it can be any size, which is nice for analysis, but in practice, it's a 256-element array. RC4 has two pointers into the array: i and j. SPRITZ adds a third: k. The parameter w is basically a constant. It's always 1 in RC4, but can be any odd number in SPRITZ (odd because that means it's always relatively prime to 256). It has a single swap of two elements of the array & produces an output byte, z, a function of all the other parameters. In SPRITZ, the previous z is part of the calculation of the current z. There are also functions for turning the key into the initial array permutation, using this as a stream cipher, using it as a hash function, and so on. It's basically a sponge function, so it has a lot of applications.

Description of the SPRITZ Stream Cipher

SPRITZ consists of a permutation S over the set {0, 1, 2,…, N – 1} (default value of N is 256) and six pointers i, j, k, w, a, z, where i, j, k are index pointers, w gives the step distance for i, a is a nibble counter, and z stores the output byte. The design specifies a number of modules that are executed for producing a key stream as defined in pseudocode given below. There are number of modes of operation using the SPRITZ structure like a stream cipher, hash function, MAC etc. In the stream cipher mode of operation the key stream is produced in the following manner. First the permutation is initialized using the INITIALIZESTATE (N) routine. The secret key K is then absorbed into the state using the ABSORB (K) module. Additionally, if an IV is to be used, then the ABSORBSTOP ( ) module is invoked and the IV is absorbed by calling the ABSORB (IV) function. Thereafter, the SQUEEZE module is invoked to produce keystream bytes.

```
INITIALIZESTATE (N)
1    i = j = k = z = a = 0
2    w = 1
3    for v = 0 to N - 1
4        S[v] = v
```

```
ABSORB (I)
1    for v = 0 to I. length - 1
2        ABSORBBYTE (I [v])
```

```
ABSORBBYTE (b)
1   ABSORBNIBBLE (LOW (b))
2   ABSORBNIBBLE (HIGH (b))

ABSORBNIBBLE(x)
1  if a = [N/2]
2  SHUFFLE ( )
3  SWAP(S[a], S [[N/2] + x])
4   a = a + 1

ABSORBSTOP ( )
1    if   a = [N/2]
2     SHUFFLE ( )
3     a = a + 1
SHUFFLE ( )

1    WHIP (2N)
2     CRUSH ( )
3      WHIP (2N)
4      CRUSH ( )
5     WHIP (2N)
6      a = 0

WHIP(r)
1    for v = 0 to r - 1
2     UPDATE ( )
3    do   w = w + 1
4    until GCD (w, N) = 1

CRUSH ( )
1  for v = 0 to [N/2] -1
2  if S[v] > S [N - 1 - v]
3   SWAP(S[v]  , S [N -1 -v])
SQUEEZE(r)
```

```
1    if  a > 0
2        SHUFFLE ( )
3    P = ARRAY.NEW(r)
4    for v = 0 to r - 1
5        P[v] = DRIP ( )
6    return P

DRIP ( )
1  if a > 0
2    SHUFFLE ( )
3  UPDATE ( )
4  return OUTPUT ( )

UPDATE ( )
1  i = i + w
2  j = k + S [j + S[i]]
3  k = i + k + S[j]
4  SWAP(S[i], S[j])

OUTPUT ( )
1  z = S [j + S [i + S [z + k]]]
2  return z
```

Pseudocode for Spritz.[14]

The main interface routines are INITIALIZESTATE, ABSORB (and ABSORB-STOP), and SQUEEZE. The state consists of byte registers i, j, k, z, w, and a, and an array S containing a permutation of {0, 1,….., N – 1}.

3.2.2.2 Sponge functions

One objective of RC4 redesign is to reformulate RC4 as a "sponge function." RC4 is a natural fit for adaptation to the sponge function paradigm, as it already has a large state space. As proposed by Bertoni [6], a sponge function has a function ABSORB that absorbs variable-length input into the state, and a function SQUEEZE that produces variable-length output from the state. Both functions may change the state, using a single state-space permutation function f.

In Spritz, the SHUFFLE procedure corresponds to the state-space permutation of the sponge function. However, SHUFFLE is not a state-space permutation, but a many-to-one map (due to its invocations of CRUSH), so the correspondence is not precise. SHUF-FLE does invoke procedure WHIP, however, which is such a state-space permutation. The sponge-like character of SPRITZ, based on SHUFFLE, WHIP & CRUSH procedure. SPRITZ in the sponge function framework, additional evaluation is definitely needed regarding the use of SPRITZ in these additional sponge function modes.

3.2.2.2.1 AbsorbStop function

The sponge function interface extended by introducing the function ABSORBSTOP ( ) to provide a simple clean way to separate different inputs being absorbed.

3.2.2.2.2 Encryption

Here is pseudocode illustrating the use of a sponge function for encryption and decryption.[14]

```
ENCRYPT (K, M)
1 KEYSETUP (K)
2 C = M + SQUEEZE (M. length)
3 RETURN C

DECRYPT (K, C)
1 KEYSETUP (K)
2 M = C – SQUEEZE (M. length)
3 RETURN M

KEYSETUP (K)
1 INITIALIZESTATE ( )
2 ABSORB (K)
```

ENCRYPT uses key-setup algorithm KEYSETUP to initialize the state and absorb the key K, then adds modulo N each byte of the message M with the corresponding byte of the output of

SQUEEZE, yielding cipher text C. Procedure DECRYPT is identical, except for switching M and C, and replacing addition modulo N with subtraction modulo N. Note that the key, message, and cipher text are all byte sequences (sequences of values modulo N).
Addition/subtraction modulo N are used instead of the more traditional exclusive-or, for consistency with the design goal that SPRITZ should work for all N, not just N that are powers of 2. Of course, when N is a power of 2, one could use exclusive-or rather than addition/subtraction.
To encrypt with an IV (initialization vector or nonce) IV, one should, after the key is input, call ABSORBSTOP procedure to separate the two fields, and then input the IV, as follows.

```
ENCRYPTWITHIV (K, IV, M)
1 KEYSETU P (K); ABSORBSTOP ( )
2 ABSORB (IV)
3 C = M + SQUEEZE (M. length)
4 return C
```

3.2.2.2.3 Hash function

The following procedure Hash produces an r-byte hash of the input message (byte sequence) M.

```
Hash (M, r)
1 INITIALIZESTATE ( )
2 ABSORB (M); ABSORBSTOP ( )
3 ABSORB(r)
4 return SQUEEZE(r)
```

HASH first absorbs the input message M (a byte sequence). Next, it call ABSORBSTOP to signal the end of the message M, and the beginning of the next input, which is the desired hash length, r.

