Ministry Of Higher Education Of Iraq
University Of Technology
Computer Science Department
Security Branch

New Approach For Serpent Algorithm

Supervisors:-
Assint. Prof. Dr. **Alaa Khadhim**

Group member:-
1-Mohammed A. Hussein
2-Ala'a Talib

2016


بسم الله الرحمن الرحيم

الاهداء

الى الذي بذل جهود السنين سخيا وصاغ فن الايام سلم العلم لأرتقي الى درب والدي.....الحياة

الى القلب الكبير الذي حل اسري الى الحنان الذي منحني الدفء الى من اخف الله والدتي.....الجنة تحت قدميها

اخوتي.....الى العيون البريئة التي تنظر الي بحب

اصدقائي.....الى اخوتي الذين لم تنجبهم امي

اساتذتي.....الي الشموع التي اضاءت الي طريق العلم

إليهم جميعا اهدي ثمرة جهدي المتواضع باذن الله تعالى وتوفيقه

Supervisor certification

I certify that the preparation of this project entitled

New approach for serpent algorithm

Was made under my supervision by

Asst. Prof. Dr. **Alaa Khadhim**

At the Computer Scenic Department at the University of Technology for the requirement of the B.S degree in computer scenic.

Signature:

Name: – *Asst. Prof. Dr.* **Alaa Khadhim**

List of content **page no.**

Chapter one: general introduction

Chapter two: principle of security and cryptography

Chapter three: new approach for serpent algorithm

Chapter four: conclusions

List of figures **page no.**

List of table **page no.**

CHAPTER ONE

General introduction

1.1 Introduction

Cryptography is the science of writing in secret code and is an ancient art; the first documented use of cryptography in writing dates back to circa 1900 B.C. when an Egyptian scribe used non-standard hieroglyphs in an inscription. Some experts argue that cryptography appeared spontaneously sometime after writing was invented, with application ranging from diplomatic missive to war-time battle plans. It is no surprise, then, that new forms of cryptography came soon after the widespread development of computer communications. In data and telecommunications, cryptography is necessary when communicating over any untrusted medium, which includes just about any network, particularly the internet within the context of any application-to-application communication, there are some specific security requirements, including authentication which is the process of proving one's identify. (The primary forms of host-to-host authentication on the internet today are name-based or address-based, both of which are notoriously weak.) Privacy/ confidentiality is ensuring that no one can read the message except the intended receiver. Integrity is assuring the receiver that the receiver message has not been altered in any way from the original. Non-repudiation is a mechanism to prove that the sender really sent this message. Cryptography, then, not only protects data from theft or alteration, but can also be used for user authentication. There are, in general, three types of cryptographic schemes typically used to accomplish these goals: secret key (or symmetric) cryptography, public key (or asymmetric) cryptography, and hash functions. In all cases, the initial unencrypted data is referred to as plaintext. It is encrypted into cipher text which will in turn (usually) be decrypted into usable plaintext.

Attacks can also interfere with the system's intended function, such as viruses, worms and Trojans. The other form of attack is when the system's resources are consumes uselessly, these can be

Caused by denial of service (DoS) attack. Other form of network intrusions also exist, such as land attacks, surf attacks, and teardrop attacks. These attacks are not as well-known as DoS attacks, but they are used in some form or another even if they aren't mentioned by name.

1.2 Aim of project

The aim of this project is new approach of serpent algorithm, design algorithm more complex and less time from the original algorithm.

1.3 Outline of content

This project is introduced through four chapters

Chapter one: introduction, aim of project, outline of content

Chapter two: introduction of cryptography, types of cryptography, symmetric cipher model include (DES algorithm, serpent algorithm, feal algorithm)

Chapter three: new approach of serpent algorithm

Chapter four: give the conclusions

CHAPTER TWO

Principle of security and cryptography

2.1 Introduction of Cryptography

Computers are now fond in every layer society, and information is being communicated and processed automatically on a large scale. Such as medical and financial files, automatic banking, video-phones, pay-tv, facsimiles, tele-shopping, and global computer networks. In all these cases there is a growing need for the protection of information to safeguard economic interests, to prevent fraud and to ensure privacy.

The term Cryptography is originally derived from the two Greek words “kryptos" and “graph", meaning hidden and writing. This is an accurate representation of the meaning of the word, as cryptography is the art of ensuring that messages (writing) are kept secure (hidden) from those recipients to whom the messages are not addressed .Cryptography is the science and study of methods of protecting data in computer and communication systems from unauthorized disclosure and modification. The Cryptographic systems are classified into two cryptosystems, private-key cryptosystem and public-key cryptosystem. Both are based on complex mathematical algorithms and are controlled by keys. The advances in cryptography have boosted its use in recent decades, opening up an amazing array of applications. It can be used to authenticate computer users, ensure the integrity and confidentiality of electronic communications, and keep sensitive information safely stored. From a secretive military technology, cryptography has emerged a key technology for all participants in the information society concerned about information security.

2.2 Symmetric Cipher Model

A symmetric encryption scheme has five ingredients (Figure 2.1):

Plaintext this is the original intelligible message or data that is fed into the algorithm as input.

Encryption algorithm the encryption algorithm performs various substitutions and transformations on the plaintext.

Secret key the secret key is also input to the encryption algorithm. The key is a value independent of the plaintext and of the algorithm. The algorithm will produce a different output depending on the specific key being used at the time. The exact substitutions and transformations performed by the algorithm Depend on the key.

Cipher text this is the scrambled message produced as output. It depends on the plaintext and the secret key. For a given message, two different keys will produce two different cipher texts. The cipher text is an apparently random stream of data and, as it stands, is unintelligible.

Decryption algorithm this is essentially the encryption algorithm run in Reverse. It takes the cipher text and the secret key and produces the original Plaintext. There are two requirements for secure use of conventional encryption:

1. We need a strong encryption algorithm. At a minimum, we would 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 figure out the key. This requirement is usually stated in a stronger form: The

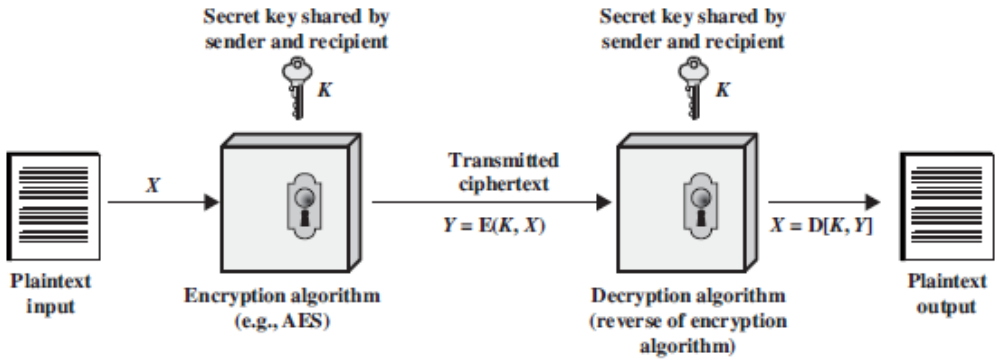

Figure (2.1): simplified model of symmetric encryption

Opponent should be unable to decrypt cipher text or discover the key even if he or she is

In possession of a number of cipher texts together with the plaintext that produced each cipher text

2. Sender and receiver must have obtained copies of the secret key in a secure Fashion and must keep the key secure. If someone can discover the key and knows the algorithm, all communication using this key is readable. We assume that it is impractical to decrypt a message on the basis of the cipher text *plus* knowledge of the encryption/decryption algorithm. In other words, we do not need to keep the algorithm secret; we need to keep only the key secret. This feature of symmetric encryption is what makes it feasible for widespread use. The fact that the algorithm need not be kept secret means that manufacturers can and have developed low-cost chip implementations of data encryption algorithms. These chips are widely available and incorporated into a number of products. With

The use of symmetric encryption, the principal security problem is maintaining the Secrecy of the key.Let us take a closer look at the essential elements of a symmetric encryption scheme, using Figure (2.2). A source produces a message in plaintext, $X = [X1, X2, \ldots, XM]$. . The M elements of X are letters in some finite alphabet. Traditionally, the alphabet usually consisted of the 26 capital letters. Nowadays, the binary alphabet {0, 1} is typically used. For encryption, a key of the form $K = [K1, K2 \ldots KJ]$ is generated. If the key is generated at the message source, then it must also be provided to the destination by means of some secure channel. Alternatively, a third party could generate the key and securely deliver it to both source and destination.

