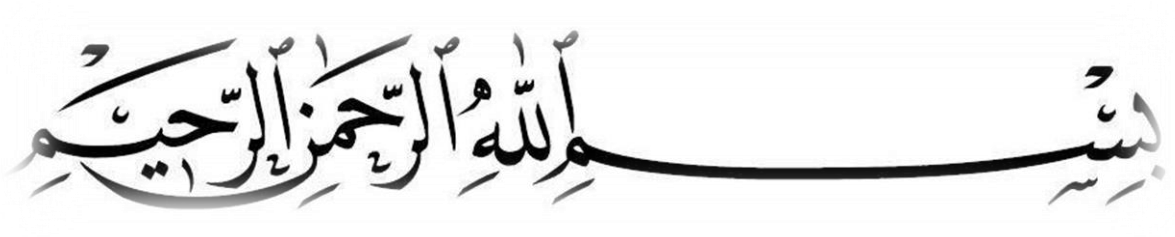


In the name Of Allah, the most beneficent, the eternally merciful

On Advanced Encryption Standard and S-boxes

By

Tanveer ul Haq

Department of Mathematics

Quaid-i-Azam University

Islamabad, Pakistan

2016

On Advanced Encryption Standard and S-boxes

By

Tanveer ul Haq

Supervised By

Prof. Dr. Tariq Shah

Department of Mathematics

Quaid-i-Azam University

Islamabad, Pakistan

2016

On Advanced Encryption Standard and S-boxes

By

Tanveer ul Haq

A thesis submitted in the partial fulfillment of the requirement for the degree of

MASTER OF PHILOSOPHY

in

Mathematics

Supervised By

Prof. Dr. Tariq Shah

Department of Mathematics

Quaid-i-Azam University

Islamabad, Pakistan

2016

Dedicated to

My

Beloved

Parents

Acknowledgement

I am heartily thankful to Dr. Tariq Shah, Professor in the department of Mathematics, Quaid-i-Azam University, Islamabad, for giving me the opportunity to work under his grand supervision.

- I would also like to convey my sincere gratitude to Dr. Ghazanfr Farooq, Assistant Professor in Computer Science department, Quaid-i-Azam University, Islamabad and Professor Dr. Tasawar Hayat, Chairman of the department of mathematics, Quaid-i-Azam University, Islamabad.
- I also like to thanks my brother Ijaz ul Haq, my friend Wahid Ullah, Taimoor Abdullah Khan, Zeeshan Afzal Satti, Atta Ullah and Yasir Naseer for the necessary cooperation in the accomplishment of my thesis.

Last but not the least, I would like to thank my family, who have been a source of encouragement and motivation throughout the duration of the study.

Tanveer ul Haq

Preface

Cryptography plays a fundamental role in the security of data transmission. After the early stages of cryptography i.e. in the Egyptians times, four thousand years before till the 2nd world war, cryptography played a prominent role in the 20th century. Nevertheless, post-World War II, the development of a variety of crypto analysis techniques weakened cryptography of the earlier stages. Accordingly, these techniques break the codes and different algorithms that were secure in the early stages. In 1970's Horst Feistel created a cipher at IBM called the Feistel Cipher and then in 1997 the US National Bureau of Standards (NBS) published a cipher named Data Encryption Standard (DES) [3]. This Cipher was considered to be the best secure algorithm till 1997. It uses a key of length 56-bit which was very small as shown by recent distributed key search exercise [14]. When DES was nearing its end, the US National Institute of Science and Technology (NIST) issued a call for an algorithm which was highly secure, simple and fast called the "Advanced Encryption Standard (AES)". The NIST shortlisted five algorithms, out of which Rijndael algorithm [18] was chosen as an AES. It is a 128-bit block cipher that accepts keys of length 128-bit, 192-bit and 256-bit. This was designed to overcome the issues of secure communication especially on platforms like ATM networks, High Definition Television (HDTV) and Integrated Services Digital Network (ISDN) (see [21]).

Among shortlisted algorithms proposed for Advanced Encryption Standard (AES), the Serpent Algorithm [1] is also included. For Serpent Algorithm, initially S-boxes are taken from DES that resulted in Serpent-0 [13], a more secure Algorithm than triple-DES [13] having a key size of length 192 or 256 bits, presented at the 5th international workshop on Fast Software Encryption [4]. After this, Serpent-1 [1] was designed which used new and stronger S-boxes (taken from DES S-boxes) with a different key schedule in order to resist different attacks like differential [4] and linear [2]. Like Rijndael, Serpent Algorithm was also designed to encrypt a 128-bit block by using keys of length 128-bit, 192-bit or 256-bit. It was especially designed for intel-based chips. Nowadays it is used in folder locks like "Folder Lock Professional", Dropbox file security [23] etc.

There are different ways to modify these algorithms e.g. using Substitution boxes (S-boxes) of good quality depending on their nonlinearity. Furthermore, it also depends on the number of rounds and key schedule.

On the lines of AES S-box, different S-boxes were constructed over finite Galois fields $F_{2^m}, m = 2, 3, 4, 5, 6, 7, 8$, such as residue prime S-box [12], perfect nonlinear S-box [16], Gray S-box [22], APA S-box and S_8 AES S-box [11]. In our proposed Algorithm (Modified Serpent Algorithm), unlike the Serpent-0 and Serpent-1, we used 4×4 S-boxes constructed from a commutative chain ring whose each entry is a byte [19]. Splitting the given key into just two vectors, we calculated approximately half of the pre-keys as compared to Serpent Algorithm [1]. Moreover, the pre-keys calculated in our proposed algorithm are different from those calculated in Serpent Algorithm. For the proposed Modified Serpent Cipher in this study use the same ideas for bit slice implementation of cipher [3] like Serpent-1. Furthermore, unlike the DES that gains extra speed by encrypting 64 different blocks in parallel, each single block of the Serpent Algorithm in this study is efficiently encrypted by bit slicing and hence there is no need of changes for gaining extra speed. The Serpent Algorithm S-boxes are limited to hexadecimal numbers (i.e. both the domain and range are confined to 16 numbers) while in our modified procedure the 4×4 S-box has the property that it take elements from commutative chain ring $R_8 = \frac{\mathbb{Z}_2[x]}{<x^8>} = \mathbb{F}_2 + x\mathbb{F}_2 + x^2\mathbb{F}_2 + x^3\mathbb{F}_2 + x^4\mathbb{F}_2 + x^5\mathbb{F}_2 + x^6\mathbb{F}_2 + x^7\mathbb{F}_2$ having 512 elements, and results again in R_8. This property extends the security of Modified Serpent Algorithm. Using less number of rounds and dealing with 64-bits at a time make the algorithm fast, whereas the main drawback of the Serpent algorithm is its less speed as compared to Rijndael. Consequently, a moderate complexity level is automatically developed.

Contents

Table of Figures:

CHAPTER 1

HISTORY OF CRYPTO-ALGORITHMS

1.1. Ancient Egyptians

The beginning of cryptography occurs from an Egyptian town known as Menet Khufu approximately 4000 years ago. In this age an Egyptian named Khnumhotep use unusual symbols to present usual symbols, which is a type of substitution cipher. They call this substitution as the hieroglyphic substitution. This cipher was helpful for the scribe of those days who want to represent their writing in a formal way. But this method was not secure enough because one who can read and write could easily understand what they have been written.

1.2. Greeks

A device call scytale was developed by Spartans is shown in Fig. 1 below. This is a cylinder shape device which was wounded with a narrow strip made up of parchment. The plaintext was written on the parchment length-wise when the parchment was unwounded the alphabets take the shape of a ciphertext. Nowadays we call these ciphers the transposition ciphers because here only the position of alphabets have been changed. Cipher like this is easily hacked nowadays but in its early ages, it was difficult to decrypt a message without having the same scytale through which it was encrypted.

Fig. 1 Scytale

1.3. Romans

One of the major problem that a nation have to face is the security of their military information and communication. It was the Romans who give cryptography an importance on the military level 2000 year ago. The commander of army Caesar initiates the secure communication between their troops by developing a new technique of cryptography. Caesar for the first time introduces the method of substitution call the Caesar cipher. In this cipher, he replaces one known alphabet by another known alphabet. This cipher gives an advantage to

Roman forces in the war. In this cipher, the concept of a secret key was introduced. The main theme of this cipher was to rotate the English alphabets by a specific number. This number is actually the cipher key. Caesar cipher is shown in the Fig. 2.

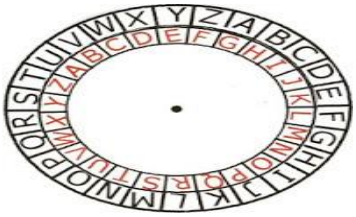


Fig. 2 Caeser Cipher Wheel

1.4. Alberti-Vigenere Cipher

Slowly and steadily the cryptographic algorithms were blooming on. After the Romans in 14th century mid "Alberti" (Italian) gives the concept of a polyalphabetic cipher. He uses a mechanical device consisting of many disks which allowed to many different substitution methods.