Chapter 4

IMPLEMENTATION & TESTING

4.1 Java Implementation

Java was conceived at Sun Microsystems, in 1991. This language is initially called "OAK" but it was renamed as java in 1995 with the Virtual Machine being known as the Java Virtual Machine (JVM). At that time, the use of the World Wide Web was starting to become widespread. The web involved the communication between all sorts of processors and systems; just the sort of situation for which Sun Micro system had developed Java. Hence Java became the preferred language for Web programming [17].
Java compiles the source file (.java) and converts into intermediate file called byte code (.class) . This beauty of the java programming language motivates to use of java anywhere or in any type of application development. This makes software developed in java platform independent.

4.2 Choice of the Programming Language: Java

Most of other language likes C, C++ are designed to be compiled for a specific target machine. Although it is possible to compile a C++ program for any type of CPU, to do so requires a full C++ compiler targeted for that CPU. The problem is that compilers are expensive and time consuming to create, solution was needed, and to find a solution, java was created which could be used to produce code that can run on a variety of CPUs under different environment.
The Java security APIs spans a wide range of areas. For developing secure application, Cryptographic and public key infrastructure (PKI) interfaces are used. Interfaces for performing authentication and access control enable applications to protect against unauthorized access to protected resources. The APIs allow for multiple interoperable implementations of algorithms and other security services. Services are implemented in providers, which are plugged into the Java platform via a standard interface that makes it easy for applications to obtain security services without having to know anything about their implementations. This allows developers to focus on how to integrate security into their applications, rather than on how to actually implement complex security mechanisms. The

Java platform includes a number of providers that implement a core set of security services. It also allows for additional custom providers to be installed. This enables developers to extend the platform with new security mechanisms.

4.3 Netbeans

NetBeans is an integrated development environment (IDE) for developing primarily with Java, but also with other languages, in particular PHP, C/C++, and HTML5. It is developed at Charles University as a student project in 1996. In 1997, it was produced as commercial versions and bought by Sun Microsystems in 1999.

It is also an application platform framework for Java desktop applications and others. The NetBeans IDE is written in Java and can run on Windows, OS X, Linux, Solaris and other platforms supporting a compatible JVM. The NetBeans Platform allows applications to be developed from a set of modular software components called modules.

Different versions of Netbeans IDE are introduced in last few years. NetBeans IDE 7.0 was released in April 2011. On August 1, 2011, the NetBeans Team released NetBeans IDE 7.0.1, which has full support for the official release of the Java SE 7 platform. As passing versions from NetBeans IDE 6.5 to currently developing version NetBeans IDE 8.0 many more features are added in newer versions. NetBeans IDE 7.4 was released in October 15, 2013. NetBeans IDE 8.0 is currently in development. NetBeans IDE 8.0.2 is used for implementing in this thesis [15].

4.4 Implementation Details of Candidate Algorithms

Two algorithms / techniques are made to run by feeding same size of message at once. Different classes are created for each algorithm and related coding, functions are kept in class. Finally, all functions and classes are accessed from the main class. Time taken to encrypt the message file is calculated individually. The main thing in stream cipher is to design and develop key stream which then will be XORed with the message. After generating key stream, 128 bit size key gets encrypted by XORing with same size of message. Important Java coding and functions for each algorithm are given as below.

4.4.1 VMPC

Main Java coding/functions for VMPC algorithm are given as below. Initial Vector and initial key is directly given in code level. Message encrypting as well as functions converting into byte code are also given below. Details about coding are mentioned in Appendix section.

- ❖ public void init(boolean forEncryption, CipherParameters params)
- ❖ protected void initKey(byte[] keyBytes, byte[] ivBytes)
- ❖ public int processBytes(byte[] in, int inOff, int len, byte[] out, int outOff)
- ❖ public void reset()
- ❖ public byte returnByte(byte in)

4.4.2 SPRITZ

We can see important functions required in SPRITZ algorithm below. Important functions involving in this algorithm are Sponge function, ABSORBSTOP function, are given. They are as follows. Details about coding are mentioned in Appendix section.

- ❖ private static final int IV_SIZE=10
- ❖ public spritz ()
- ❖ public static final in gcd(int x1,int x2)
- ❖ private void update()
- ❖ private void whip(int r)
- ❖ private void crush()
- ❖ private void shuffle()
- ❖ private void absorb_nibble(int x){
- ❖ private void absorbe_byte(byte b)
- ❖ private void absorb(byte[])
- ❖ private void absorb_stop()
- ❖ private byte drip()

4.5 Research Methodology

Research Study, here is to find out the technique which is of very high speed i.e. the algorithm which can encrypt the given message (plaintext) with very high speed than other.

For this, random size of data that is of any kinds will be collected to feed to individual algorithm and time taken to encrypt will be calculated.

4.5.1 Data Collection

Various sample messages of different sizes will be fed to the different modules. Messages may be either text or number or images/graphics etc. But for easy plaintext data of different sizes are taken to input for all algorithms. Secondary data collection method is used here, because no primary data is required in this study.