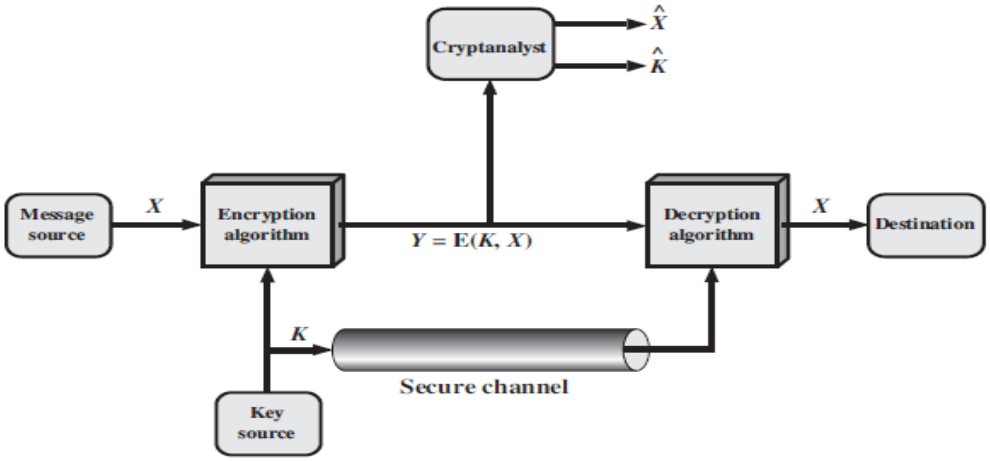

Figure (2.2): model of symmetric cryptosystem

With the message *X* and the encryption key *K* as input, the encryption algorithm forms the cipher text $Y = [Y1, Y2 \ldots YN]$ .We can write this as

$$Y = E(K, X)$$

This notation indicates that *Y* is produced by using encryption algorithm E as a function of the plaintext X, with the specific function determined by the value of the key *K*. The intended receiver, in possession of the key, is able to invert the transformation:

$$X = D(K, Y)$$

An opponent, observing *Y* but not having access to *K* or *X*, may attempt to recover *X* or *K* or both *X* and *K*. It is assumed that the opponent knows the encryption (E) and decryption (D) algorithms. If the opponent is interested in only this particular Message, then the focus of the effort is to recover *X by* generating a plaintext estimate. Often, however, the opponent is interested in being able to read future messages as Well, in which case an attempt is made to recover *K* by generating an estimate.

2.2.1THE DATA ENCRYPTION STANDARD

The most widely used encryption scheme is based on the Data Encryption Standard (DES) adopted in 1977 by the National Bureau of Standards, now the National Institute of Standards and Technology (NIST), as Federal Information Processing Standard 46 (FIPS PUB 46). The algorithm itself is referred to as the Data Encryption Algorithm (DEA).7 For DES, data are encrypted in 64-bit blocks using a 56-bit key. The algorithm transforms 64-bit input in a series of steps into a 64-bit output. The same steps, with the same key, are used to reverse the encryption. The DES enjoys widespread use. It has also been the subject of much controversy concerning how secure the DES .To appreciates the nature of the controversy; let us quickly review the history of the DES.

In the late 1960s, IBM set up a research project in computer cryptography led by Horst Feistel. The project concluded in 1971 with the development of an algorithm with the designation LUCIFER [FEIS73], which was sold to Lloyd's of London for use in a cash-dispensing system, also developed by IBM. LUCIFER is a Feistel block cipher that operates on blocks of 64 bits, using a key size of 128 bits. Because of the promising results produced by the LUCIFER project, IBM embarked on an effort to develop a marketable commercial encryption product that ideally could be implemented on a single chip. The effort was headed by Walter Tuchman and Carl Meyer, and it involved not only IBM researchers but also outside consultants and technical advice from the National Security Agency (NSA). The outcome of this effort was a refined version of LUCIFER that was more resistant to cryptanalysis but that had a reduced key size of 56 bits, in order to fit on a single chip. In 1973, the National Bureau of Standards (NBS) issued a request for proposals for a national cipher standard. IBM submitted the results of its Tuchman–Meyer project. This was by far the best algorithm proposed and was adopted in 1977 as the Data Encryption Standard.

Before its adoption as a standard, the proposed DES was subjected to intense criticism, which has not subsided to this day. Two areas drew the critics' fire. First, the key length in IBM's original LUCIFER algorithm was 128 bits, but that of the proposed system was only 56 bits, an enormous reduction in key size of 72 bits. Critics feared that this key length was too short to withstand brute-force attacks. The second area of concern was that the design criteria for the internal structure of DES, the S-boxes, were classified. Thus, users could not be sure that the internal structure of DES was free of any hidden weak points that would enable NSA to decipher messages without benefit of the key. Subsequent events, particularly the recent work on differential cryptanalysis, seem to indicate that DES has a very strong internal structure. Furthermore, according to IBM participants, the only changes that were made to the proposal were changes to the S-boxes, suggested by NSA, that removed vulnerabilities identified in the course of the evaluation process. Whatever the merits of the case, DES has flourished and is

Widely used, especially in financial applications. In 1994, NIST reaffirmed DES for federal use for another five years; NIST recommended the use of DES for applications other than the protection of classified information. In 1999, NIST issued a new version of its standard (FIPS PUB 46-3) that indicated that DES should be used only for legacy systems and that triple DES (which in essence involves repeating the DES algorithm three times on the plaintext using two or three different keys to produce the cipher text) be used. We study triple DES in Chapter 6. Because the underlying encryption and decryption algorithms are the same for DES and triple DES, it remains important to understand the DES cipher.

2.2.2 DES Encryption

The overall scheme for DES encryption is illustrated in Figure (2.3). As with any encryption scheme, there are two inputs to the encryption function: the plaintext to be encrypted and the key. In this case, the plaintext must be 64 bits in length and the key is 56 bits in length.

Looking at the left-hand side of the figure, we can see that the processing of the plaintext proceeds in three phases. First, the 64-bit plaintext passes through an initial permutation (IP) that rearranges the bits to produce the permuted input. This is followed by a phase consisting of sixteen rounds of the same function, which involves both permutation and substitution functions. The output of the last (sixteenth) round consists of 64 bits that are a function of the input plaintext and the key. The left and right halves of the output are swapped to produce the pre output. Finally, the pre output is passed through a permutation [IP-1] that is the inverse of the initial permutation function, to produce the 64-bit cipher text .With the exception of the initial and final permutations; DES has the exact structure of a Feistel cipher.