In the 15th century, Blaise De Vigenere introduces a cipher called Vigenere Cipher. The main idea uses by Vigenere was the same like Alberti cipher. The only difference from Caesar cipher is that of the key. Throughout the encryption and decryption process, the key has been changed by the Vigenere in order that the cipher became more secure. He uses the following table to secure the data:

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A
B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B
C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C
D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D
E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E
F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F
G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G
H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H
I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I
J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J
K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K
L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L
M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M
N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N

O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P
Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q
R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R
S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S
T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T
U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U
V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V
W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W
X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X
Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y
Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z

Vigenere Table

The substitution under the cipher occurs in a simple way. The first column represents alphabets of plaintext while the first row is that of the given key. While their intersection gives the ciphertext. The key is used repeatedly again and again until all the plaintext alphabets are encrypted for example:

Plaintext: PAKISTAN

Key: KPK

Ciphertext: AQVTIEKD

1.5. Jefferson Wheel Cipher

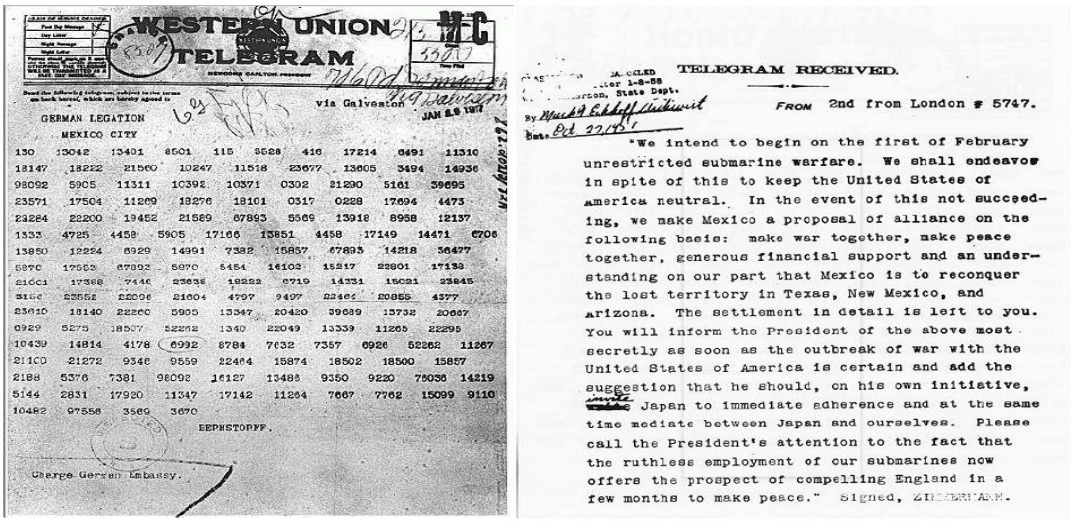
In between 1700's and 1800's Jefferson modified the Vigenere cipher by inventing a 26 wheel instrument having randomly scattered alphabet on each wheel. While the role of a key is played by the numbers given to each wheel. In encryption process, the plaintext lined up on the wheels to obtain a cipher text. The ciphertext lies in a row above or below the plaintext for example:

Fig. 3 Jefferson Cipher Wheel

In 19th century army of the United States reinvented the cipher without any previous knowledge because the Jefferson doesn't develop his cryptographic system. The American army uses this system for his data security from 1923 to 1942.

1.6. World War 1

Cryptography played an important role in the World War 1 (WW1) when the British army caught the telegram of Germans. This telegram was encoded by Zimmerman. The British hack and decode the message which results the USA's to be with the British. The encrypted and decrypted telegram of Zimmerman is shown in the Fig. 4.


WESTERN UNION TELEGRAM

via Galveston

JAN 16 1917

GERMAN LEGATION
MEXICO CITY

130 13042 13401 8501 115 9528 416 17214 6491 11310
18147 18222 21560 10247 11518 23677 13605 3494 14936
98092 5905 11311 10392 10371 0302 21290 5161 39695
23571 17504 11269 18276 18101 0317 0228 17694 4473
23284 22200 19482 21569 67893 5569 13918 8958 12137
1333 4725 4458 5905 17166 13851 4458 17149 14471 6706
13850 12224 6929 14991 7382 15857 67893 14218 36477
5870 17553 67893 5870 5454 16102 15217 22801 17138
21001 17388 7110 23538 18222 6719 14331 15021 23845
3156 22551 22096 21604 4797 9407 22161 20855 4377
23610 18140 22260 5905 13347 20420 39039 13732 20667
0929 5275 18507 52262 1340 22049 13339 11265 22295
10439 14814 4178 6992 8784 7632 7357 0926 52262 11267
21100 21272 9346 9559 22464 15874 18502 18500 15857
2188 5376 7381 98092 16127 13486 9350 9220 75036 14219
5144 2831 17920 11347 17142 11264 7667 7762 15099 9110
10482 97556 3569 3670

BEPNSTORFF.

Charge German Embassy.

TELEGRAM RECEIVED.

FROM 2nd from London # 5747.

"We intend to begin on the first of February unrestricted submarine warfare. We shall endeavor in spite of this to keep the United States of America neutral. In the event of this not succeeding, we make Mexico a proposal of alliance on the following basis: make war together, make peace together, generous financial support and an understanding on our part that Mexico is to reconquer the lost territory in Texas, New Mexico, and Arizona. The settlement in detail is left to you. You will inform the President of the above most secretly as soon as the outbreak of war with the United States of America is certain and add the suggestion that he should, on his own initiative, invite Japan to immediate adherence and at the same time mediate between Japan and ourselves. Please call the President's attention to the fact that the ruthless employment of our submarines now offers the prospect of compelling England in a few months to make peace." Signed, ZIMMERMANN.

Fig. 4 Zimmerman Encrypted and Decrypted Telegram

War Driven Cryptography:

1.7. Enigma Machine

In $the\ 19^{th}$ century Arthur Scherbius a German engineer invented the enigma machine for the encryption and decryption purposes of data. The encryption of enigma was strong enough because the enigma allows 10^{114} possible configurations. This machine gives enough strength to Germany during World War2. But the over confidence of Germans on Enigma Machine fall them in a result of lose of the WW2. Many encryptions made by Germans through enigma was deciphered by the allied forces resulting in the victory of the Allied Forces. Enigma machine is shown in the Fig. 5 below:

Fig. 5 Enigma Machine

1.8. One-Time Paid

Modern encryption begins in early 1900's when the One-time Pad algorithm was invented. This was the early age of playing with bits in an algorithm. The One-time Pad algorithm was proven to be unbreakable in the beginning. Later on the probabilistic approach decrypt this algorithm. In this algorithm, a plaintext is considered and a key is XORed (Exclusive-or) with it to obtain a ciphertext. In the reverse process, the same key is XORed with the ciphertext to obtain the plaintext. Example is given below:

Let plaintext $= 11101110$ and the given key is 10001000 then the ciphertext becomes:

$$11101110 \oplus 10001000 = 01100110$$

After passing through these stages cryptography arrived at Symmetric and Asymmetric key cryptography which is briefly discussed in the 2^{nd} chapter.

CHAPTER 2

ELEMENTARY CONCEPTS OF ALGEBRAIC STRUCTURES AND CRYPTO-ALGORITHMS

This chapter includes some of the basic concepts of algebra and cryptography. This chapter has been divided into two main section. The first section contains some of the basic concepts of algebra mainly related to algorithms especially in the key part of the algorithm so call Substitution box and the second one contain some basic ideas about cryptography and the cryptographic algorithm.

2.1. Algebraic Structures Basics:

In this section, we present some definition with examples like a ring, field, Galois field, Galois ring, chain ring and some other preliminaries which help us to understand the algorithms properly.

Binary operation:

A binary operation is a mapping $* : A \times A \to A$, where A is a non-empty set, defined by:

$*(a, b) = a * b \in A$ for all $a, b \in A$

Groupied:

It is a closed non-empty set i.e. if A is a non-empty set with the property that $a * b \in A$ for all $a, b \in A$ then A is closed and hence called a groupied. For example:

The binary operation $*$ on the set of rational numbers Q defined by $a * b = \frac{a-b}{2}$ for all $a, b \in Q$.

Semigroup:

A closed non-empty set A that holds Associative property w.r.t a given Binary operation is given the name of Semigroup e.g. the set containing elements of the form 2n where n belongs to the set of natural numbers.

Monoid:

A closed set containing identity (e is the identity of A if $e * x = x * e = x$ for all $x \in A$) w.r.t to the operation $*$ and satisfying the associative law under the same binary operation $*$ is given the name of monoid e.g. the set of natural number form a monoid.

Group:

A monoid with an additional property that it's each element has inverse i.e. $x * x^{-1} = x^{-1} * x = e$ (e is the identity w.r.t $*$) is called a group. Examples of group are:
The set of real numbers with the operation $* = +$ (the usual addition), the set of rational numbers w.r.t $+$, the set of integer's w.r.t. + etc. form a group. The set of all $n \times n$ invertible matrices $GL(n, R)$ form a group under the binary operation $* = \cdot$, the usual multiplication. This group is called General Linear Group.

Subgroup:

Any non-empty subset of a group which itself form a group under the same binary operation define on that group is given the name of a subgroup. For example:
$(Q\backslash 0, \cdot)$ is a subgroup of $(R\backslash 0. \cdot)$.

Ring:

A non-empty set R with two binary operations + and · is called a ring if:

i) $(R, +)$ is a group with the property that $ab = ba$, for all $a, b \in R$
ii) $(R, \cdot)$ is a semi group.

In addition with this R satisfies the right and left distributive law of multiplication over addition.

Field:

A non-empty set F with two binary operations $+$ and $\cdot$ is called a field if $(F, +)$ form a group and $(F\backslash 0, \cdot)$ form a group, also it satisfies distributive law e.g. set of real numbers.

Commutative Ring:

If R has the property that $x \cdot y = y \cdot x \; \forall \; x, y \in R$ then it is commutative.

Unit element:

An element $x \in R$, where R is a commutative ring with identity, is said to be a unit element if there exists an element $y \in R$ such that $xy = yx = 1$. The set unit elements of R is represented by $U(R)$.

Ideal:

A non-empty subset S of R is termed to be an ideal if it satisfies the following properties:

i) $a - b \in S \; for \; all \; a, b \in S$

ii) $rs \in S, sr \in S \; for \; all \; r \in R \; \& \; s \in S$

For example, $\mathbb{Z}/9\mathbb{Z} \cong \mathbb{Z}_9$ has an ideal $3\mathbb{Z}/9\mathbb{Z}$

Irreducible polynomial:

Polynomial who's splitting up is not possible.

Finite ring:

A ring R is finite if it has a finite number of elements for example $\mathbb{Z}_9$.

Finite Local Ring:

Consider a finite Ring R (R is commutative) with identity. We call R a local finite ring if its subset of all the non-invertible elements is closed under addition for example:
$\mathbb{Z}_4$

Chain Ring:

A finite local ring R whose radical M form a principal ideal is called a chain ring for example $\frac{\mathbb{Z}_2[x]}{<x^3>} = \mathbb{F}_2 + x\mathbb{F}_2 + x^2\mathbb{F}_2$ and $\frac{GF(q)[x]}{<x^k>} = \sum_{i=0}^{k-1} x^i GF(q)$ Where $q = p^r$, p is a prime number and r is any positive integer. The polynomial $x^k \in GF(q)[x]$.

Galois Ring:

A finite polynomial ring R with coefficients from $\mathbb{Z}_{p^k}$ when factored by an ideal generated by an irreducible polynomial $f(x)$ in $\mathbb{Z}_{p^k}[x]$ is said to be Galois ring for example:

$$\frac{\mathbb{Z}_{2^2}[x]}{< x^2 + x + 1 >}$$

Galois Field:

A finite polynomial field with coefficients from $\mathbb{Z}_p$, p is a prime, when factored by an ideal generated by irreducible polynomial f(x) in $\mathbb{Z}_p[x]$ is called Galois field for example:

$$\frac{\mathbb{Z}_2[x]}{< x^2 + x + 1 >}$$

2.2. Cryptography

Cryptology

The science of encryption and decryption of algorithms and their analysis is given the name of cryptology. Here the first part forms Cryptography while the second Cryptanalysis. Furthermore, cryptography is divided into two main branches depending on the key schedule routine named Symmetric key cryptography and Asymmetric key cryptography.