4.6 Sample Test Cases

For testing data input, the different size of text file is taken as input message. Size of 30 bytes file has been taken as smallest size and 10KB as big message. Constant keys as well as IV given in program, sample of input message and generated ciphers are as follows:

4.6.1 Key

I. VMPC

```
K =  "AAAAAAAAqweAAAAT"
```

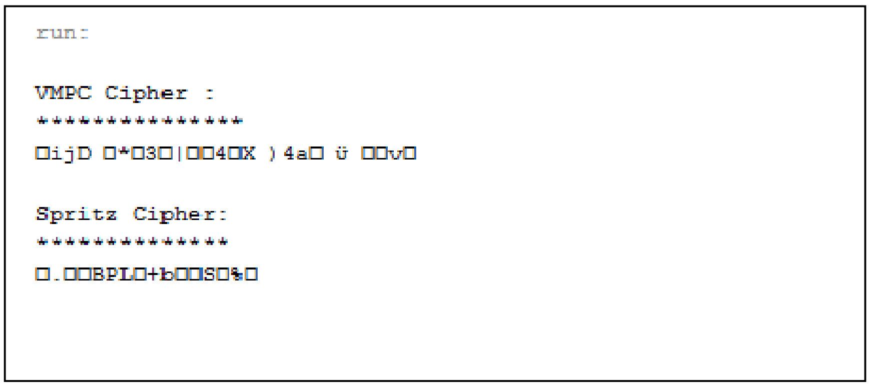
II. SPRITZ

```
Byte [] key = {
(byte)0xa0,   (byte)0x33,   (byte)0xd6,   (byte)0x78,   (byte)0x6b,   (byte)0x05,
(byte)0x14,  (byte)0xac,  (byte)0xfc, (byte)0x3d, (byte)0x8e, (byte)0x2d,  (byte)0x6a,
(byte)0x2c,   (byte)0x27,    (byte)0x1d
}
```

4.6.2 Input Message (30 Bytes)

```
"Tribhuvan University Santosh"
```

4.6.3 Cipher After Encryption

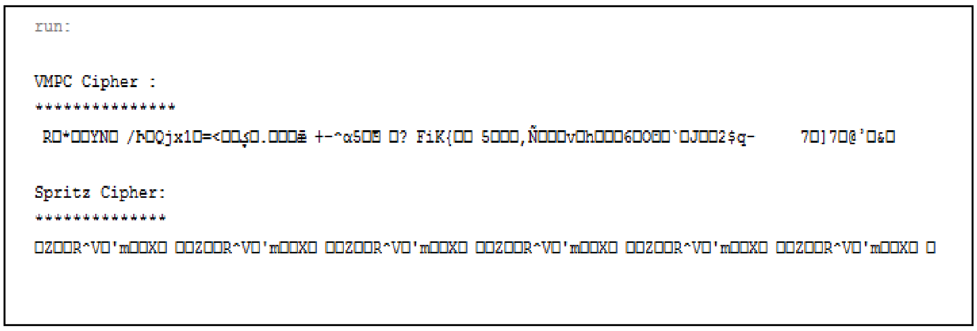


```
run:

VMPC Cipher  :
***************
□ijD □*□3□|□□4□X ) 4a□ ü □□v□

Spritz Cipher:
**************
□.□□BPL□+b□□S□%□
```

4.6.4 Input Message (100 Bytes)

" Performance Analysis of Stream Cipher RC4 Variants: VMPC and SPRITZ " by Santosh Sharma CDCSIT-TU.

4.6.5 Cipher After Encryption



```
run:

VMPC Cipher :
***************
 R□*□□YN□ /Ⱶ□Qjx1□=<□□ş□.□□□ǣ +-^α5□□ □? FiK{□□ 5□□□,Ñ□□□v□h□□□6□O□□`□J□□2$q-      7□]7□@'□&□

Spritz Cipher:
**************
□Z□□R^V□'m□□X□ □□Z□□R^V□'m□□X□ □□Z□□R^V□'m□□X□ □□Z□□R^V□'m□□X□ □□Z□□R^V□'m□□X□ □□Z□□R^V□'m□□X□ □
```

Chapter 5
RESULT & ANALYSIS

This chapter presents an overview of comparison stream cipher RC4 variants in terms of performance and cost. Target Architecture and Specifications are described in this chapter. Time in system nanosecond needed for encrypting all stream cipher RC4 variants algorithms implemented in java is calculated and performance is analyzed as cycle/byte.

5.1 Target Architectures

The main goal of this thesis is to measure the performance of stream cipher RC4 variants. These candidate algorithms are tested on laptop. The following system is used:

- A PC with an Intel Core i5 Processor 2.50 GHz having 4GB RAM .The operating system is Windows 8 running in 64-bit mode. The system is running the JDK (Java Development Kit) with NetBeans IDE 8.0.2.

5.2 Measuring Cost

There is some extra cost which may be added to the absolute cost for encrypting the different size of messages but this is equally affected to all candidates algorithm on the execution. The system time in nanosecond is taken just before the execution of code segment for generating key stream and encrypting the message in each algorithm and the completion of the execution. The time spending for encrypting message is calculated by subtracting start time taken before execution from completion time taken after completing execution of specific code segment [22]. The time required for each algorithm is calculated as follows:

```
long    startTime = System.nanoTime();
//  createKEY and encrypt function call
long    timeRequired  =  System.nanoTime() -  startTime;
```

Various processes may be run in background of system so absolute measurement may not be carried out. Due to this reason, time needed for encrypting given message in all algorithms may not be observe same in every run of program. Therefore at least 5 times, the program implemented in java is run in architecture described as above section and finally average required time observed in every run is calculated as:

Average required Time = $\sum_{i=1}^{5} \frac{T_i}{5}$ where T_i represent time obtained in i^{th} run of execution. This average calculated time is used to calculate cycle per byte.

5.3 Measuring Performance

Timing cryptographic primitive is useful when analyzing the performance of multiple algorithms on a single machine. But, it may vary on other machine therefore, cryptographers prefer to measure how many cycle it takes to process each byte. Different cycle/byte is calculated in the same box also because of background other process. So to optimize such extra cost, average is taken running multiple times in same machine for each candidate algorithms.