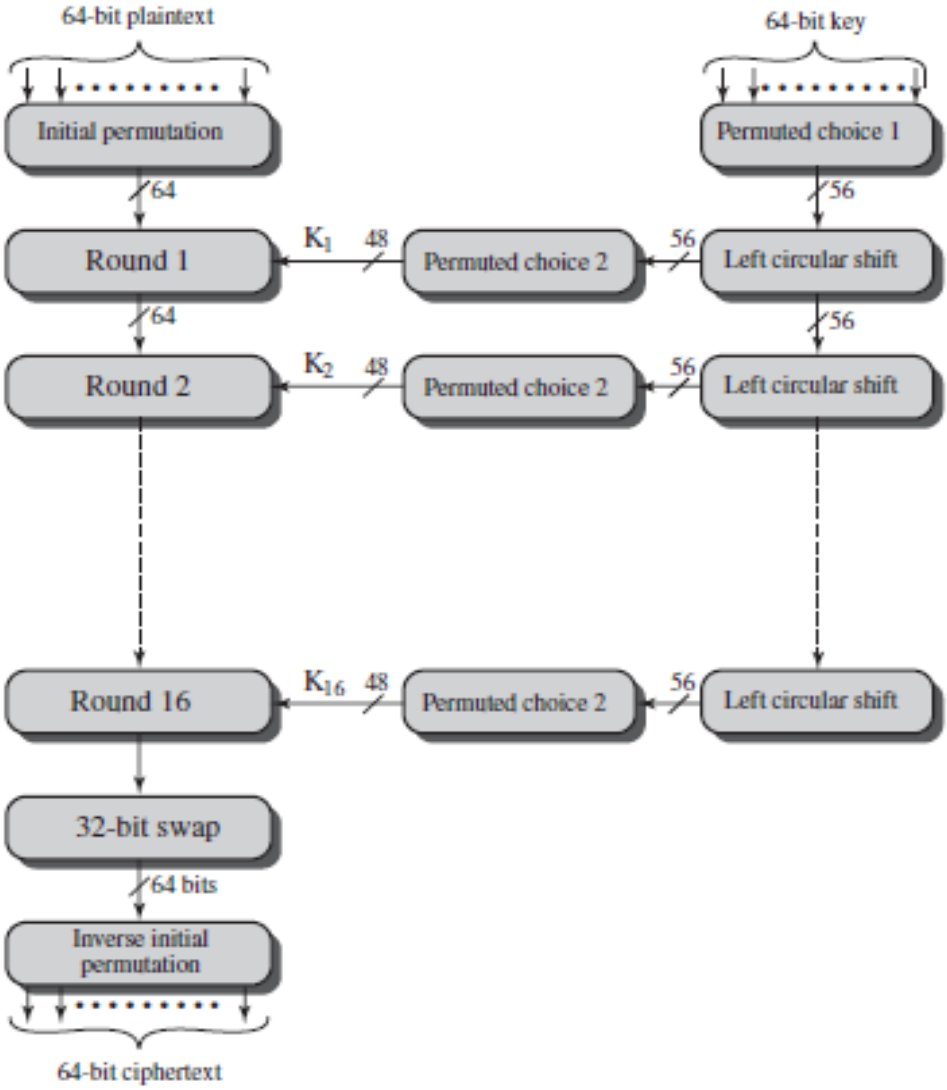

Figure (2.3): general depiction of des encryption algorithm

The right-hand portion of Figure (2.3) shows the way in which the 56-bit key is used. Initially, the key is passed through a permutation function. Then, for each of the sixteen rounds, a sub key (Ki) is produced by the combination of a left circular shift and a permutation. The permutation function is the same for each round, but a different sub key is produced because of the repeated shifts of the key bits.

INITIAL PERMUTATION the initial permutation and its inverse are defined by tables, as shown in Tables 2.1a and 2.1b, respectively. The tables are to be interpreted as follows. The input to a table consists of 64 bits numbered from 1 to 64.The 64 entries in the permutation table contain a permutation of the numbers from 1 to 64. Each

Table 2.1: ip and ip invers

(a) Initial Permutation (IP)

58	50	42	34	26	18	10	2
60	52	44	36	28	20	12	4
62	54	46	38	30	22	14	6
64	56	48	40	32	24	16	8
57	49	41	33	25	17	9	1
59	51	43	35	27	19	11	3
61	53	45	37	29	21	13	5
63	55	47	39	31	23	15	7

(b) Inverse Initial Permutation (IP^{-1})

40	8	48	16	56	24	64	32
39	7	47	15	55	23	63	31
38	6	46	14	54	22	62	30
37	5	45	13	53	21	61	29
36	4	44	12	52	20	60	28
35	3	43	11	51	19	59	27
34	2	42	10	50	18	58	26
33	1	41	9	49	17	57	25

(c) Expansion Permutation (E)

32	1	2	3	4	5
4	5	6	7	8	9
8	9	10	11	12	13
12	13	14	15	16	17
16	17	18	19	20	21
20	21	22	23	24	25
24	25	26	27	28	29
28	29	30	31	32	1

(d) Permutation Function (P)

16	7	20	21	29	12	28	17
1	15	23	26	5	18	31	10
2	8	24	14	32	27	3	9
19	13	30	6	22	11	4	25

Entry in the permutation table indicates the position of a numbered input bit in the output, which also consists of 64 bits. To see that these two permutation functions are indeed the inverse of each other, consider the following 64-bit input M

$$M_1M_2M_3M^4M_5M_6M_7M_8$$

$$M_9M_{10}M_{11}M_{12}M_{13}M_{14}M_{15}M_{16}$$

$$M_{17}M_{18}M_{19}M_{20}M_{21}M_{22}M_{23}M_{24}$$

$$M_{25}M_{26}M_{27}M_{28}M_{29}M_{30}M_{31}M_{32}$$

$$M_{33}M_{34}M_{35}M_{36}M_{37}M_{38}M_{39}M_{40}$$

$$M_{41}M_{42}M_{43}M_{44}M_{45}M_{46}M_{47}M_{48}$$

$$M_{49}M_{50}M_{51}M_{52}M_{53}M_{54}M_{55}M_{56}$$

$$M_{57}M_{58}M_{59}M_{60}M_{61}M_{62}M_{63}M_{64}$$

Where MI is a binary digit. Then the permutation X = (IP (M) is as follows:

$$M_{58}M_{50}M_{42}M_{34}M_{26}M_{18}M_{10}M_2$$

$$M_{60}M_{52}M_{44}M_{36}M_{28}M_{20}M_{12}M_4$$

$$M_{62}M_{54}M_{46}M_{38}M_{30}M_{22}M_{14}M_6$$

$$M_{64}M_{56}M_{48}M_{40}M_{32}M_{24}M_{16}M_8$$

$$M_{57}M_{49}M_{41}M_{33}M_{25}M_{17}M_9M_1$$

$$M_{59}M_{51}M_{43}M_{35}M_{27}M_{19}M_{11}M_3$$

$$M_{61}M_{53}M_{45}M_{37}M_{29}M_{21}M_{13}M_5$$

$$M_{63}M_{55}M_{47}M_{39}M_{31}M_{23}M_{15}M_7$$

If we then take the inverse permutation Y = IP-1(X) = IP-1(IP (M)), it can be seen that the original ordering of the bits is restored.

DETAILS OF SINGLE ROUND Figure (2.4) shows the internal structure of a single round.

Again, begin by focusing on the left-hand side of the diagram. The left and right halves of each 64-bit intermediate value are treated as separate 32-bit quantities, labeled L (left) and R (right). As in any classic Feistel cipher, the overall processing at each round can be summarized in the following formulas:

$$L_i = R_i\text{-}1$$

$$R_I = \text{Li- 1 \{F } (R_I\text{- 1}, K_i)$$

-40The round key K_i is 48 bits. The R input is 32 bits. This R input is first expanded to 48 bits by using a table that defines a permutation plus an expansion that involves duplication of 16 of the bits (Table 2.1c).The resulting 48 bits are XORed with. This 48-bit result passes through a substitution function that produces a 32-bit output, which is permuted as defined by Table 2.1d. The role of the S-boxes in the function F is illustrated in Figure **(2.5)**.The substitution consists of a set of eight S-boxes, each of which accepts 6 bits as input and produces 4 bits as output. These transformations are defined in Table 2.2, which is the role of the S-boxes in the function F is illustrated in Figure **(2.5)**.The substitution consists of a set of eight S-boxes, each of which accepts 6 bits as input and produces 4 bits as output. These transformations are defined in Table 2.2, which is