Symmetric and Asymmetric key cryptography

On some occasions we use a single key to encrypt a message and the same key is used for its decryption process while in other cases a different key is used for its decryption procedure we call it as Symmetric and Asymmetric key cryptography.

2.3. Notions used in Crypto-Algorithms

The following ideas are used in algorithms:

Bit

Every binary representation of a number is a combination of two distinct digits 0 and 1 known as bit. Whereas the collection of four digits form a nibble and that of eight form a byte. e.g. 0110 is a nibble and 10011001 is a byte.

Bit Rotation

Bit rotation or circular shift is the rearrangement of bits in a way that the last bit takes the position of first bit while all the other bits are moved to their next position, this type of bit rotation is called right circular shift while if the first bit is moved to the last position and all the other bits are moved to their previous position then this types of bit rotation is called left circular shift. >>> and <<< represents the right and left circular shift respectively e.g.: $10011001 <<< 1 = 00110011$ and $10011001 >>> 1 = 11001100$

Most Significant and least significant Bit

In a collection of numbers there are some bits which start that combination while other ends up it, these bits are called least significant and most significant bits respectively e.g.:

10000000010 has 0 as its least significant bit while 1 as its most significant bit.

This thesis mainly includes algorithms. Some of the basic concepts for an algorithm are:

a) Plaintext: the original message given to cipher is called plaintext.
b) Ciphertext: The text received after passing the plaintext through the cipher is known as cipher text.
c) Encryption: The process by which a plaintext is converted into a ciphertext is called encryption.
d) Decryption: the process of converting ciphertext to plaintext is called deciphering or decryption.
e) Cipher: A Cipher is a particular system of Encryption. The figure shows plaintext, cipher, and ciphertext.

f) Stream Cipher: In some ciphers, we treats single bit at a time and processed it this type of cipher is called Stream cipher.
g) Block Cipher: Dealing with a combination of bits in a cipher instead of a single bit at a time is given the name of Block cipher.
h) Algorithm: A step-by-step operation followed in a cipher is called an algorithm. Examples of algorithms are RC6 algorithm [17], Rijndael algorithm [18] and MC6 algorithm.
i) Cipher Key: In an algorithm, there is a given key known as cipher Key. This key is used to generate subkeys used in an algorithm.
j) Subkeys: Keys obtained from the given cipher key by using a key expansion routine.
k) Avalanche effect: Some time whenever we perform an operation on an array of bytes the resultant obtained is totally different this effect of a large change in output due to a small change in input is called avalanche effect.

l) Diffusion: The property of changing a single character in plaintext or ciphertext resulting a large change in the characters of ciphertext or plaintext respectively.

m) Confusion: It is a process in which each character of ciphertext depends on a number of parts of the key.

2.4. Crypto-Algorithms (Description of DES and AES proposed Algorithms)

Most of the time we need to send a message secretly which is possible only when it is shared in between limited and trusty persons. The person who send a text secretly is called the sender while the other one who receive it is called the receiver. This art of secret message transferring is given the name of cryptography. Nowadays this communication is frequently made on computers. In the beginning (i.e. in 1960's) of cryptography, the secret communication was limited to government. In 1970's horst Feistel (German Cryptographer) created a cipher at IBM called the Feistel cipher. This was the first commercially seen cipher of the cryptographic history seen in 1973. The U.S National Bureau of Standards (NBS), now call the National Institute of Standards and Technology (NIST), published symmetric cipher in 1977 based on the Feistel cipher called the Data Encryption Standard (DES). It was considered to be highly secure and as a standard up to the end of 20^{th} century.

The Data encryption standard also called the data encryption algorithm was a Federal Information Processing Standards (FIPS). As we know that Cryptography has two main types "Symmetric key cryptography" and "Asymmetric key cryptography" and DES has just a single Key there for it lie in the first type. It was designed for the National Bureau of Standards (NBS) by International Business Machines (IBM) in 1976 and was considered one of the best algorithm until $the\ 20^{th}$ century. In this algorithm a 64-bit string is encrypted with the help of a secret key which is of 56-bit to obtain a cipher text of length 64-bit. In this algorithm, an Initial permutation is applied to the plaintext to split it to two words i.e. L_0R_0. 16 subkeys each of length 48-bit are computed as a function of the given key K and with the help of these subkey $L_{16}R_{16}$ are find out and then apply the inverse linear permutation on their swapped result to get the cipher text. The main deficiency of this algorithm was its small key length which gives help for the cryptanalysts decrypt any secret message which was encrypted by the DES. In the Electronic Frontier Foundation and distributed.net DES secret Key were break within 22 hours and 15 minutes which is one the big achievement for the cryptanalysts in 1999.

In 1997 NIST (National Institute of Standards and Technology) call for ciphers, because of the theoretical and exhaustive key search attacks on DES. In June 1998 fifteen candidates were accepted and after shortlisting in aug-1999, five were chosen. The shortlist include:

- Rijndael represented by Joan Daemen and Vincient Rijmen from RSA (lab) [8].
- RC6 [17] was presented by Rivest et al. (USA)
- Serpent [1] was presented from Euro by Lars Knudsen, Ross Aderson and Eli Biham.
- MARS [5] was presented by IBM Corporation.
- TwoFish [18] was published in 1998 by Jhon Kelay, David Wazner, Niels Furguson, Bruce Schneier, Chris Hall and Doug Whiting.
  Rijndael, a fast symmetric cryptosystem, was chosen as an AES algorithm in $oct - 2000$.

Requirements of AES:

- A Block cipher that encrypts $128 - bit$ block of a plaintext.
- Applicable for a key of length 128, 192 and $256 - bit$.
- Highly secure, Fast and of less complexity.
- Strong immunity to all well-known attacks.
- Efficiency in hardware and software.
- The active life of minimum 20 years.

The proposed algorithms for AES are discussed as under:

2.4.1. Rijndael Algorithm

This algorithm satisfies all the requirements of AES. It encrypts a 128-bit block. In this case, the plain text and cipher text are each of length 128-bit. In this algorithm firstly we convert any secret message into bytes with the help of ASCII system. The plain text of 16 bytes is written in state and Exclusive-Ored (XOR) it with the supposed key which also consists of 16 bytes. The resultant is passed from round 1. In this algorithm, the number of rounds depends on the Key size. For $128 - bit$, $192 - bit$ and $256 - bit$ $10, 12$ and 14 rounds are performed respectively. Each round consists of the following stages:

i. Sub bytes: the S-box convert each byte to another byte.
ii. Shift Row: A permutation process.
iii. Mix column: A substitution using $GF(2^8)$.
iv. Add round key: XOR of round key and Mix column.

v. The detailed structure of Rijndael algorithm [18] is given below:

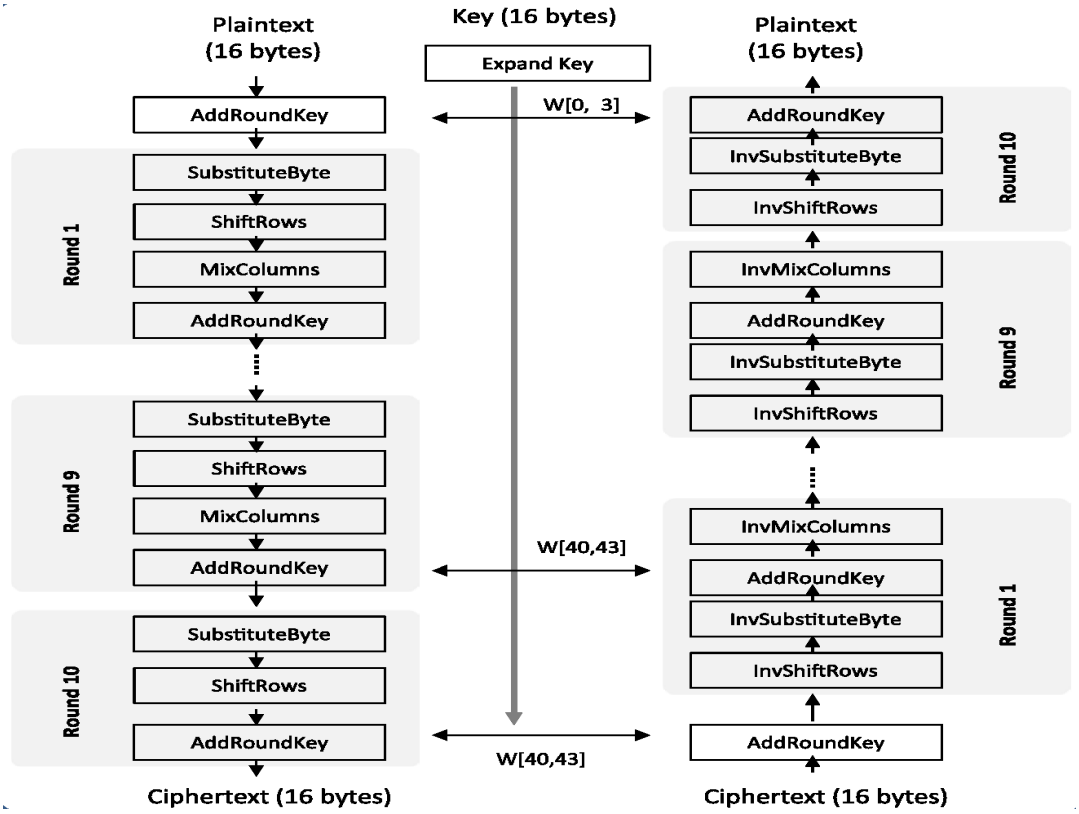


Fig. 6 Advanced Encryption Standard Algorithm flow chart [14]

Stage-i (Sub Byte):

In this stage, each given byte is replaced by another byte taken from an 8×8 substitution box given by Rijndael [18] which is calculated from the transformation $B' = AX^{-1} + B$. Where A is a fix 8×8 Matrix having entries in bits, X^{-1} is the inverse of a byte X which represents a column vector whereas B is a fix column matrix. This transformation occurs in the form that the first nibble of each byte represents the x-axis while the 2^{nd} nibble represents the y-axis and the intersection value of these nibbles gives the sub-byte.