In this thesis, Cycle/byte calculation with the following parameters: time in second spent generating key and encrypting the message (T_s), frequency of the CPU in Hz(F) and message input in bytes(L). The formula for creating cycle/byte suggested by [22] is:

$$\text{Cycle/byte} = \frac{T_S\ F}{L}$$

5.4 Analysis

This section shows the result of the performance tests for various input sizes of each algorithm. A simple multiple histograms for each candidate algorithms will be presented each for cycle per byte calculation.

Following table and corresponding charts show the overall performance in the different encryption algorithms, VMPC & SPRITZ. Different sizes of data like 30 bytes, 100 bytes, 1KB, 5KB and 10KB are taken by every candidate algorithms as below.

Message Size = 30 Bytes

Candidate Algorithms	1st Run	2nd Run	3rd Run	4th Run	5th Run	Average
VMPC	1520297	10354	9561	8408	8490	311422
SPRITZ	1818724	121933	121932	107974	121111	458334.8

Table 5.1: Average Time in Nanosecond Spent by Candidate Algorithms to Generating Key and Encrypting the Message of Small Size (30 Bytes)

Candidate Algorithms	Cycle/Byte
VMPC	25951.83
SPRITZ	38194.56

Table 5.2: Performance of the Candidate Algorithms for Small Message Size (30 bytes) calculated in Cycle/Byte.

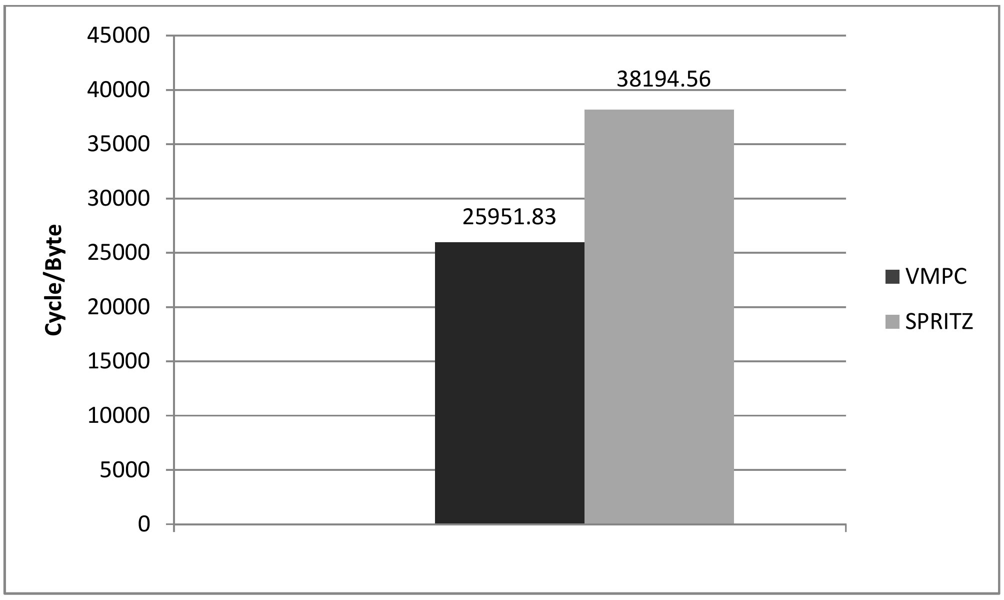


Figure 5.1: Performance of Candidate Algorithms for Small Message Size (30 Bytes) shown in Bar Diagram.

Message Size = 100 Bytes

Candidate Algorithms	1st Run	2nd Run	3rd Run	4th Run	5th Run	Average
VMPC	2156464	10670	9163	8945	9738	438996
SPRITZ	1713624	584619	540690	533711	949595	864447.8

Table 5.3: Average Time in Nanosecond Spent by Candidate Algorithms to Generating Key and Encrypting the Message of Size (100 Bytes)

Candidate Algorithms	Cycle/Byte
VMPC	1097.49
SPRITZ	21611.19

Table 5.4: Performance of Candidate Algorithms for Message Size (100 bytes) in Cycle/Byte.

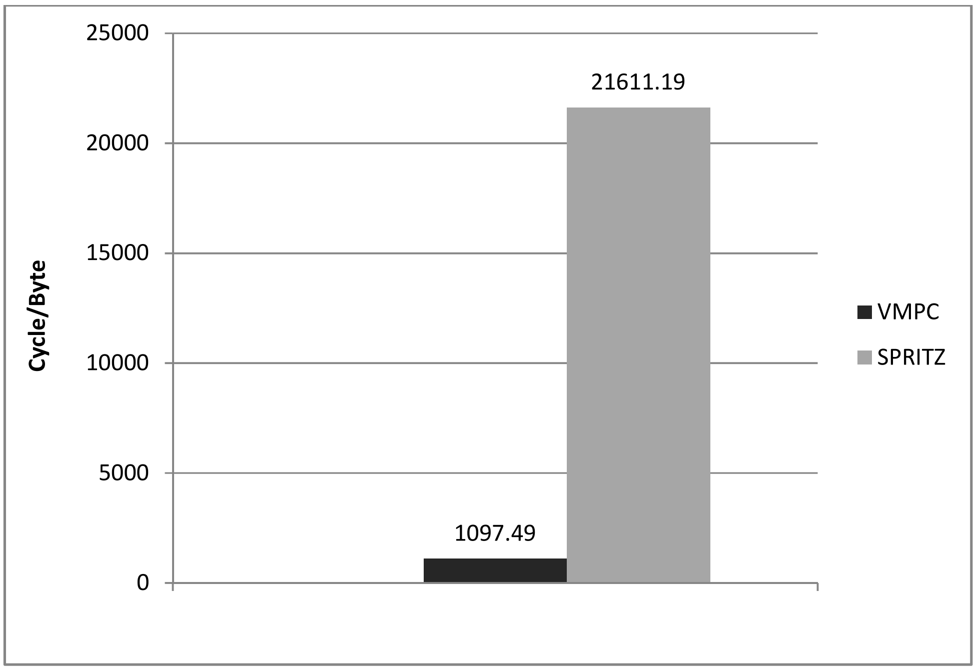


Figure 5.2: Performance of Candidate Algorithms for Message Size (100 bytes) shown in Bar Diagram

Message Size = 1KB

Candidate Algorithms	1st Run	2nd Run	3rd Run	4th Run	5th Run	Average
VMPC	1527805	14369	36748	18244	11220	321677.2
SPRITZ	4470858	1785470	6787986	1318677	1057158	3040298

Table 5.5: Average Time in Nanosecond Spent by Candidate Algorithms to Generating Key and Encrypting the Message of Size (1KB)

Candidate Algorithms	Cycle/Byte
VMPC	785.34
SPRITZ	7529.36

Table 5.6: Performance of Candidate Algorithms for Message Size (1KB) in Cycle/Byte.