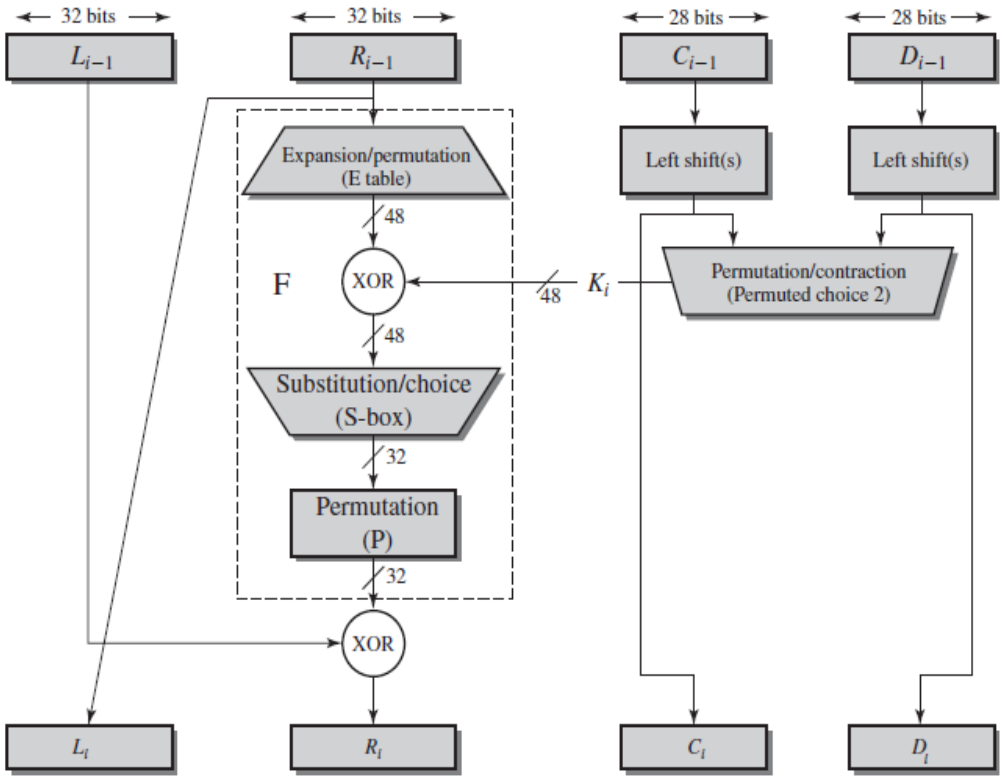

Figure (2.4): Single Round of DES Algorithm

Interpreted as follows: The first and last bits of the input to box *Si* form a 2-bit binary number to select one of four substitutions defined by the four rows in the table for. The middle four bits select one of the sixteen columns. The decimal value in the cell selected by the row and column is then converted to its 4-bit representation to produce the output. For example, in S1, for input 011001, the row is 01 (row 1) and the column is 1100 (column 12).The value in row 1, column 12 is 9, so the output is 1001. Each row of an S-box defines a general reversible substitution. Figure 3.2 may be useful in understanding the mapping. The figure shows the substitution for row 0 of box. The operation of the S-boxes is worth further comment. Ignore for the moment the contribution of the key ( ). If you examine the expansion table, you see that the

32 bits of input are split into groups of 4 bits and then become groups of 6 bits by taking the outer bits from the two adjacent groups. For example, if part of the input word is

... Efgh ijkl mnop...

This becomes

... Defghi hijklm lmnopq...

Ki

S1

Si

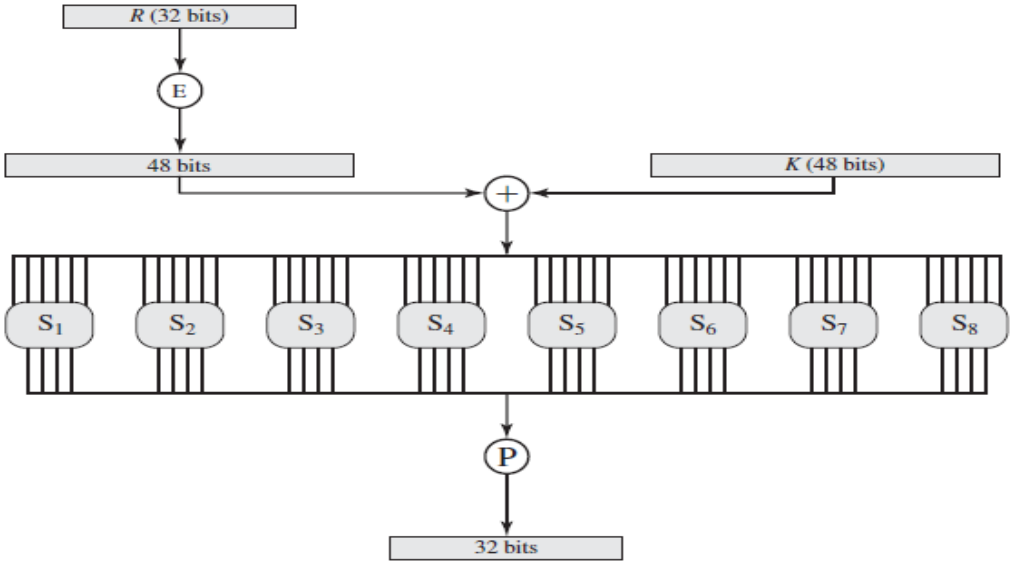

Figure (2.5): Calculation of F(R, K)

The outer two bits of each group select one of four possible substitutions (one row of an S-box). Then a 4-bit output value is substituted for the particular 4-bit input (the middle four input bits). The 32-bit output from the eight S-boxes is then permuted, so that on the next round, the output from each S-box immediately affects as many others as possible.

2.2.3 Key generation

Returning to Figures (2.3) and (2.4), we see that a 64-bit key is used as input to the algorithm. The bits of the key are numbered from 1 through 64; every eighth bit is ignored, as indicated by the lack of shading in Table 3.4a.The key is first subjected to a permutation governed by a table labeled Permuted Choice One (Table 2.3b).The resulting 56-bit key is then treated as two 28-bit quantities, labeled and . At each round, and are separately subjected to a circular left shift or (rotation) of 1 or 2 bits, as governed by Table 2.3d.These shifted values serve as input to the next round. They also serve as input to the part labeled Permuted Choice Two (Table 2.3c), which produces a 48-bit output that serves as input to the function.

2.2.4 DES Decryption

As with any Feistel cipher, decryption uses the same algorithm as encryption, except that the application of the sub keys is reversed.

Table 2.2: definition of des s-box

Table 3.3 Definition of DES S-Boxes

S_1	14	4	13	1	2	15	11	8	3	10	6	12	5	9	0	7
	0	15	7	4	14	2	13	1	10	6	12	11	9	5	3	8
	4	1	14	8	13	6	2	11	15	12	9	7	3	10	5	0
	15	12	8	2	4	9	1	7	5	11	3	14	10	0	6	13
S_2	15	1	8	14	6	11	3	4	9	7	2	13	12	0	5	10
	3	13	4	7	15	2	8	14	12	0	1	10	6	9	11	5
	0	14	7	11	10	4	13	1	5	8	12	6	9	3	2	15
	13	8	10	1	3	15	4	2	11	6	7	12	0	5	14	9
S_3	10	0	9	14	6	3	15	5	1	13	12	7	11	4	2	8
	13	7	0	9	3	4	6	10	2	8	5	14	12	11	15	1
	13	6	4	9	8	15	3	0	11	1	2	12	5	10	14	7
	1	10	13	0	6	9	8	7	4	15	14	3	11	5	2	12
S_4	7	13	14	3	0	6	9	10	1	2	8	5	11	12	4	15
	13	8	11	5	6	15	0	3	4	7	2	12	1	10	14	9
	10	6	9	0	12	11	7	13	15	1	3	14	5	2	8	4
	3	15	0	6	10	1	13	8	9	4	5	11	12	7	2	14
S_5	2	12	4	1	7	10	11	6	8	5	3	15	13	0	14	9
	14	11	2	12	4	7	13	1	5	0	15	10	3	9	8	6
	4	2	1	11	10	13	7	8	15	9	12	5	6	3	0	14
	11	8	12	7	1	14	2	13	6	15	0	9	10	4	5	3
S_6	12	1	10	15	9	2	6	8	0	13	3	4	14	7	5	11
	10	15	4	2	7	12	9	5	6	1	13	14	0	11	3	8
	9	14	15	5	2	8	12	3	7	0	4	10	1	13	11	6
	4	3	2	12	9	5	15	10	11	14	1	7	6	0	8	13
S_7	4	11	2	14	15	0	8	13	3	12	9	7	5	10	6	1
	13	0	11	7	4	9	1	10	14	3	5	12	2	15	8	6
	1	4	11	13	12	3	7	14	10	15	6	8	0	5	9	2
	6	11	13	8	1	4	10	7	9	5	0	15	14	2	3	12
S_8	13	2	8	4	6	15	11	1	10	9	3	14	5	0	12	7
	1	15	13	8	10	3	7	4	12	5	6	11	0	14	9	2
	7	11	4	1	9	12	14	2	0	6	10	13	15	3	5	8
	2	1	14	7	4	10	8	13	15	12	9	0	3	5	6	11

Table 2.3: des key schedule calculation

(a) Input Key

1	2	3	4	5	6	7	8
9	10	11	12	13	14	15	16
17	18	19	20	21	22	23	24
25	26	27	28	29	30	31	32
33	34	35	36	37	38	39	40
41	42	43	44	45	46	47	48
49	50	51	52	53	54	55	56
57	58	59	60	61	62	63	64