Stage-ii (Shift Row):

SHIFT ROW TRANSFORMATION:

In this process 1^{st} row is unaltered. 1^{st} element of 2^{nd} row goes to the extreme and the remaining 3 bytes shifts to left. Similarly, in 3^{rd} row 3-bytes goes to the extreme and the remaining 2-bytes and 1-byte respectively shift to the left. Consider the example:

$$\begin{pmatrix} 7C & 77 & 7B & F2 \\ 6B & 6F & C5 & 30 \\ 62 & 91 & 95 & E4 \\ EA & 65 & 7A & AE \end{pmatrix} \longrightarrow \begin{pmatrix} 7C & 77 & 7B & F2 \\ 6F & C5 & 30 & 6B \\ 95 & E4 & 62 & 91 \\ AE & EA & 65 & 7A \end{pmatrix}$$

Stage-iii (Mix column):

In the Mix column operation, each row value after transformation mapped to a new value which is the combination of all entries in that column. Each column is multiplied with a fixed matrix (given below) to get a combinatory entry of that column.

$$\begin{pmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{pmatrix}$$

Where multiplication and addition are performed in $GF(2^8)$ e.g. if the entries 87, 6E, 46 and A6 are the entries of a column then its multiplication results in 47.

Stage-iv (Add round key):

In this stage we XORed the result coming from Mix column operation with the subkeys obtained from the given key and thus we got an output of the first round.

This process (four stages) is repeated a total of 10 times called the rounds for 128-bit key. In the last round, Mixcolumn operation is skipped out and results in the shape of a cipher text.

2.4.2. TwoFish Algorithm

The carefully designed TwoFish algorithm [18] was also one the proposed and even the shortlisted AES algorithm. It takes a 128-bit block as an input and produces an output of length 128-bit. All of the processes in this algorithm occurs in the little endian convention. Firstly 128-bit is split into four words and then XORed with the subkeys generated from the given key. This step is called the input whitening. The whole process is illustrated in the Fig. 7 below:

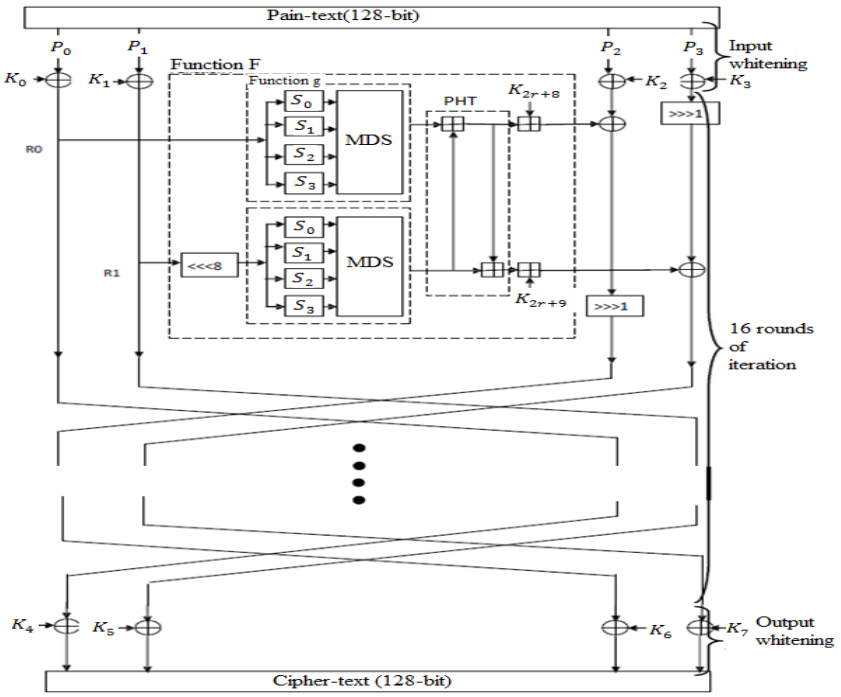


Fig. 7 TwoFish Algorithm flow chart [9]

After the input whitening step, this algorithm uses a Feistel function F (bijective) in its rounds. A total of 12 or 16 or 20 rounds is performed depending on the key size. The bijective function F further comprises of rotation of bits, g-function, Pseudo-Hadamard transform (PHT) and Key addition under modulo 2^{32}. The use of substitution boxes mainly occur in the g-function and also this function includes a Maximum Distance Separable (MDS) matrix. Entries of these look-up tables are in nibbles but the substitution occurs here in bytes. The responsible fact for this situation is Q-permutation. If x is a byte then the Q-permutations are given as:

a_0 = [x / 16] and b_0 = x mod 16

i.e. the byte is first split into two 4-bit nibbles, a_0 and b_0

$a_1 = a_0 \oplus b_0$

$b_1 = a_0 \oplus ROR(b_0, 1) \oplus (8a_0 \bmod 16)$

$a_2 = t_0[a_1]$ (t_i are the look-up tables)

$b_2 = t_1[b_1]$

$a_3 = a_2 \oplus b_2$

$b_3 = a_2 \oplus ROR(b_2, 1) \oplus (8a_2 \bmod 16)$

$a_4 = t_2[a_3]$

$b_4 = t_3[b_3]$

$y = 16b_4 + a_4$ (Output of Q-permutation)

The internal structure is of S-boxes are shown below:

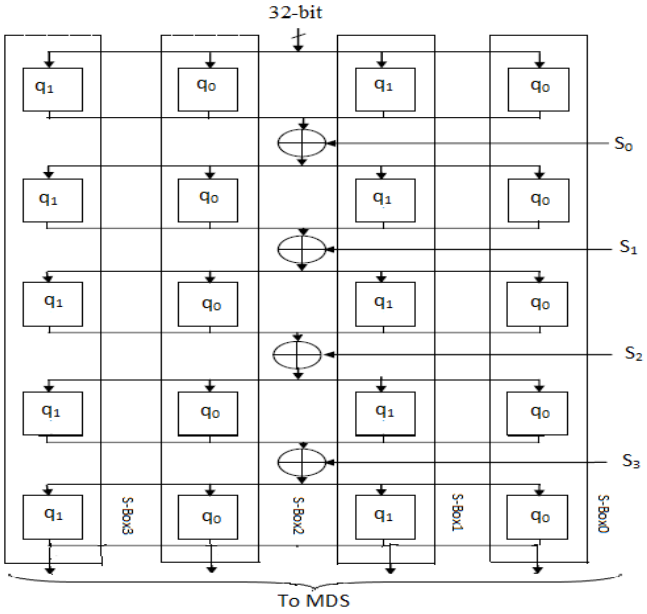


Fig. 8 Internal structure of S-box of TwoFish Algorithm [9]

As four look-up tables are used in a single round and for 128-bit key the number of rounds are twelve so a total of 48 look-up tables are used to encrypt a 128-bit block while in the case of Rijndael it was only 10 which makes Rijndael better than TwoFish in hardware implementation.

2.4.3. RC6 Algorithm

RC6 [17] is one of the proposed shortlisted algorithm having a unique structure from the other four because of the excludeness of any substitution box. This is the improved version of the RC5 algorithm. Like other proposed algorithms it also encrypts a 128-bit block by using a variable key of length up 2040 bits in 20 rounds. It uses a Feistel-cipher structure. Generally, this cipher is denoted by RC6-w/r/b [17] where w is a 32-bit word, r represent a number of rounds and b stand for number of bytes of the given key. The flow chart of RC6 is given Fig. 9 below:

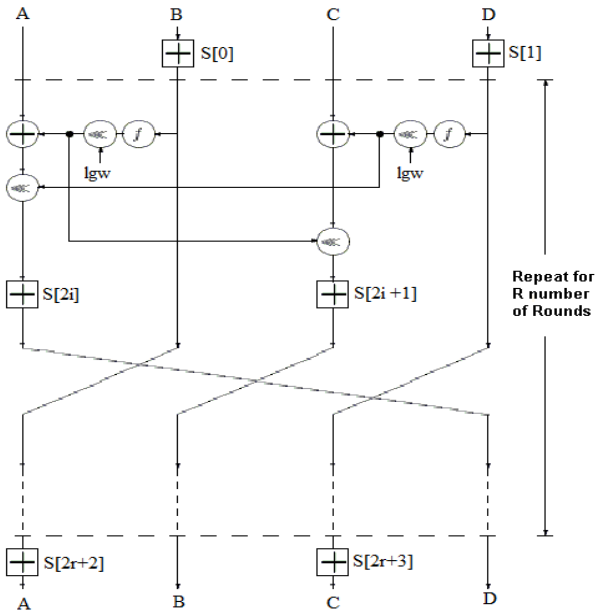


Fig. 9 RC-6 Algorithm flow chart [7]

In this algorithm the plain text is split into four w-bit words A, B, C and D. S[0] and S[1] are added with B and D under modulo 2^w respectively. Where $S[0] = P_w$ and $S[i] = S[i-1] + Q_w$. And P_w and Q_w are magic constant use to calculate key expansion algorithm in the following manner:

$P_w = odd((e-2)2^w)$ e=2.7182818 (base of natural logarithm)

$Q_w = odd((\phi - 1)2^w)$ ϕ=1.618033988 (called golden ratio)

$B = B + S[0]$

$D = D + S[1]$

$for\ i = 1\ to\ r\ do$

{

$t = (B \times (2B + 1)) <<< lg\ w$

$u = (D \times (2D + 1)) <<< lg\ w$

$A = ((A \oplus t) <<< u) + S[2i]$

$C = ((C \oplus u) <<< t) + S[2i + 1]$

$(A, B, C, D) = (B, C, D, A)$

}

$A = A + S[2r + 2]$

$C = C + S[2r + 3]$

These complex arithmetic operations takes long time as compared to the other algorithms therefore it was not suitable for hardware implementation and hence is not chosen as an AES.

2.4.4. Serpent Algorithm

All the five algorithm have their own identification in the field of cryptography but serpent algorithm has a unique identification due to its high speed. Although it is faster than DES but performing 32 number of rounds its speed become slow enough so that it fails to become an AES. An initial permutation is applied on a plain text to get the first round input. This input is XORed with a subkey, passed through an S-box and then applied a linear transformation to get the first round output. In the 32nd round an extra subkey is added and after passing this through the final permutation we get the cipher text. This algorithm is briefly described in the second chapter.

2.4.5. Mars Algorithm

It was also a proposed symmetric key block cipher for AES. It performs the Encryption/Decryption process on a plaintext of 128-bit. Split it into four w-bit words and the operations are performed on 32-bit word. This is a 16 round Feistel network surrounding all the rounds by two layers, the forward and backward mixing. Both the layers consists of 8 rounds.