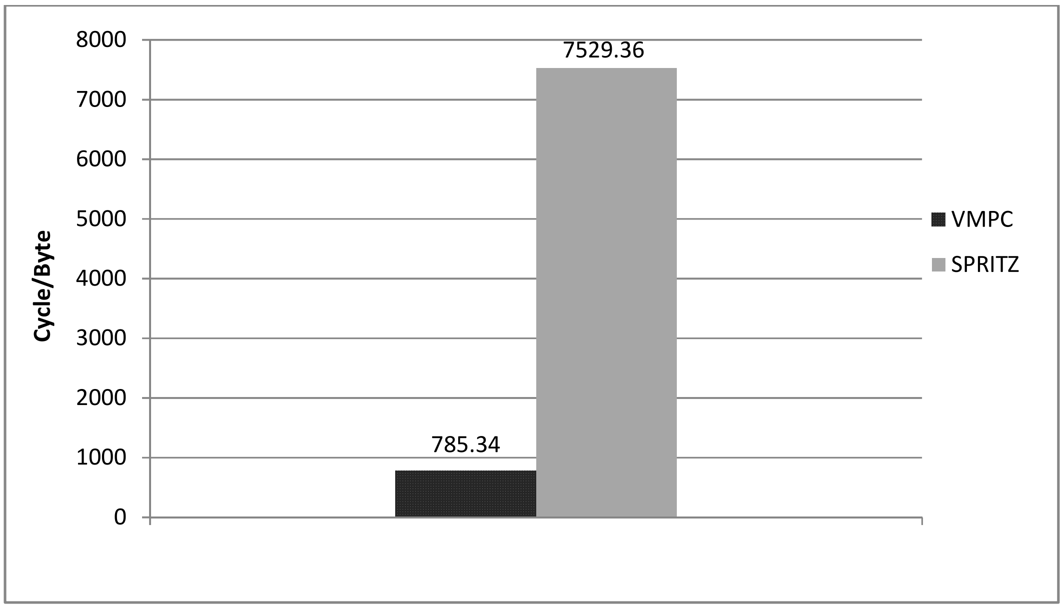


Figure 5.3: Performance of Candidate Algorithms for Message Size (1KB) shown in Bar Diagram.

Message Size = 5KB

Candidate Algorithms	1st Run	2nd Run	3rd Run	4th Run	5th Run	Average
VMPC	1624075	37068	343505	35943	13100	348907.2
SPRITZ	11532482	12955439	17697044	15230280	13091633	14101375.6

Table 5.7: Average Time in Nanosecond Spent by Candidate Algorithms to Generating Key and Encrypting the Message of Size (5KB)

.

Candidate Algorithms	Cycle/Byte
VMPC	170.36
SPRITZ	6885.43

Table 5.8: Performance of Candidate Algorithms for Message Size (5KB) in Cycle/Byte.

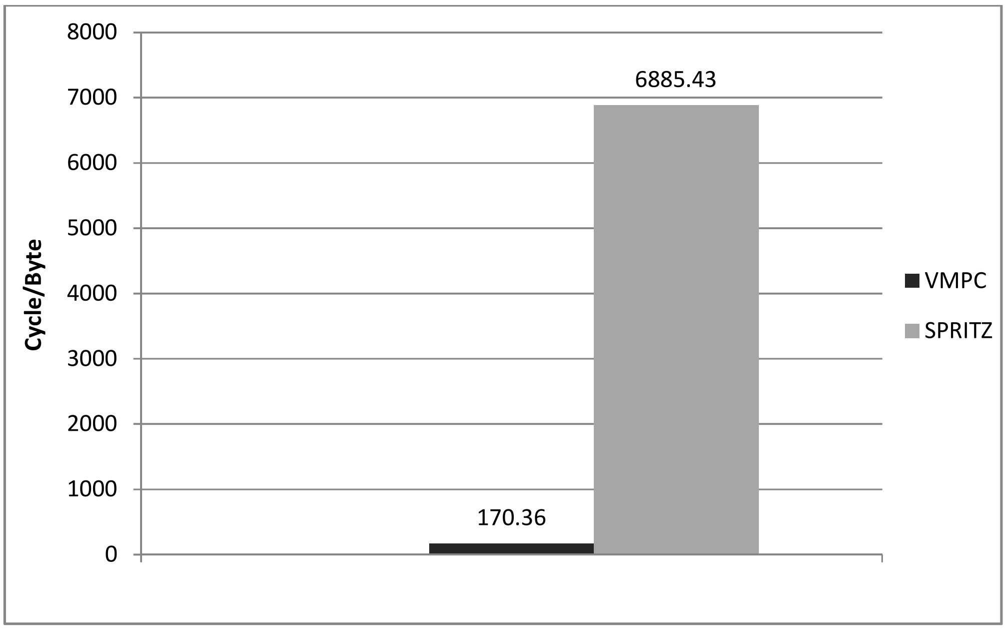


Figure 5.4: Performance of Candidate Algorithms for Message Size (5KB) shown in Bar Diagram.