(b) Permuted Choice One (PC-1)

57	49	41	33	25	17	9
1	58	50	42	34	26	18
10	2	59	51	43	35	27
19	11	3	60	52	44	36
63	55	47	39	31	23	15
7	62	54	46	38	30	22
14	6	61	53	45	37	29
21	13	5	28	20	12	4

(c) Permuted Choice Two (PC-2)

14	17	11	24	1	5	3	28
15	6	21	10	23	19	12	4
26	8	16	7	27	20	13	2
41	52	31	37	47	55	30	40
51	45	33	48	44	49	39	56
34	53	46	42	50	36	29	32

(d) Schedule of Left Shifts

Round Number	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
Bits Rotated	1	1	2	2	2	2	2	2	1	2	2	2	2	2	2	1

2.2.3 Serpent Block Cipher Algorithm [1, 2, 3]

Serpent is a 32-round SP-network operating on four 32-bit words, thus giving a block size of 128 bits. All values used in the cipher are represented as bit streams. The indices of the bits are counted from 0 to bit 31 in one 32-bit word, 0 to bit 127 in 128-bit blocks, 0 to bit 255 in 256-bit keys, and so on. For internal computation, all values are represented in little-endian, where the first word (word 0) is the least significant word, and the last word is the most significant, and where bit 0 is the least significant bit of word 0. Externally, we write each block as a plain 128-bit hex number.

Serpent encrypts a 128-bit plaintext *P* to a 128-bit cipher text *C* in 32 rounds under the control of 33 128-bit sub keys *K*0… *K32*. The user key length is variable, but for the purposes of this submission we fix it at 128, 192 or 256 bits, short keys with less than 256 bits are mapped to full-length keys of 256 bits by appending one “1" bit to the MSB end, followed by as many “0" bits as required to make up 256 bits. This mapping is designed to map every short key to a full-length key, with no two short keys being equivalent. The serpent algorithm cipher consists of:

-**Initial permutation** IP.

-**Rounds**: 32 rounds, each consisting of a key mixing operation, a pass through S- boxes, and (in all but the last round) a linear transformation. In the last round, this linear transformation is replaced by an additional key mixing operation,

-**Final permutation** FP.

Can explain the algorithm of serpent in the following figure:

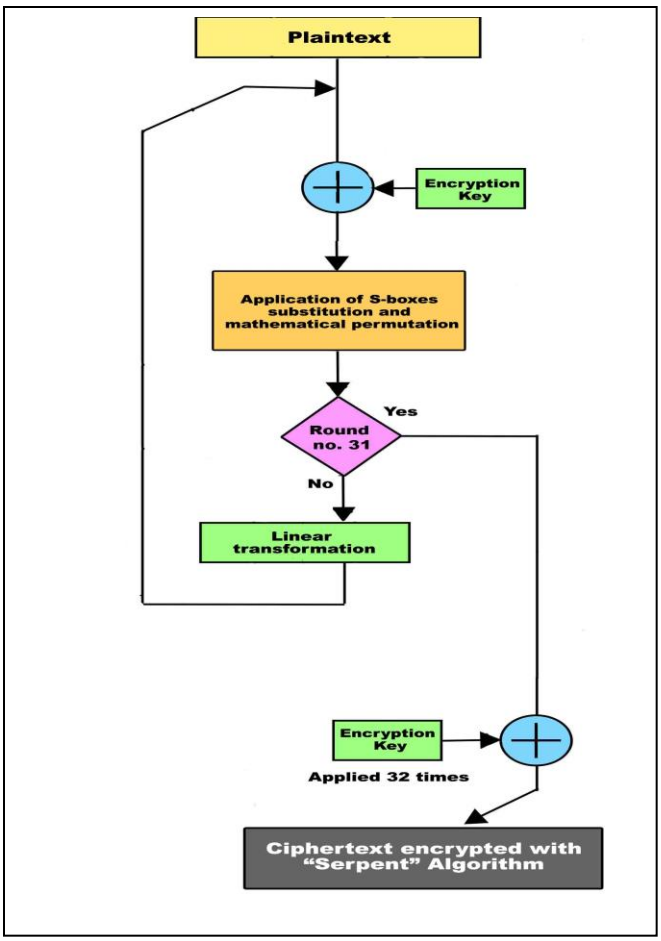

Figure (2.6): Serpent block Algorithm

The initial and final permutations do not have any cryptographic significance. They are used to simplify an optimized implementation of the cipher, both these two permutations and the linear transformation are to add more complexity .can explain the initial permutation and final

Table (2.4): ip and ip invers for serpent

0	32	64	96	1	33	65	97	2	34	66	98	3	35	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

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	35	39	43	47	51	55	59	63
67	71	75	79	83	87	91	95	99	103	107	111	115	119	123	127

The initial permutation *IP* is applied to the plaintext *P* giving ^*B*0, which is the input to the first round. The rounds are numbered from 0 to 31, where the first round is round 0 and the last is round 31. The output of the first round (round 0) is ^*B*1, the output of the second round (round 1) is ^*B*2, the output of round *i* is ^*Bi*+1, and so on, until the output of the last round (in which the linear transformation is replaced by an additional key mixing) is denoted by ^*B*32. The final permutation *FP* is now applied to give the cipher text *C*.

Each round function R_I (i € {0… 31*}* uses only a single replicated S-box. For example, *R*0 uses *S*0, 32 copies of which are applied in parallel. Thus the first copy of *S*0 takes bits 0, 1, 2 and 3 of (^*B*0 *XOR* ^*K*0) as its input and returns as output the first four bits of an intermediate vector, the next copy of *S*0 inputs bits 4-7 of(^*B*0 *XOR* ^*K*0) and returns the next four bits of the intermediate vector, and so on. The intermediate vector is then transformed using the linear transformation, giving ^*B1*. Similarly, *R*1 uses 32 copies of *S*1 in parallel on ^*B*1 *XOR* ^*K*1 and transforms their output using the linear transformation, giving ^*B*2.

The set of eight S-boxes is used four times. Thus after using *S*7 in round 7, we use *S*0 again in round 8, then *S*1 in round 9, and so on. The last round *R*31 is slightly different from the others, apply *S*7 on ^*B*31 *XOR* ^*K*31, and XOR the result with ^*K*32 rather than applying the linear transformation. The result ^*B*32 is then permuted by *FP*, giving the cipher text.

Thus the 32 rounds use 8 different S-boxes each of which maps four input bits to four output bits. Each S-box is used in precisely four rounds, and in each of these it is used 32 times in parallel. Can explain the in the following table:

Table (2.5): S-Box and S-Box inverse

The S-boxes used in Serpent for encryption from S_0 to S_7 are given below:

S0:	3	8	15	1	10	6	5	11	14	13	4	2	7	0	9	12
S1:	15	12	2	7	9	0	5	10	1	11	14	8	6	13	3	4
S2:	8	6	7	9	3	12	10	15	13	1	14	4	0	11	5	2
S3:	0	15	11	8	12	9	6	3	13	1	2	4	10	7	5	14
S4:	1	15	8	3	12	0	11	6	2	5	4	10	9	14	7	13
S5:	15	5	2	11	4	10	9	12	0	3	14	8	13	6	7	1
S6:	7	2	12	5	8	4	6	11	14	9	1	15	13	3	10	0
S7:	1	13	15	0	14	8	2	11	7	4	12	10	9	3	5	6

The inverse S-boxes used in Serpent for decryption from InvS0 to InvS7 are given below:

InvS0:	13	3	11	0	10	6	5	12	1	14	4	7	15	9	8	2
InvS1:	5	8	2	14	15	6	12	3	11	4	7	9	1	13	10	0
InvS2:	12	9	15	4	11	14	1	2	0	3	6	13	5	8	10	7
InvS3:	0	9	10	7	11	14	6	13	3	5	12	2	4	8	15	1
InvS4:	5	0	8	3	10	9	7	14	2	12	11	6	4	15	13	1
InvS5:	8	15	2	9	4	1	13	14	11	6	5	3	7	12	10	0
InvS6:	15	10	1	13	5	3	6	0	4	9	14	7	2	12	8	11
InvS7:	3	0	6	13	9	14	15	8	5	12	11	7	10	1	4	2

As with DES, the final permutation is the inverse of the initial permutation. Thus the cipher may be formally described by the following equations:

$$\mathbf{B}_0 = \text{IP (P)}$$

$$\mathbf{\hat{}B_{i+1}} = \text{RI } (\hat{}B_i)$$

$$\mathbf{C} = \text{FP } (\hat{}B_{32})$$

Where

$$RI(X) = L\ (\hat{}S_i\ (X\ XOR\ \hat{}K_i)) \quad i = \mathbf{0\ldots\ 30}$$

$$R_I(X) = \hat{}S_i(X\ XOR\ \hat{}K_i)\ \hat{}K_{32} \qquad i = \mathbf{31}$$

Where $\hat{}\ S_i$ is the application of the S-box $S_{i \bmod 8}$ 32 times in parallel, and L is the linear transformation.