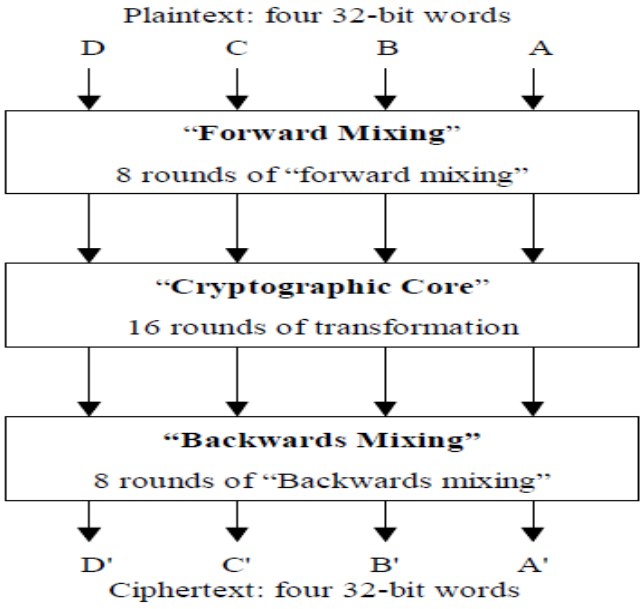


Fig. 10 Mars Algorithm flow chart [5]

It accepts a key of length varying between 4 and 14 words and expand the user supplied key to forty subkeys each of length 32-bit. Each of the input w-bit words are passed through the forward mixing step then through the 16 rounds and at last from the backward mixing step to produce the cipher text. MARS [5] uses a look-up table consisting of 512 w-bit words (Sometimes it is considered as two S-boxes S_0 *and* S_1 each of length 16×16 to oppose different known attacks like linear [15]

and differential attack. The operation used in the three steps are: addition, subtraction and multiplication over modulo2^{32}, exclusive-or, fix and data dependent rotation. The encryption procedure of MARS algorithm [5] is as under:

‘$\boldsymbol{Forward\ Mixing}$
$(A,B,C,D) = (A,B,C,D) + (K[0],K[1],K[2],K[3])$
$\boldsymbol{For\ i = 0\ to\ 7\ do}\{$
$B = (B \oplus S_0[A]) + S_1[A >>> 8]$
$C = C + S_0[A >>> 16]$
$D = D \oplus S_1[A >>> 24]$
$A = (A >>> 24) + B(if\ i = 1,5) + D(if\ i = 0,4)$
$(A,B,C,D) = (B,C,D,A)$
}

$\boldsymbol{Cryptographic\ Core}$
$\boldsymbol{For\ i = 0\ to\ 15\ do}\{$
$R = ((A <<< 13) \times K[2i+5]) <<< 10$
$M = (A + K[2i+4]) <<< (low\ 5\ bits\ of\ (R >>> 5))$
$L = (S[M] \oplus (R >>> 5) \oplus R) <<< (low\ 5\ bits\ of\ R)$
$B = B + L(if\ i < 8) \oplus R(if\ i \geq 8)$
$C = C + M$
$D = D \oplus R(if\ i < 8) + L(if\ i \geq 8)$
$(A,B,C,D) = (B,C,D,A <<< 13)$
}

$\boldsymbol{Backwards\ Mixing}$
$\boldsymbol{For\ i = 0\ to\ 7\ do}\{$
$A = A - B(if\ i = 3,7) - D(if\ i = 2,6)$
$B = B \oplus S_1[A]$
$C = C - S_0[A <<< 8]$
$D = (D - S_1[A <<< 16]) \oplus S_0[A <<< 24]$
$(A,B,C,D) = (B,C,D,A <<< 24)$
}
$(A,B,C,D) = (A,B,C,D) - (K[36],K[37],K[38],K[39])$’

Because of the complex arithmetic operation, addition and Multiplication, it was not suitable for hardware implementation and hence is not selected as an AES.

CHAPTER 3

SERPENT ALGORITHM

Data Encryption Standard was used for many applications from 1970's to 1997. Due to its small key length i.e. 56-bit key cryptanalysts break the key very quickly even with in a single day. Triple DES solve the key length problem but even then DES is used for many applications in hardware implementation due to its high speed.

Keeping all the situation in mind, the US National institute of standards and Technology call for a cipher which in highly secure, having high speed and less complex, given the name of Advanced Encryption Standard in 1997. A total of fifteen algorithms was proposed for AES in June 1998. Since AES required a block cipher that encrypts a 128-bit block, accepting a varying key of length 128-bit, 192-bit and 256-bit, highly secure, fast and best implemented to hardware so NIST chooses five of those including Serpent algorithm.

Our second chapter includes a review of Serpent algorithm, while the third chapter is responsible for some improvement in this algorithm.

3.1. Construction of the Serpent Algorithm

Serpent [1], a symmetric block cipher, was designed for AES by Eli Biham (Technion Israeli Institute of Technology), Ross Anderson (University of Cambridge Computer Laboratory) and Lars Knudsen (University of Bergen, Norway). In this Algorithm, a 128-bit block is ciphered by using a key of length 256 bit in 32 different rounds. The first 31 rounds are identical, consisting of the same sequence of elementary operations while the last round differs only in the key schedule. Instead of mixing a single key like in the first 31 rounds, an additional key is mixed in the last round. Hence 33 round keys are required in the whole process that are generated from the external key.

Fig. 11 represents the whole encryption process of Serpent Algorithm.

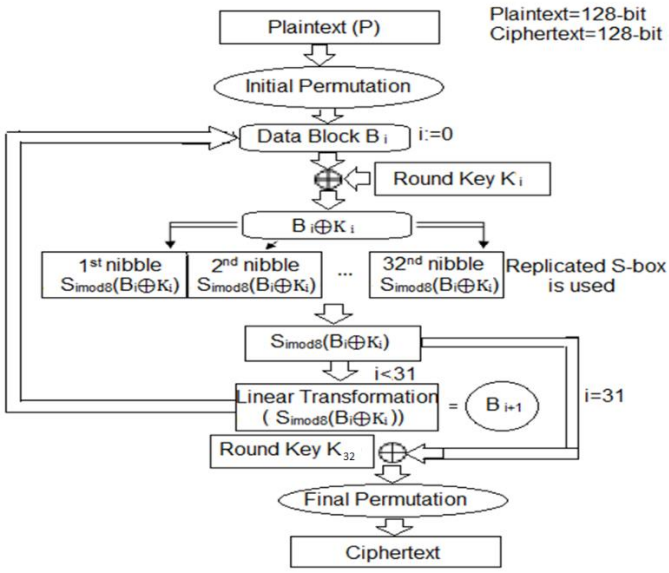


Fig. 11 Serpent Algorithm flow chart [1]

Serpent Algorithm [1] is a block cipher encrypting a 128-bit block of a plaintext by using a key of length 256 bits. The Algorithm consists of three basic functions:

- Initial permutation (IP)
- Round function
- Final permutation (FP)

3.1.1. Initial and Final Permutation

Basically, initial permutation is responsible for change in the position of bits. This change of position is performed by using a fixed table. This table is given as:

0	32	64	96	1	33	65	97	2	34	66	98	3	36	67	99
4	36	68	100	5	37	69	101	6	38	70	102	7	39	71	103
8	40	72	104	9	41	73	105	10	42	74	106	11	43	75	107
12	44	76	108	13	45	77	109	14	46	78	110	15	47	79	111
16	48	80	112	17	49	81	113	18	50	82	114	19	51	83	115
20	52	84	116	21	53	85	117	22	54	86	118	23	55	87	119
24	56	88	120	25	57	89	121	26	58	90	122	27	59	91	123
28	60	92	124	29	61	93	125	30	62	94	126	31	63	95	127

Where the inverse permutation table is given by as follow:

0	4	8	12	16	20	24	28	32	36	40	44	48	52	56	60
64	68	72	76	80	84	88	92	96	100	104	108	112	116	120	124
1	5	9	13	17	21	25	29	33	37	41	45	49	53	57	61
65	69	73	77	81	85	89	93	97	101	105	109	113	117	121	125
2	6	10	14	18	22	26	30	34	38	42	46	50	54	58	62
66	70	74	78	82	86	90	94	98	102	106	110	114	118	122	126
3	7	11	15	19	23	27	31	36	39	43	47	51	55	59	63
67	71	75	79	83	87	91	95	99	103	107	111	115	119	123	127

This Permutation can also be performed by the formula $[(i * 32)\ mod 127]$ where i represents the position of the bit we want to replace by using the table. The inverse permutation can also be performed algorithmically by using the formula $[(i * 4)\ mod 127]$. We note from tables that the first and last bit stay fixed during the transformation. Applying IP to a plaintext, we get a data block B_0 while applying FP on the result of the last round we get the cipher text.

3.1.2. Round Function

Many algorithms use a Substitution Permutation Network (SP-Network). These algorithms include the Serpent algorithm [1]. It operates on four w-bit vectors. These vectors combine to form a 128-bit block. Which we consider as an input to the round. There is a total of 32 rounds performed by serpent algorithm [1]. Firstly an initial permutation IP is applied to the plaintext which produces the data block B_0. This data block plays the role of input in the 1^{st} round. Inside these rounds, each data block B_i is mixed with a subkey K_i (i.e. taking XOR), then pass $B_i \oplus K_i$ through $S_{(imod8)}$ which is one of the eight S-boxes. After this a linear transformation LT is applied to $S_i(B_i \oplus K_i)$ to get $B_{(i+1)}$, where $i = 0, 1, 2, 3, \dots, 30$. In the 32^{nd} round (last round) a 33^{rd} key is XORed instead of applying the linear transformation (LT) i.e. $S_7(B_{31} \oplus K_{31}) \oplus K_{32}$ to get B_{32}. Now the final permutation $[(i * 4)\ mod 127]$ is applied to get the ciphertext. The whole process is described shortly as:

$B_0 = IP\ (P)$

$B_{i+1} = LT(S - box_{imod8}(B_i \oplus K_i))$ Where "i" is from 0 to 30

$B_{32} = S_7\ (B_{31} \oplus K_{31}) \oplus K_{32}$

$C = FP\ (B_{32})$

3.1.3. Elementary Transformations

Key mixing, bit substitution and linear transformation are the basic elementary operations. Key mixing is just an exclusive-or (XOR) of subkey and a data block obtained from the plain text. Bit substitution is simply a substitution from eight different S-boxes while linear transformation consists of XOR, rotation and shift.