Message Size = 10KB

Candidate Algorithms	1st Run	2nd Run	3rd Run	4th Run	5th Run	Average
VMPC	1404284	51244	29900	28348	15929	30594
SPRITZ	30177677	30793087	15594222	14507503	17645726	21743673

Table 5.9: Average Time in Nanosecond Spent by Candidate Algorithms to Generating Key and Encrypting the Message of Size (10KB)

Candidate Algorithms	Cycle/Byte
VMPC	74.69
SPRITZ	5308.50

Table 5.10: Performance of Candidate Algorithms for Message Size (10KB) in Cycle/Byte.

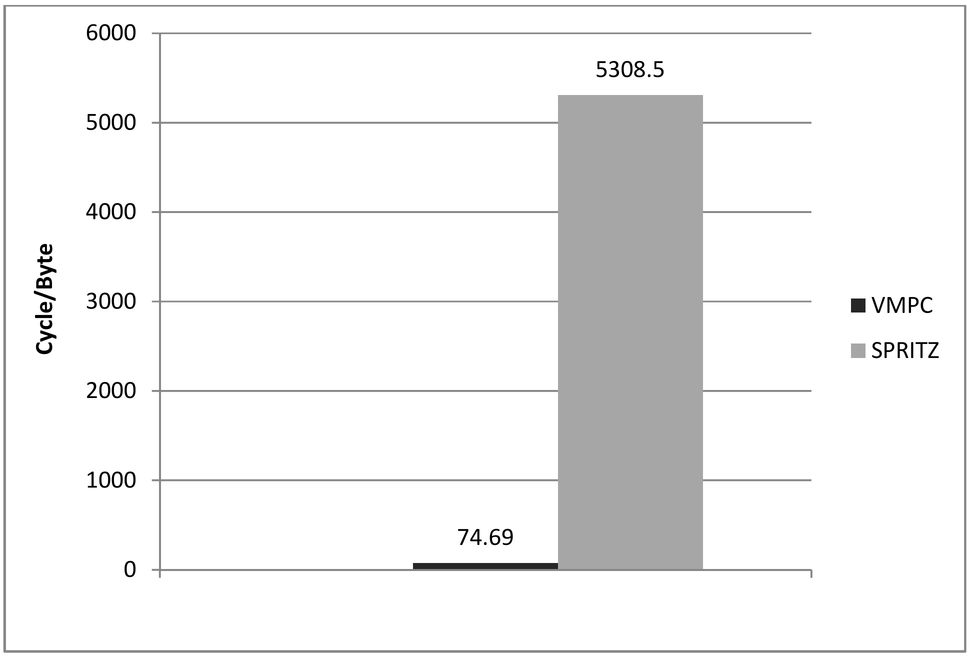


Figure 5.5: Performance of Candidate Algorithms for Message Size (10KB) shown in Bar Diagram

5.5 Result

By the help of above mentioned measuring criteria in the targeted architecture as given in section-5.1 to 5.4, thus obtained result is analyzed. Cycle/byte measuring unit is the best way to check the performance of any algorithms. After running the each algorithm with different sized message, it is observed that performance of VMPC is better than SPRITZ algorithms if the message size is very large.

With the increment of size of message, performance of VMPC is found to be better than SPRITZ. The speed of stream cipher SPRITZ is much slower, although performance of SPRITZ also increases with the increment of size of message. With the small size of message these algorithms VMPC & SPRITZ shows worst performance. In bigger size messages (5KB, 10KB) SPRITZ yields (6885.43 cycle/byte, 5308.50 cycle/byte) respectively & VMPC yields (170.36 cycle/byte, 74.69 cycle/byte) respectively. Therefore VMPC stream cipher algorithm is better algorithm purposed in this study.

Chapter 6
CONCLUSION & FUTURE WORK

6.1 Conclusions

In this thesis, stream cipher RC4 variants (algorithms) are studied, discussed and implemented using most popular and highly accurate programming language Java. After implementation, different size messages were feed to encrypt to the candidate algorithms and results were observed and analyzed. With the small size of message these stream cipher RC4 variants VMPC & SPRITZ shows low performance. The result of empirical performance comparison shows that VMPC is the better than SPRITZ .The speed of SPRITZ is much slower due to cryptographic function "sponge like construction", where different functions are used for the additional layer of security in RC4 variants, although performance of SPRITZ also increases with increment of size of message. While message size gets increased, performance of VMPC is better in all the size of messages. Hence, stream cipher VMPC shows the better performance in large size message than stream cipher SPRITZ.

6.2 Future Works

Actually, considering and keeping in mind that securities as well as all the other parameters are constant, the thesis work is carried out here. In this thesis, performance analysis among stream cipher RC4 variants is done. Selecting algorithm which can encrypt with high speed only is not matter but security is also. Security has been a great thread and challenge for entire field of cryptography. Similarly, there are some other parameters and cases as well which should be also counted while analyzing these algorithms. So, in future work, it can be the study to optimize and find better algorithm by improving security issue.

REFERENCES

[1] ‘‘A. Maximov and D. Khovratovich. *“New State Recovery Attack on RC4.”* In CRYPTO 2008, LNCS, Vol. 5157, pp. 297-316.

[2] *“A note on random number generation”*, [online] available: https://cran.r-project.org/web/packages/randtoolbox/vignettes/fullpres.pdf

[3] B. Zoltak. *“VMPC One-Way Function and Stream Cipher.”* In proceedings of FSE 2004, LNCS, Springer, Vol. 3017, pp. 210{225, 2004.

[4] C. Paar, J. Pelzl, *Understanding Cryptography*, DOI 10.1007/978-3-642-04101-32,_c Springer-Verlag Berlin Heidelberg 2010

[5] *“Cryptography just for beginners”*, [online] available: http://www.tutorialspoint. .com/

[6] Guido Bertoni, Joan Daemen, Michael Peeters, and Gilles Van Assche. *“Cryptographic sponge functions.”* http://sponge.noekeon.org/, January14, 2011. Version 0.1.