Linear Transformation the 32 bits in each of the output words are linearly mixed, by the equations:

$$
\begin{aligned}
X_0, X_1, X_2, X_3 &:= \mathcal{S}_i(B_i \oplus K_i) \\
X_0 &:= X_0 <<< 13 \\
X_2 &:= X_2 <<< 3 \\
X_1 &:= X_1 \oplus X_0 \oplus X_2 \\
X_3 &:= X_3 \oplus X_2 \oplus (X_0 << 3) \\
X_1 &:= X_1 <<< 1 \\
X_3 &:= X_3 <<< 7 \\
X_0 &:= X_0 \oplus X_1 \oplus X_3 \\
X_2 &:= X_2 \oplus X_3 \oplus (X_1 << 7) \\
X_0 &:= X_0 <<< 5 \\
X_2 &:= X_2 <<< 22 \\
B_{i+1} &:= X_0, X_1, X_2, X_3
\end{aligned}
$$

Where <<< denotes rotation, and << denotes shift. In the last round, this linear transformation is replaced by an additional key mixing: $B_{32} = S7\ (B31\ XOR\ K31)\ XOR\ K_{32}$. Note that at each stage IP $(B_i) = \wedge B_i$, and $IP\ (K_i) = \wedge K_i$. Can explain the linear transformation in the following figure:

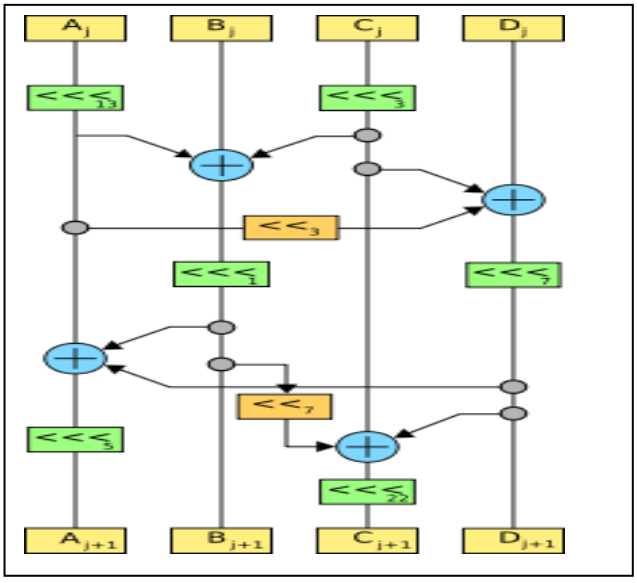

Figure (2.7): Linear Transformation Function

2.2.3.1 Key Schedule

As with the description of the cipher, can describe the key schedule in either standard or bit slice mode. Give the substantive description for the latter case.

Our cipher requires 132 32-bit words of key material. We first pad the user supplied key to 256 bits, if necessary. We then expand it to 33 128-bit sub keys $K0 \ldots K32$, in the following way. We write the key K as eight 32-bit words $w_{-8} \ldots w_{-1}$ and expand these to an intermediate key (which we call *pre key*) $w_0 \ldots w_{131}$ by the following affine recurrence:

$$W_I = (w_{i-8}\, XOR\, w_{i-5}\, XOR\, w_{i-3}\, XOR\, w_{i-1}\, XOR\, \phi\, XOR\, i) <<< 11$$

Where ϕ is the fractional part of the golden ratio $(\sqrt{5} + 1) = 2$ or 0x9e3779b9 in hexadecimal. The underlying polynomial $x^8 + x^7 + x^5 + x^3 + 1$ is primitive, which together with the addition of the round index is chosen to ensure an even distribution of key bits throughout the rounds, and to eliminate weak keys and related keys.

The round keys are now calculated from the rekeys using the S-boxes, again in bit slice mode .can used the S-boxes to transform the rekeys W_I into words K_i of round key in the following way:

{k0, k1, K2, k3} = S3 (w0, w1, w2, w3)

{k4, k5, k6, k7} = S2 (w4, w5, w6, w7)

{k8, k9, k10, k11}= S1 (w8, w9, w10, w11)

{k12, k13, k14, k15} = S0 (w12, w13, w14, w15)

{k16, k17, k18, k19} = S7 (w16, w17, w18, w19)

: : :

{k124, k125, k126, k127} = S4 (w124, w125, w126, w127)

{k128, k129, k130, k131} = S3 (w128, w129, w130, w131)

Then renumber the 32-bit values k_j as 128-bit sub keys K_i (for i € {0 . . . r}) as follows:

$$K_i = \{k_{4i}, k_{4i+1}, k_{4i+2}, k_{4i+3}\}$$

2.2.3.2 Decryption in Serpent Algorithm

Decryption is different from encryption in that the inverse of the S-boxes must be used in the reverse order, as well as the inverse linear transformation and reverse order of the sub keys.

2.2.4 FEAL

FEAL was designed by Akihiro Shimizu and Shoji Miyaguchi from NTT Japan [1435]. It uses a 64-bit block and a 64-bit key. The idea was to make a DES-like algorithm with a stronger round function. Needing fewer rounds, the algorithm would run faster. Unfortunately, reality fell far short of the design goals.

2.2.4.1 Description of FEAL

Figure (2.8) is a block diagram of one round of FEAL. The encryption process starts with a 64-bit block of plaintext. First, the data block is XORed with 64 key bits. The data block is then split into a left half and a right half. The left half is XORed with the right half to form a new right half. The left and new right halves go through *n* rounds (four, initially). In each round the right half is combined with 16 bits of key material (using function f) and XORed with the left half to form the new right half. The original right half (before the round) forms the new left half. After *n* rounds (remember not to switch the left and right halves after the *n*th round) the left half is again XORed with the right half to form a new right half, and then the left and right halves are concatenated Together to form a 64-bit whole. The data block is XORed with another 64 bits of key material, and the algorithm terminates.

Function f takes the 32 bits of data and 16 bits of key material and mixes them together. First the data block is broken up into 8-bit chunks, then the chunks are XORed and substituted with each other. Figure (2.9) is a block diagram of function f. The two functions *S*0 and *S*1, are defined as:

*S*0 (*a, b*) = rotate left two bits ((*a* + *b*) mod 256)

*S*1 (*a, b*) = rotate left two bits ((*a* + *b* + 1) mod 256)

The same algorithm can be used for decryption. The only difference is: When decrypting, the key material must be used in the reverse order. Figure (2.11) is a block diagram of the key-generating function. First the 64-bit key is divided into two halves. The halves are XORed and operated on by function fk, as indicated in the diagram. Figure (2.10) is a block diagram of function fk. The two 32-bit inputs are broken up into 8-bit blocks and combined and substituted as shown. *S*0 and *S*1 are defined as just shown. The 16-bit key blocks are then used in the encryption/decryption algorithm.

On a 10 megahertz 80286 microprocessor, an assembly-language implementation of FEAL-32 can encrypt data at a speed of 220 kilobits per second. FEAL-64 can encrypt data at a speed of 120 kilobits per second [1104].