Unlike Rijndael [18], Serpent-1 [1] uses 8 different 4×4 S-boxes in which each entry is a nibble and each round uses a single replicated S-box in order to encrypt a 128-bit block. In Serpent-0, the S-boxes used were adopted from DES in order to ensure the high level of public confidence and then in Serpent-1 [1], the new and better S-boxes were generated which had stronger immunity to attacks. For maximal avalanche effect, the author used the XOR operation. Some complex operations (e.g. word addition) were dropped due to high cost in hardware and software implementation [4].

3.1.3.1. The Key Schedule

The Serpent algorithm [1] accepts a key of variable length. It takes a key of length up to 256 bit. If the supplied key is 256 bits, it is divided into eight 32-bit words namedw_{-8}, w_{-7}, …, w_{-1}. From these words, we find the prekeysW_i , where $i = 0, 1, 2, \dots, 131$, by using the following affine recurrence:

$$W_i = (w_{i-8} \oplus w_{i-5} \oplus w_{i-3} \oplus w_{i-1} \oplus \phi \oplus i) <<< 11$$

Where ϕ is the fractional part of golden ratio whose value is given in hexadecimal as $0 \times 9e3779b9$ and " $<<<$ " denotes left shift. From this we can calculate the subkeys, K_i as:

$$K_i = IP(S_{(3-i) mod 8}(w_{4i}, w_{4i+1}, w_{4i+2}, w_{4i+3})); \; i = 0,1,2, \dots, 32.$$

If the key length is less than 256 bit, then it can be mapped to a 256-bit key by writing single "1" at the MCB end proceeded by as many zeroes as required to obtain a 256-bit key [1].

3.1.3.2. Substitution box

In the initial stages i.e. when serpent algorithm [1] was first proposed for AES, the substitution boxes was adapted from DES. The reason for this was that these S-boxes have been studied very well and hence is understood clearly. This results serpent-0 which was as fast as DES and secure as triple-DES. After this new S-boxes were defined which was stronger than the older one. The procedure adapted to obtain these substitution boxes was as follow.
A matrix of 32 arrays (DES S-boxes) was used to obtain these S-boxes, where each array was made up of 16 entries. These rows are transformed by exchanging entries in the i^{th} array depending on $the\ (i+1)$ array and on an initial key in order that it satisfy the desired (linear and differential) properties. The repetition of this procedure ends up when 8 S-boxes have been produced. These S-boxes are given below:

S_0	3	8	15	1	10	6	5	11	14	13	4	2	7	0	9	12
S_1	15	12	2	7	9	0	5	10	1	11	14	8	6	13	3	4
S_2	8	6	7	9	3	12	10	15	13	1	14	4	0	11	5	2
S_3	0	15	11	8	12	9	6	3	13	1	2	4	10	7	5	14
S_4	1	15	8	3	12	0	11	6	2	5	4	10	9	14	7	13
S_5	15	5	2	11	4	10	9	12	0	3	14	8	13	6	7	1
S_6	7	2	12	5	8	4	6	11	14	9	1	15	13	3	10	0
S_7	1	13	15	0	14	8	2	11	7	4	12	10	9	3	5	6

The inverse S-boxes are given below:

$invS_0$	13	3	11	0	10	6	5	12	1	14	4	7	15	9	8	2
$invS_1$	5	8	2	14	15	6	12	3	11	4	7	9	1	13	10	0
$invS_2$	12	9	15	4	11	14	1	2	0	3	6	13	5	8	10	7
$invS_3$	0	9	10	7	11	14	6	13	3	5	12	2	4	8	15	1
$invS_4$	5	0	8	3	10	9	7	14	2	12	11	6	4	15	13	1
$invS_5$	8	15	2	9	4	1	13	14	11	6	5	2	7	12	10	0
$invS_6$	15	10	1	13	5	3	6	0	4	9	14	7	2	12	8	11
$invS_7$	3	0	6	13	9	14	15	8	5	12	11	7	10	1	4	2

3.1.3.3. Linear Transformation

The linear transformation takes an input of 32-bit and produces an input of 32-bit. It consists of Exclusive-or operation, rotation and shifting of bits. Any 32-bit input is first split into four bytes X_0, X_1, X_2 and X_3 and then apply the mentioned operations in the following manner:

$$X_0, X_1, X_2, X_3 \coloneqq S_i(B_i \oplus K_i)$$

$$X_1 \coloneqq X_0 <<< 13$$

$$X_2 \coloneqq X_2 <<< 3$$

$$X_1 \coloneqq X_1 \oplus X_0 \oplus X_2$$

$$X_3 \coloneqq X_3 \oplus X_2 \oplus (X_0 << 3)$$

$$X1 \coloneqq X1 <<< 1$$

$$X_3 \coloneqq X_3 <<< 7$$

$$X_0 \coloneqq X_0 \oplus X_1 \oplus X_3$$

$$X_2 \coloneqq X_2 \oplus X_3 \oplus (X_1 << 7)$$

$$X_0 \coloneqq X_0 <<< 5$$

$$X_2 \coloneqq X_2 <<< 22$$

$$B_{i+1} \coloneqq X_0, X_1, X_2, X_3$$

Where " $\oplus$ " denotes the exclusive-or, " <<< " denote left rotation of bits, " << " is used for left rotation of bits and "B_{i+1}" is the $(i+1)^{th}$ data block.

3.1.3.4. Inverse Linear Transformation

Instead of left shift and left rotation the inverse linear transformation consists of right shift and right rotation. It takes an input of 32-bits and produces an output of 32-bits. The inverse linear transformation is summarized in the below equations:

$X_0, X_1, X_2, X_3 :=$

$X_2 := X_2 >>> 22;$

$X_0 := X_0 >>> 5;$

$X_2 := X_2 \oplus X_3 \oplus (X_1 << 7)$

$X_0 := X_0 \oplus X_1 \oplus X_3;$

$X_3 := X_3 >>> 7;$

$X_1 := X_1 >>> 1;$

$X_3 := X_3 \oplus X_2 \oplus X_0 << 3;$

$X_1 := X_1 \oplus X_0 \oplus X_2;$

$X_2 := X_2 >>> 3;$

$X_0 := X_0 >>> 13;$

$B_{i-1} := X_0, X_1, X_2, X_3$

3.2. Decryption

The inverse process of encryption is given the name of decryption. In some algorithm, it is same as the encryption process while in other it is different from its encryption process. In the case of Serpent algorithm [1], it is different from the process of encryption. Here the inverse S-boxes are used in the decryption process. Also, the inverse linear transformation and reverse order of subkeys are used [1].

CHAPTER 4

MODIFIED SERPENT ALGORITHM

4.1. The Cipher

We modify the Serpent-1 [1] algorithm by using different S-boxes from [19]. Like Serpent algorithm, this is also a block cipher that encrypts a 128-bit block using a key of 256 bit. This algorithm consist of three basic functions, all different in the structure by Serpent-1 [1]. The steps are:

- Initial permutation (IP) (different from the previous algorithm)
- Round function (R) (less number of rounds and in each round of this algorithm each step inspect different number of bit from Serpent-1)
- Final Permutation (FP) (inverse of the initial permutation)

The whole process of the algorithm under consideration is shown in the Fig. 12 below:

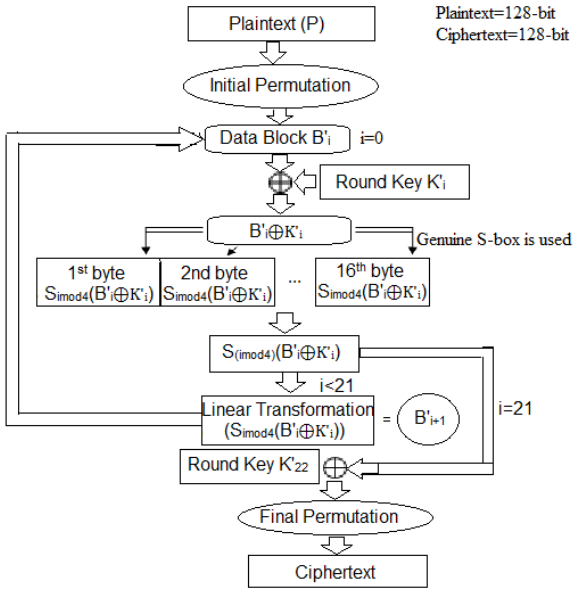


Fig. 12 Modified Serpent Algorithm flows chart

We take the initial permutation to be $[(i * 64)\ mod 127]$. Applying this permutation to 128 bits of a plaintext we get a block, B_0 which is the input to the 1st round.

Unlike the Serpent algorithm [1], we use four 4×4 S-boxes taken from [19], consisting of entries in bytes which represent polynomials to perform the round functions, i.e., 22 rounds.

We reduced the number of rounds from 32 to 22 in order to decrease the complexity and make the algorithm fast. Moreover, we also observe that this change affect the security level, but not to the level that the algorithm becomes unsecured. Inside these rounds we Mix B_i' with K_i' and then pass through the S-box S_{imod4} and apply the linear transformation to get B_{i+1}'; i = 0,1,2, …, 20. In the 22nd round, instead of applying linear transformation we mix it with an additional pre-key K_{32}'. Then the final permutation $[(i * 2)\ mod 127]$ (inverse of initial permutation) is applied to obtain the cipher text.

The whole process is described as:

B_0'=IP (P)

B_{i+1}'=LT(S-box $_{imod4}$ $(B_i' \oplus K_i')$);

i is from 0 to 20

B_{32}'=S_3 $(B_{21}' \oplus K_{21}') \oplus K_{22}'$

C= FP (B_{22}')

4.2. Elementary Transformations in proposed Cipher

Key mixing, linear transformation, and bit substitution are the three operations used in Serpent Algorithm [1] under study. As it seems that all the operations are like Serpent Algorithm operations but in contrast to Serpent, we mix a key that is made up of two 2w-bit (w=32-bit) vectors and then we XORed it with a data block initially obtained from the plain text by applying the initial permutation. Furthermore, the bit substitution, in this case, is a substitution from four different S-boxes consisting of invertible elements obtained from a multiplicative subgroup of the group of units of finite chain ring R_8 [19], whereas in the case of Serpent algorithm, these are obtained from DES. The linear transformation consists of XOR, rotation and shift.