[7] H.C.A.V. Tilborg, *Fundamentals of Cryptology*, Kluwer Academic Publisher Boston , 1988

[8] I. Mantin and A. Shamir. *“A Practical Attack on Broadcast RC4.”* In FSE 2001, LNCS, Vol. 2355, pp. 152-164.

[9] Mantin, *“Analysis of the stream cipher RC4, Master’s thesis”,* The Weizmann Institute of Science, 2001

[10] M. Bellare, P. Rogaway, and D. Wagner. EAX *“A conventional authenticated-encryption mode.”* (rev. 2003/9/9). Cryptology ePrint Archive, Report 2003/069, April 13, 2003. eprint.iacr. org/2003/069.

[11] NIST. *“Recommendation for random number generation using deterministic random bit generators.”* Technical Report SP 800-90 A Rev. 1 (Draft), NIST, 2013.http://csrc.nist.gov/publications/drafts/800-90/draft_sp800_90a_rev1.pdf. 14

[12] “Netbeans IDE Features”,[online]available: http://netbe-ans.org/features/index.html

[13] “One-Time-Pad”,[online]available: https://www.ranum.com/security/computer_security/papers/otp-faq/

[14] R. Rivest and J. Schuldt. *“Spritz - a Spongy RC4-like Stream Cipher and Hash Function.”,*[online] available : https://people.csail.mit.edu/rivest/pubs/RS14.pdf

[15] Ronald L. Rivest. *"RSA security response to weaknesses in key scheduling algorithm of RC4".* Technical note, RSA Data Security, Inc., 2001.

[16] Rogaway, P., Black, J., *A block- cipher mode operation for parallelizable message Authentication.* Advances in cryptology – EUROCRYPTO 2002, LNCS 2332, pp. 384-397, Springer – Verlag, 2002.

[17] "Sun Microsystems Inc.",[online] available : http://java.sun.com/javame

[18] S. Maitra and G. Paul. *"Analysis of RC4 and Proposal of Additional Layers for Better Security Margin." I*n INDOCRYPT 2008, LNCS, Vol. 5365, pp. 27{39, 2008

[19] S. Paul and B. Preneel. *"A New Weakness in the RC4 Keystream Generator and an Approach to Improve the Security of the Cipher."* In FSE 2004, LNCS, Vol. 3017, pp.

[20] *"True Random number generation"*, [online] available : https://koclab.cs.ucsb.edu/d oc/ koc/b08.pdf

[21] "VMPC stream cipher",[online] available: http://www.vmpcfunction.com/function. htm

[22] William Stallings *Cryptography And Network Security Principles and Practice*, Prentice hall , Fifth Edition ,2010

[23] Yashir Nawaz , Design of stream Ciphers & Cryptographic Properties of Nonlinear functions, Doctor of Philosophy thesis in Electrical & Computer Engineering, University of Waterloo, Waterloo Canada,(2007)

APPENDIX

Code of Implementation

VMPC

Important functions and java coding are given as below for VMPC algorithm.

```
public void init(boolean forEncryption, CipherParameters params)
{
    if (!(params instanceof ParametersWithIV))
    {
        throw new IllegalArgumentException(
            "VMPC init parameters must include an IV");
    }

    ParametersWithIV ivParams = (ParametersWithIV) params;

    if (!(ivParams.getParameters() instanceof KeyParameter))
    {
        throw new IllegalArgumentException(
            "VMPC init parameters must include a key");
    }

    KeyParameter key = (KeyParameter) ivParams.getParameters();

    this.workingIV = ivParams.getIV();

    if (workingIV == null || workingIV.length < 1 || workingIV.length > 768)
    {
        throw new IllegalArgumentException("VMPC requires 1 to 768 bytes of IV");
    }
    this.workingKey = key.getKey();
    initKey(this.workingKey, this.workingIV);
}

protected void initKey(byte[] keyBytes, byte[] ivBytes)
{
    s = 0;
    P = new byte[256];
    for (int i = 0; i < 256; i++)
    {
        P[i] = (byte) i;
    }

    for (int m = 0; m < 768; m++)
    {
        s = P[(s + P[m & 0xff] + keyBytes[m % keyBytes.length]) & 0xff];
        byte temp = P[m & 0xff];
        P[m & 0xff] = P[s & 0xff];
        P[s & 0xff] = temp;
    }
    for (int m = 0; m < 768; m++)
    {
        s = P[(s + P[m & 0xff] + ivBytes[m % ivBytes.length]) & 0xff];
        byte temp = P[m & 0xff];
        P[m & 0xff] = P[s & 0xff];
        P[s & 0xff] = temp;
    }
    n = 0;
}

public int processBytes(byte[] in, int inOff, int len, byte[] out,
    int outOff)
{
    if ((inOff + len) > in.length)
    {
        throw new DataLengthException("input buffer too short");
    }

    if ((outOff + len) > out.length)
    {
        throw new OutputLengthException("output buffer too short");
    }

    for (int i = 0; i < len; i++)
    {
        s = P[(s + P[n & 0xff]) & 0xff];
        byte z = P[(P[(P[s & 0xff]) & 0xff] + 1) & 0xff];
        // encryption
        byte temp = P[n & 0xff];
        P[n & 0xff] = P[s & 0xff];
        P[s & 0xff] = temp;
        n = (byte) ((n + 1) & 0xff);

        // xor
        out[i + outOff] = (byte) (in[i + inOff] ^ z);
    }

    return len;
}

public void reset()
{
    initKey(this.workingKey, this.workingIV);
}

public byte returnByte(byte in)
{
        s = P[(s + P[n & 0xff]) & 0xff];
        byte z = P[(P[(P[s & 0xff]) & 0xff] + 1) & 0xff];
        // encryption
        byte temp = P[n & 0xff];
        P[n & 0xff] = P[s & 0xff];
        P[s & 0xff] = temp;
        n = (byte) ((n + 1) & 0xff);

        // xor
        return (byte) (in ^ z);
    }
}
```