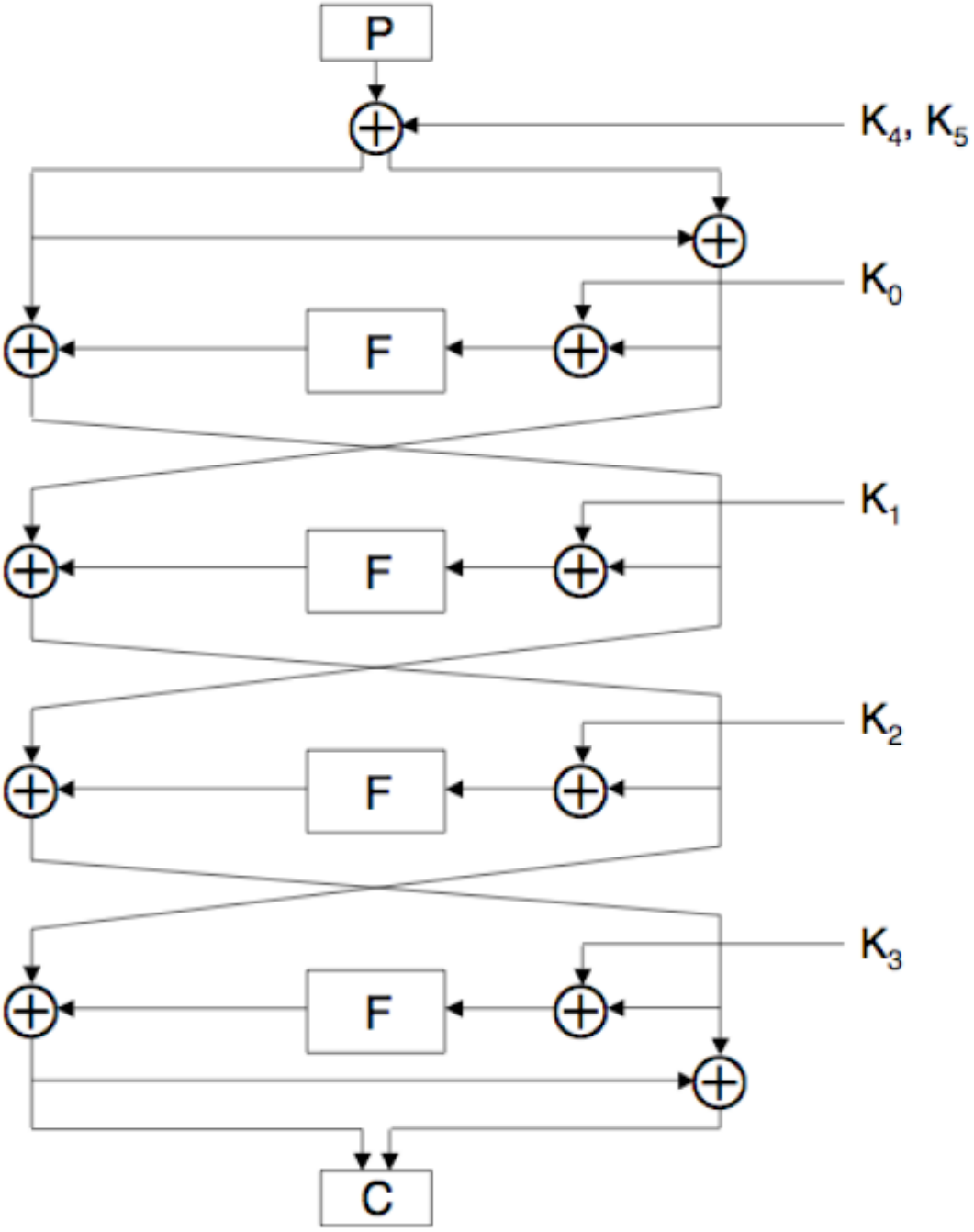

Figure (2.8) is a block diagram of one round of FEAL

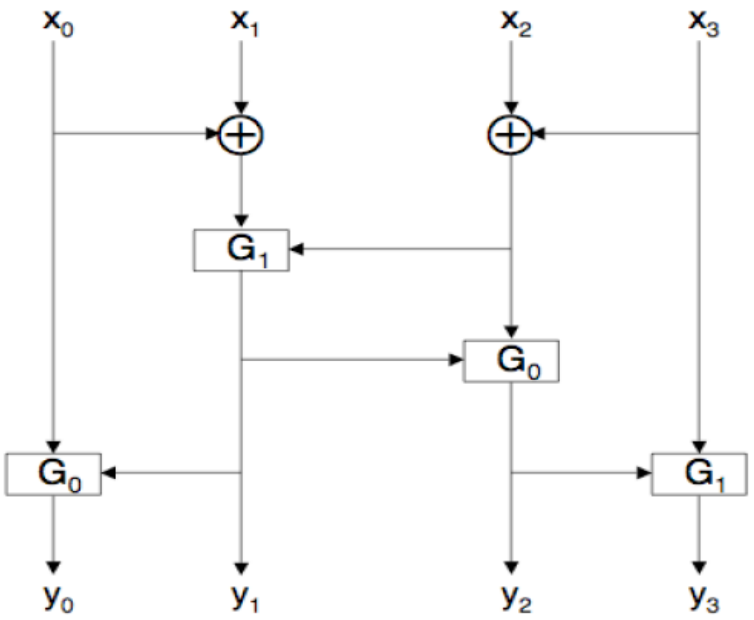

Figure (2.9) is a block diagram of function f

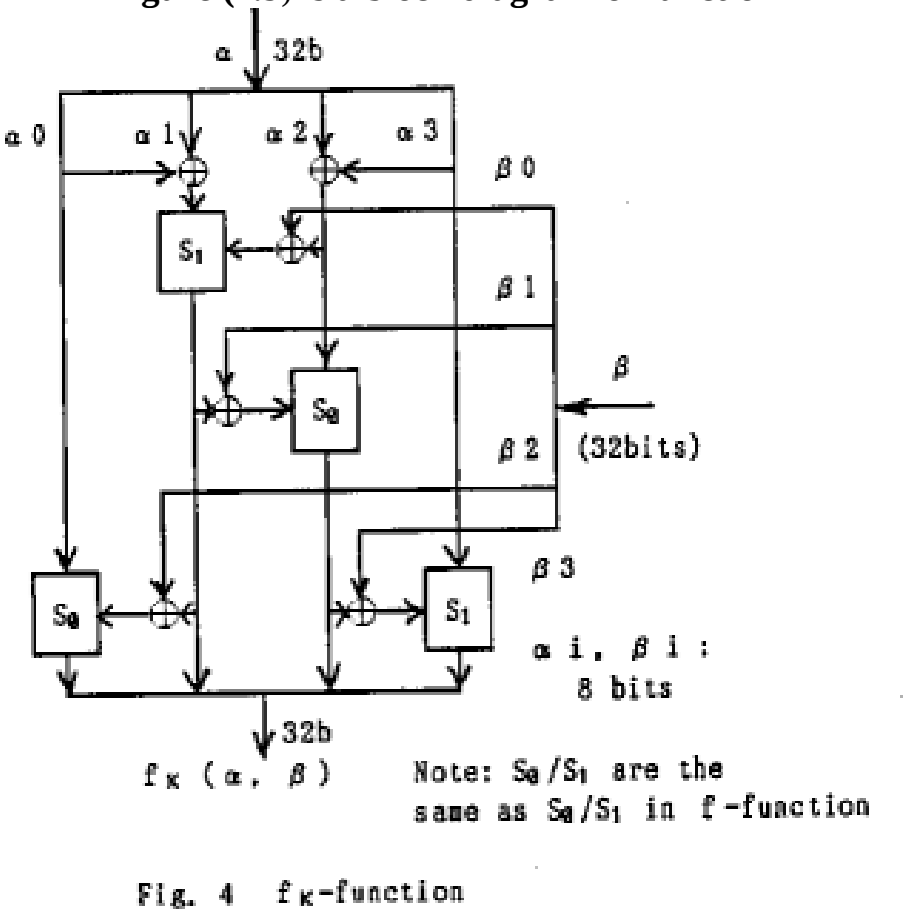

Fig. 4 f_K-function

Figure (2.10) is a block diagram of the key-generating function

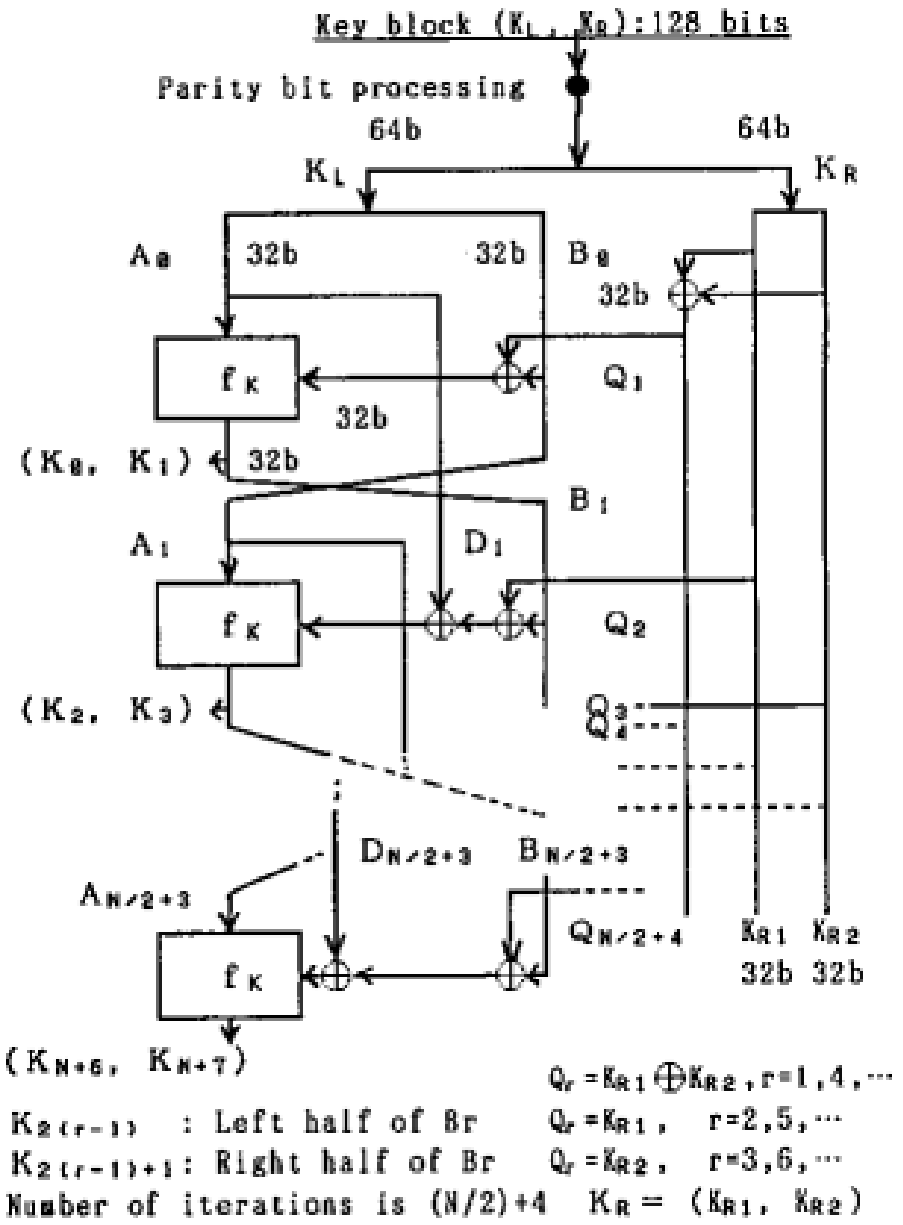

Figure (2.11) is a block diagram of function fk

2.2.4.2Cryptanalysis of FEAL

FEAL-4, FEAL with four rounds, was successfully cryptanalyzed with a chosen-plaintext attack in [201] and later demolished [1132]. This later attack, by Sean Murphy, was the first published differential-cryptanalysis attack and required only 20 chosen plaintexts. The designers retaliated with 8-round FEAL [1436, 1437, 1108] which Biham and Shamir cryptanalyzed at the

SECURICOM ’89 conference [1427]. Another chosen-plaintext attack, using only 10,000 blocks, Against FEAL-8 [610] forced the designers to throw up their hands and define FEAL-*N* [1102, 1104], with a variable number of rounds (greater than 8, of course).

Biham and Shamir used differential cryptanalysis against FEAL-*N*; they could break it more quickly than by brute force (with fewer than 264 chosen plaintext encryptions) for *N* less than 32 [169]. FEAL-16 required 228 chosen plaintexts or 246.5 known plaintexts to break. FEAL-8 required 2000 chosen plaintexts or 237.5 known plaintexts to break. FEAL-4 could be broken with just eight carefully selected chosen plaintexts.

The FEAL designers also defined FEAL-*NX*, a modification of FEAL, that accepts 128-bit keys [1103, 1104]. Biham and Shamir showed that FEAL-*NX* with a 128-bit key is just as easy to break as FEAL-*N* with a 64-bit key, for any value of *N* [169]. Recently FEAL-*N*(X) S has been proposed, which strengthens FEAL with a dynamic swapping function [1525].