Unlike Serpent-1, in the modified Serpent-1 algorithm we use 4 different 4 × 4 S-boxes in which each entry is a byte and each round use a single genuine S-box in order to encrypt or decrypt a 128-bit block. In Serpent-0, for high level of public confidence the S-boxes used were those adopted from DES and then in Serpent-1 the new and better S-boxes were generated which had stronger immunity to attacks but in our case we use a new, more secure and more complicated S-boxes which might have stronger immunity to attacks than the previous ones. For maximum avalanche effect, we use the exclusive-OR operation.

4.2.1. S-boxes under consideration

Nowadays good quality of ciphers are available depending on the non-linearity of substitution boxes and many other factors such as key schedule, number of rounds etc. (see [10, 12]). Primarily, Shah et al. [20] constructed S-boxes by using maximal cyclic subgroups G_3 and G_{15} of groups of units of the Galois rings GR(2^2,2) and GR(2^2,4), respectively. Obviously, injective and surjective S-boxes are reversible [10]. In sequel [19], Shah et al. constructed an S-box, which is a subgroup of multiplicative group of units of finite commutative chain ring $R_8 = \frac{\mathbb{Z}_2[u]}{<x^8>} = \mathbb{F}_2 + u\mathbb{F}_2 + u^2\mathbb{F}_2 + u^3\mathbb{F}_2 + u^4\mathbb{F}_2 + u^5\mathbb{F}_2 + u^6\mathbb{F}_2 + u^7\mathbb{F}_2$ consisting of 256 elements. Here, we follow the construction of S-box from [19] and obtained 4 different S-boxes. The multiplicative group of units of the ring R_8 is given as under (see [19]):

{10000000, 11000000, 10100000, 10010000, 10001000, 10000100, 10000010, 10000001, 11100000, 11010000, 11001000, 11000100, 11000010, 11000001, 10110000, 10101000, 10100100, 10100010, 10100001, 10011000, 10010100, 10010010, 10010001, 10001101, 10001010, 10001001, 10000110, 10000110, 10000011, 11110000, 11101000, 11100100, 11100010, 11100001, 11011000, 11010100, 11010010, 11010001, 11001100, 11001010, 11001001, 11000110, 11000101, 11000011, 10111000, 10110100, 10110010, 10110001, 10101100, 10101010, 10101001, 10100110, 10100101, 10100011, 10011000, 10011010, 10011001, 10010110, 10010101, 10010011, 10001110, 10001101, 10001011, 10000111, 11111000, 11110100, 11110010, 11110001, 11101100, 11101010, 11101001, 11100110, 11100101, 11100011, 11011100, 11011010, 11011001, 11010110, 11010101, 11010011, 11001110, 11001101, 11001011, 11000111, 11011100, 11011010, 10111001, 10110110, 10110101, 10110011, 10101110, 10101101, 10101011, 10100111, 10011110, 10011101, 10011011, 10010111, 10001111, 11111100, 11111010, 11110110, 11101110, 11011110, 10111110, 11111001, 11110101, 11101101, 11011101, 10111101, 11110011, 11101011, 11011011, 10111011, 11100111, 11010111, 10110111, 11001111, 10101111, 10011111, 11111110, 11111101, 11111011, 11110111, 11101111, 11011111, 10111111, 11111111}

Clearly this group contains a total of 128 elements. With the property that:

Number of elements of order $8 = 64$

Number of elements of order $4 = 48$

Number of elements of order $2 = 15$

Number of elements of order $1 = 01$

This group has a subgroup of cardinality 16 which is $< 1 + u^3 + u^6, 1 + u^2 + u^4 + u^5 + u^7 >$. Define $f: H_{G_8} \longrightarrow H_{G_8}$ by $f(a) = a^{-1}$ and $g: H_{G_8} \longrightarrow H_{G_8}$ by g(a)=a'a where $a' = 1 + u^4 + u^6$. Thus fog is the S-box S_0' on the finite commutative chain ring (R_8) which is as under:

10001010	10011001	10000010	10001000
10011011	10101111	10100101	10111010
10010010	10110001	10000000	10101101
10100111	10111000	10010000	10110011

Table 1: S-box on finite commutative chain ring R_8

The second S-box S_1' is obtained by rotating the rows of the 1^{st} S-box. The first row remain unchanged, 2^{nd} is rotated by one byte, 3^{rd} row is rotated by 2 bytes and the 4^{th} by 3 bytes. The 3^{rd} S-box is obtained by repeating the same process on the 2^{nd} S-box and the 4^{th} one is obtained by repeating the same process on the 3^{rd} one in order to maximize the avalanche effect and for high-security purpose. Where in each round we use a single genuine S-box.

Inverse S-box: The inverse S-box is given by the table 2:

10001010	10011011	10000010	10001000
10011001	10100101	10101111	10110011
10010000	10111000	10000000	10100111
10101101	10110001	10010010	10111010

Table 2: Inverse S-box on finite commutative chain ring R_8

4.2.2. Implementation of S-box

Divide the 128-bit into 16 bytes, numbering the entries of S-box from 0 to 15. Multiply the 1^{st} byte of 128-bit with 1^{st} entry of S-box, 2^{nd} byte with 2^{nd} entry of S-box and so on i.e. $X_iY_i;\ 0 \leq i \leq 15$, where X_i represents byte of the state and Y_i represents byte of the S-box. Note that the multiplication is carried out in the finite local ring $\mathbb{Z}_{2^8}$ and addition in binary field $\mathbb{F}_{2^8}$.

4.2.3. Key under consideration

If the supplied key is of length 256 bit we divide it into four 64 bit words named w'_{-4}, w'_{-3}, w'_{-2} and w'_{-1}. From these vectors we find the pre-keys w'_i; $i = 0,1,2,\dots,46$ by the following affine recurrence:

$$w'_i = (w_{i-4} \oplus w_{i-1} \oplus \phi, \phi \oplus i, i) <<< 5$$

Φ is the fractional part of the golden ratio $(\sqrt{5}+1)/2$ dentod by $0 \times 9e3779b9$ in hexadecimal and " $<<<$ " denotes left rotation of the bits. From this we can calculate the subkeys K'_i as:

$K'_0 = IP(S_1\ (W_0, W_1))$

$K'_1 = IP(S_0\ (W_2, W_3))$

$K'_2 = IP(S_3\ (W_4, W_5))$

$K'_3 = IP(S_2\ (W_6, W_7))$

...

$K'_{22} = IP(S_3\ (W_{45}, W_{46}))$

$K'_i = IP(S_{(1-i)\ mod4}(w_{2i}, w_{2i+1}));\ i = 0,1,2,\dots,32.$

Every key of length less than 256 bits can be mapped to a 256-bit key by writing one "1" at the extreme left followed by as many zeros as required to become a 256 bit key.

4.2.4. Linear Transformation:

We use the same linear transformation as used in Serpent-1.

4.3. Decryption

Since we are using the elements of an S-box obtained from a subgroup of the multiplicative group of commutative chain ring (R_8). Therefore each entry must have an inverse that forms the inverse S-box which we use in the reverse process. The inverse linear transformation and reverse order of subkeys are used in order to achieve our goal, i.e., to convert a ciphertext to plaintext.

4.4. Pseudo Code

```
//Main: start main function dataBlock

//input of 128 bits key
```

```
// input of a 256 bits, append 1 and make 256
S[4][16]

// initialization of S-boxes
makeWKeys (key);
for (i:0→21)
{//begin for

initialPermutaion(dataBlock);
dataBlock=dataBlock⊕getPrimeKey (i);
dataXORPrime(dataBlock, getPrimeKey(i), i);
if (i != 21)   // "!" represent not.

     linearTransformation (dataBlock);

}//end for
dataBlock=dataBlock⊕getPrimeKey(22);
finalPermutation(dataBlock);
// now dataBlock is contains encrypted form of    input data

//end main function

// Used Funcitons:
```

```
Function makeWKeys (dataBlock)

//WKeys represent Prekeys

{//begin function

     //making w-4 to w-1 : w0→w3
     for (i:0→31)

          W[i]=dataBlock[i];

     for (i:32→63)

          W[i-32]=dataBlock[i];

     for (i:64→95)

          W[i-64]=dataBlock[i];
```

```
     for (i:96→128)
          W[i-96]=dataBlock[i];
     //making w0 to w65:w4→w49
     for (i:4→49)
W[i]= (wi-4⊕wi-1⊕ ϕ, ϕ⊕i,i)<<<5;
}//end function
```

```
Function initialPermutation(&dataBlock)
{//begin function
     for (i:0→127) {//begin for
          if (i != 127)
               index = (i*64)%127;     // "%" is used for
               modulo
          else
               index =127;
          newBlock[i] = dataBlock [index];
     }//end for
     dataBlock= newBlock;
}//end function
```

```
Function getPrimeKey(index)
// prime-key represent subkey
{//begin function
     primeBlock= w(2*index), w(2*index+1);
     // the bits of the two are taken side by side.
     polyMult(primeBlock, S(1-index)mod4);
```

```
     initialPermutation(primeBlock);
     return primeBlock;
}//end function
```

```
Function dataXORprime(&data, pkey, index)    // pkey represent
subkey
{//begin function
     t;     //temporary byte
     d[16];           //byte size array
     k[16];           //byte size array
     for (i: 0→15 )
     {//begin for
          // now we split data and pkey into 16 bytes.
          d[i] = data[i*8] to data[i*8+8];
          k[i] = pkey[i*8] to pkey[i*8+8];
          d[i] = d[i] ⊕ k[i];                    //taking XOR
          of bytes and storing
          polyMult(t, s[index%4][i], d[i]);
          d[i] = t;
     }//end for
     data = d[0] to d[15];
     // combining all bytes
}//end function
```

```
Function linearTransformation(& input)
{//begin function
```

```
    // split input into 4 of 32 bit variables.
    X0 = input[0] to input[31]
    X1 = input[32] to input[63]
    X2 = input[64] to input[95]
    X3 = input[96] to input[127]

    //operations
    X0=rotleft(X0,13);
    X2=rotleft(X2,3);
    X1=X1⊕X0⊕X2;
    X3=X3⊕X2⊕(X0<<3);
    X1=rotleft(X1,1);
    X3=rotleft(X3,7);
    X0=X0⊕X1⊕X3;
    X2=X2⊕X3⊕(X1<<7);
    X0=rotleft(X0,5);
    X2=rotleft(X2,22);

    // rejoining X0, X1, X2, X3 to replace input data
    input= X0, X1, X2, X3         // their bits are taken
side by side.
}//end function
```

```
Function polyMult(&result, w, s)
{//begin function
```

```
     // all inputs are byte sized
     result=0; // Resetting all bits to 0
     for (i:0→8)
     {//begin outer for
          if (w[i]==0)    //if ith bit is zero
continue;     // then jump to next iteration directly
for(j: 0→ 8)
{//begin inner for
If (s[j]==0)
     Continue;
If (i+j>7)
     Continue;
flipBit (result, i+j);     //flip (i+j)th bit in result
// flipBit() isn't implemented, an already built programming
construct is used
// to do it.
          }//end inner for
     }//end outer for
}//end function
```

```
Function finalPermutation (&dataBlock)
{//begin function
     for (i:0→127)
     {//begin for
          if (i!=127)
               index=(i*2)%127;
          else
```

```
                index=127;
            newBlock[i]=dataBlock[index];
      }//end for
      dataBlock=newBlock;
}//end function
```

4.5. Complexity and Speed Analysis

Complexity of the proposed Algorithm is taken in comparison with Serpent Algorithm.