SPRITZ

Important functions and java coding are given as below for VMPC algorithm.

```
public class Spritz {

  private static final int SPRITZ_N=256;

  private int i;
  private int j;
  private int k;
  private int z;
  private int a;
  private int w;

  private int[] S = new int[SPRITZ_N];

  private static enum ProgramMode {
```

```
  ENCRYPT,
  DECRYPT
};

private static final int IV_SIZE=10;

public Spritz() {

  i = 0;
  j = 0;
  k = 0;
  z = 0;
  a = 0;
  w = 1;

  for(int v=0; v < SPRITZ_N; v++) {
    S[v] = v;
  }

}

private int low(byte b) {
  return (b & 0x0F);
}

private int high(byte b) {
  return ((b & 0xFF) >>> 4);
}

private void swap(int a, int b) {
  int tmp = S[a];
  S[a] = S[b];
  S[b] = tmp;
```

```
 }

 public static final int gcd(int x1,int x2) {

  if(x1<0 || x2<0) {
    throw new IllegalArgumentException("Cannot compute the GCD if one integer is
negative.");
  }

  int a1,b1,g1,z1;

  if(x1>x2) {
    a1 = x1;
    b1 = x2;
  } else {
    a1 = x2;
    b1 = x1;
  }

  if(b1==0) return 0;

  g1 = b1;
  while (g1!=0) {
    z1= a1%g1;
    a1 = g1;
    g1 = z1;
  }
  return a1;
 }

 private void update() {
  i = ((byte) (i + w)) & 0xff;
  j = ((byte) (((byte) (k + S[(byte)(j + S[i]) & 0xff]) & 0xff))) & 0xff;
  k = ((byte) (i + k + S[j])) & 0xff;
  swap(i, j);
}

private void whip(int r) {
  for(int v = 0; v < r; v++) {
    update();
  }

  do {
    w = ((byte) (w + 1)) & 0xff;
  } while(gcd(w, SPRITZ_N) != 1);

}

private void crush() {
  for(int v=0; v < (SPRITZ_N / 2); v++) {
    if((S[v]) > S[SPRITZ_N - 1 - v]) {
      swap(v, SPRITZ_N - 1 - v);
    }
  }
}

private void shuffle() {
  whip(SPRITZ_N * 2);
  crush();
  whip(SPRITZ_N * 2);
  crush();
  whip(SPRITZ_N * 2);
  a = 0;
}

private void absorb_nibble(int x) {
  if(a == (SPRITZ_N/2)) {
    shuffle();
  }

  swap(a, (byte) ((SPRITZ_N/2) + x) & 0xff);

  a = ((byte) (a + 1)) & 0xff;
}

private void absorb_byte(byte b) {
  absorb_nibble(low(b));
  absorb_nibble(high(b));
}

private void absorb(byte[] I) {
  for(int v=0; v < I.length; v++) {
    absorb_byte(I[v]);
  }
}

private void absorb_stop() {
  if(a == SPRITZ_N/2) {
    shuffle();
  }

  a = ((byte) (a + 1)) & 0xff;
}

private byte drip() {
  if(a != 0) {
    shuffle();
  }
  update();

  z = S[(byte)(j + S[(byte)(i + S[(byte)(z + k) & 0xff]) & 0xff]) & 0xff];

  return (byte) z;
}

public static void print_usage() {
  System.out.println("Usage:\n" +
        "  spritz [mode] [key] [input file] [output file]\n" +
        "\n" +
        "Options: \n" +
        "  mode: encrypt / decrypt\n" +
        "  key: encryption key, may be up to 245 bytes long\n" +
        "  input file: the input to read\n" +
        "  output file: the input to write");

}

public static void main(String[] args) {

  if(args.length < 3) {
    print_usage();
    System.exit(0);
  }

  ProgramMode program_action = null;
  byte[] key;
  byte[] iv = new byte[IV_SIZE];
  Spritz spritz;
  InputStream in = null;
  OutputStream out = null;
  int c=0;
```

```
if(args[0].equals("encrypt")) {
  program_action = ProgramMode.ENCRYPT;
} else if(args[0].equals("decrypt")) {
  program_action = ProgramMode.DECRYPT;
} else {
  System.out.println("Unknown mode " + args[0] + ".");
  print_usage();
  System.exit(0);
}

key = args[1].getBytes();

if(args.length >= 3) {
  try {
    in = new FileInputStream(args[2]);
  } catch(Exception ex) {
    System.out.println("Input file " + args[2] + " not found.");
    System.exit(0);
  }
}

if(args.length >= 4) {
  try {
    out = new FileOutputStream(args[3]);
  } catch(Exception ex) {
    System.out.println("Output file " + args[3] + " not found.");
    System.exit(0);
  }
}

if(program_action == ProgramMode.ENCRYPT) {
  SecureRandom random = new SecureRandom();
```

```
      random.nextBytes(iv);
      try {
        out.write(iv);
      } catch(Exception ex) {
        ex.printStackTrace(System.out);
        System.exit(0);
      }
    } else if (program_action == ProgramMode.DECRYPT) {
      int r = 0;
      try {
        r = in.read(iv);
      } catch(Exception ex) {
        ex.printStackTrace(System.out);
        System.exit(0);
      }
      if(r != IV_SIZE) {
        System.out.println("Could not read initialisation vector.");
        System.exit(0);
      }
    }

    spritz = new Spritz();
    spritz.absorb(key);
    spritz.absorb_stop();
    spritz.absorb(iv);

    try {

      c = in.read();

      while(c != -1) {
        byte r=0;
        if(program_action == ProgramMode.ENCRYPT) {
          r = (byte)(c + (spritz.drip() & 0xff));
        } else if (program_action == ProgramMode.DECRYPT) {
          r = (byte)(c - (spritz.drip() & 0xff));
        }

        out.write(r);
        c = in.read();

      }

      in.close();
      out.close();
    } catch(Exception ex) {
      ex.printStackTrace(System.out);
      System.exit(0);
    }

    System.exit(0);
  }

}
```