Conventions:

- The basic unit cost of an operation is taken as Constant (C).
- Bitwise Operations will have a cost of Constant(C). i.e. XOR operations, shifts and rotation.

Analysis of Serpent algorithm:

We find the complexities of the individual parts of the algorithm first.

i. Initial Permutation: Total 128 bits are copied. Cost: 128C

ii. Making W_{-8} to W_{-1}: Total 128 bits are copied. Cost: 128C

iii. Making W_0 to W_{131}: (5 XOR operations + 1 Shifts) = 6C, 132 times: 792C

iv. Making Round Key: Initial Permutation = 128C, Polynomial Multiplication of 128 bits
=128*128 = 16384C, 16384+128 = 16512C (extreme worst Case)

v. Data Block XOR: 32 XOR operations: 32C

vi. Linear Transformation: 16 unit operations: 16C

vii. Final Permutation: Total 128 bits are copied: 128C

Sequential Algorithm Flow:

Now we follow the algorithm with the above costs to calculate overall cost.

Making W_{-8} to W_{-1} 128C

Making W_0 to W_{131} 792C

for:i=0 to 31

{

Initial Permutation	128Cx32	=	4096C
Making Round Key	16512Cx32	=	528384C
Data Block XOR	32Cx32	=	1024C
if(i!=31)			
Linear Transformation	16Cx31	=	496C

}

Data Block XOR 32C

Final Permutation 128C

Total: 535080C

Analysis of Proposed Algorithm:

We find the complexities of the individual parts of the algorithm first.

i. Initial Permutation: Total 128 bits are copied. Cost: 128C

ii. Making W_{-4} to W_{-1}: Total 128 bits are copied. Cost: 128C

iii. Making W_0 to W_{49}: (3 XOR operations + 1 Shifts) = 4C, 49 times: 390C

iv. Making Round Key: Initial Permutation = 128C, Polynomial Multiplication of 128 bits $= 128 * 128 = 16384C$,
16384+128 = 16512C (extreme worst Case)

v. Data Block XOR: 16 XOR operations: 16C

vi. Linear Transformation: 16 unit operations: 16C

vii. Final Permutation: Total 128 bits are copied: 128C

Sequential Algorithm Flow:

Now we follow the algorithm with these costs to calculate overall cost.

Making W_{-4} to W_{-1} 128C

Making W_0 to W_{65} 390C

for i = 0 to 21

{

Initial Permutation: 128Cx22 = 2816C

Making Round Key: 16512Cx22 = 363264C

Data Block XOR: 16Cx22 = 352C

if(i!=21)

Linear Transformation: 16Cx21 = 336C

}

Data Block XOR: 16C

Final Permutation: 128C

Total: 367430C

Performance:

Relation between the two can be taken from:

|535080 – 367430| / 535080 * 100

= 31%

The new algorithm is 31% better.

CHAPTER 5

CONCLUSION

Crypto-Algorithms play an important role in secure communication. For the purpose of secure communication different algorithms like Rijndael algorithm, Serpent Algorithm, MARS Algorithm, TwoFish Algorithm and RC6 Algorithm have been designed. The preference is given to Rijndael Algorithm for AES because of its secureness, less complexity and fast speed.

We amended the Serpent Cipher and consequently it became 31% less complex and faster than the original algorithm. Dealing with a vector of 64 bits instead 32 bits at a time in the prekeys formation take a rule for speeding up the process of key formation and hence the whole algorithm (Slowness, the main drawback of Serpent Algorithm). Instead of using the replicated S-boxes obtained from DES as in the Serpent, we used genuine S-boxes that was obtained from the commutative chain ring $R_8 = \frac{\mathbb{Z}_2[x]}{<x^8>} = \mathbb{F}_2 + x\mathbb{F}_2 + x^2\mathbb{F}_2 + x^3\mathbb{F}_2 + x^4\mathbb{F}_2 + x^5\mathbb{F}_2 + x^6\mathbb{F}_2 + x^7\mathbb{F}_2$. The security of the proposed algorithm is hidden in the operation "addition" and "multiplication" which we performed in different systems (i.e. in R_8, addition coincide with the field "$\mathbb{F}_{2^8}$" and multiplication with the ring $\mathbb{Z}_{2^8}$)). In our modified algorithm when S-box is applied on a byte, the result falls in a set of 512 elements which means that the range of our S-box is the double time the range of Rijndael algorithm that gave strength to our algorithm. Also we believe that by performing half of the rounds (i.e. 11 rounds), the algorithm was secure as much as triple DES [13], the reason is that the best considered well known attack on AES and Serpent "the Extended spares linearization (XSL) attack" [6] is applicable to the elements of a Galois field, where there is an inverse for each element but in our case each element of the ring did not have inverse. This algorithm is free in all aspects from DES and all other algorithms especially in the case that how to apply the S-boxes. Also, in this case, we treated each element as a polynomial where addition and multiplication occur in the finite commutative chain ring R_8. It also gave a high level of confidence due to excludeness of any trapdoor.

REFRENCES

[1] Anderson, R., Biham, E., and Knudsen, L.: (1998). Serpent: A proposal for the advanced encryption standard. NIST AES Proposal, 174, 1-23.

[2] Barlow, R. H., and Evans, D. J.: (1982). Parallel algorithms for the iterative solution to linear systems. COMPUT J, 25(1), 56-60.

[3] Biham, E.: (1997, January). A fast new DES implementation in software. In Fast Software Encryption: (pp. 260-272). Springer Berlin Heidelberg.

[4] Biham, E., and Shamir, A.: (2012). Differential cryptanalysis of the data encryption standard. Springer Science and Business Media.

[5] Burwick, C., Coppersmith, D., D'Avignon, E., Gennaro, R., Halevi, S., Jutla, C., and Zunic, N.: (1998). MARS-a candidate cipher for AES. NIST AES Proposal, 268.

[6] Cid, C., and Leurent, G.: (2005). An analysis of the XSL algorithm. In Advances in Cryptology-ASIACRYPT 2005: (pp. 333-352). Springer Berlin Heidelberg.

[7] Contini, S., Rivest, R. L., Robshaw, M. J. B., and Yin, Y. L.: (1998). The Security of the RC6 TM Block Cipher. v1. 0. Available at http://www. rsa. com/rsalabs/aes/security.pdf.

[8] Daemen, J., and Rijmen, V.: (1999). AES proposal: Rijndael.

[9] Gehlot, P., Biradar, S. R., and Singh, B. P.: (2013). Implementation of Modified Twofish Algorithm using 128 and 192-bit keys on VHDL. IJCA, 70(13).

[10] Hussain, I., and Shah, T.: (2013). Literature survey on nonlinear components and chaotic nonlinear components of block ciphers. NONLINEAR DYN, 74(4), 869-904. DOI: DOI 10.1007/s11071-013-1011-8

[11] Hussain, I., Shah, T., and Mahmood, H.: (2010). A new algorithm to construct secure keys for AES. INT. J. CONTEMP. MATH. SCIENCES, 5(26), 1263-1270.

[12] Hussain, I., Shah, T., Mahmood, H., Gondal, M. A., and Bhatti, U. Y.: (2011). Some analysis of S-box based on residue of prime number. PROC PAK ACAD SCI, 48(2), 111-115.

[13] Kelsey, J., Schneier, B., and Wagner, D.: (1996, August). Key-schedule cryptanalysis of idea, g-des, gost, safer, and triple-des. In ADVANCES IN CRYPTOLOGY—CRYPTO'96: (pp. 237-251). Springer Berlin Heidelberg.

[14] Kahate, A.: (2013). Cryptography and network security. Tata McGraw-Hill Education.

[15] Matsui, M.: (1993, May). Linear cryptanalysis method for DES cipher. In Advances in Cryptology—EUROCRYPT'93: (pp. 386-397). Springer Berlin Heidelberg.

[16] Nyberg, K.: (1991, April). Perfect nonlinear S-boxes. In ADVANCES IN CRYPTOLOGY—EUROCRYPT'91: (pp. 378-386). Springer Berlin Heidelberg.

[17] Rivest, R. L., Robshaw, M. J., and Yin, Y. L.: (2000, April). RC6 as the AES. In AES Candidate Conference: (pp. 337-342).

[18] Schneier, B., Kelsey, J., Whiting, D., Wagner, D., Hall, C., and Ferguson, N.: (1999). The Twofish encryption algorithm: a 128-bit block cipher. John Wiley and Sons, Inc.

[19] Shah, T., Jahangir, S., and de Andrade, A. A.: Design of new 4×4 S-box from finite commutative chain rings. COMPUT APPL MATH, 1-15. DOI 10.1007/s40314-015-0265-9

[20] Shah, T., Qamar, A., and Hussain, I.: (2013). Substitution box on maximal cyclic subgroup of units of a Galois ring. Z NATURFORSCH A, 68(8-9), 567-572.

[21] Stanger, J., Lane, P. T., and Crothers, T.: (2006). CIW Security Professional Study Guide: Exam 1D0-470. John Wiley and Sons.

[22] Tran, M. T., Bui, D. K., and Duong, A. D.: (2008, December). Gray S-box for advanced encryption standard. In Computational Intelligence and Security (ICCIS), 2008. CIS'08. International Conference on Computational Intelligence and Securtiy: (Vol. 1, pp. 253-258). IEEE.

[23] Yulianto, A., Kom, S., and Prasetiyowati, M. I.: (2013, December). BoxLock: Mobile-based Serpent cryptographic algorithm and One-Time Password mechanism implementation for Dropbox files security. In Internet Technology and Secured Transactions (ICITST), 2013, 8th International Conference for Internet Technology and Secured Transaction (pp. 357-362). IEEE.