There’s more. Another attack against FEAL-4, requiring only 1000 known plaintexts, and against FEAºL-8, requiring only 20,000 known plaintexts, was

Published in [1520]. Other attacks are in [1549, 1550]. The best attack is by Mitsuru Matsui and Atshuiro Yamagishi [1020]. This is the first use of linear cryptanalysis, and can break FEAL-4 with 5 known plaintexts, FEAL-6 with 100 known plaintexts and FEAL-8 with 215 known plaintexts. Further refinements are in [64]. Differential-linear cryptanalysis can break FEAL-8 with only 12 chosen plaintexts [62]. Whenever someone discovers a new

Cryptanalytic attack, he always seems to try it out on FEAL first.

Patents

FEAL is patented in the United States [1438] and has patents pending in England, France, and Germany. Anyone wishing to license the algorithm should contact the Intellectual Property Department, NTT, 1-6 Uchisaiwai-cho, 1-chome, Chiyoda-ku, 100 Japan.

2.3 Public-Key Cryptosystems

Asymmetric algorithms rely on one key for encryption and a different but related key for decryption. These algorithms have the following important characteristic:

It is computationally infeasible to determine the decryption key given only knowledge of the cryptographic algorithm and the encryption key.

In addition, some algorithms, such as RSA, also exhibit the following characteristic:

Either of the two related keys can be used for encryption, with the other used for decryption.

A public-key encryption scheme has six ingredients (in the following Figure (2.8); compare with anther Figure*)*

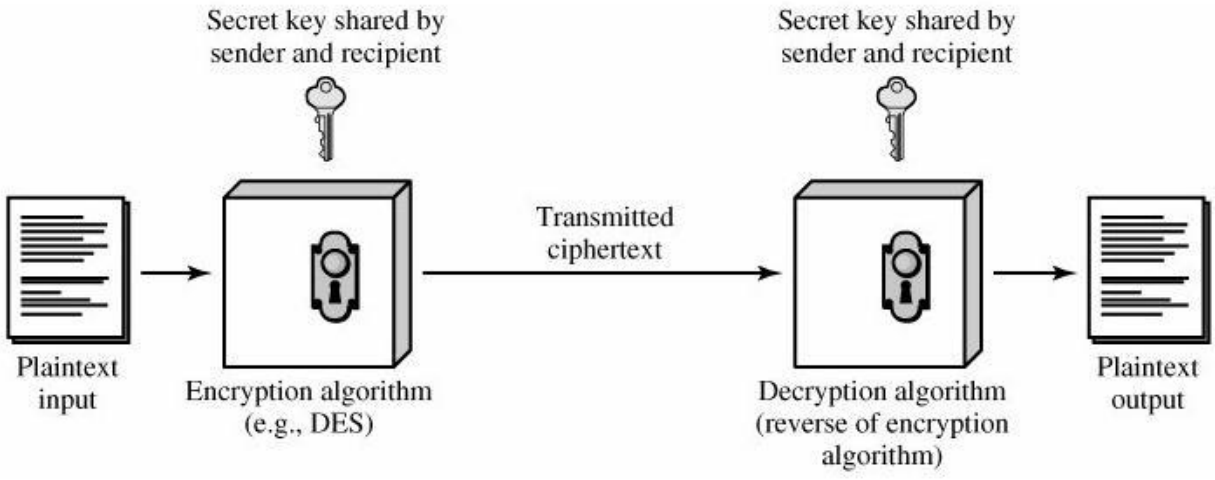

Figure (2.12): **Simplified Model of Conventional Encryption**

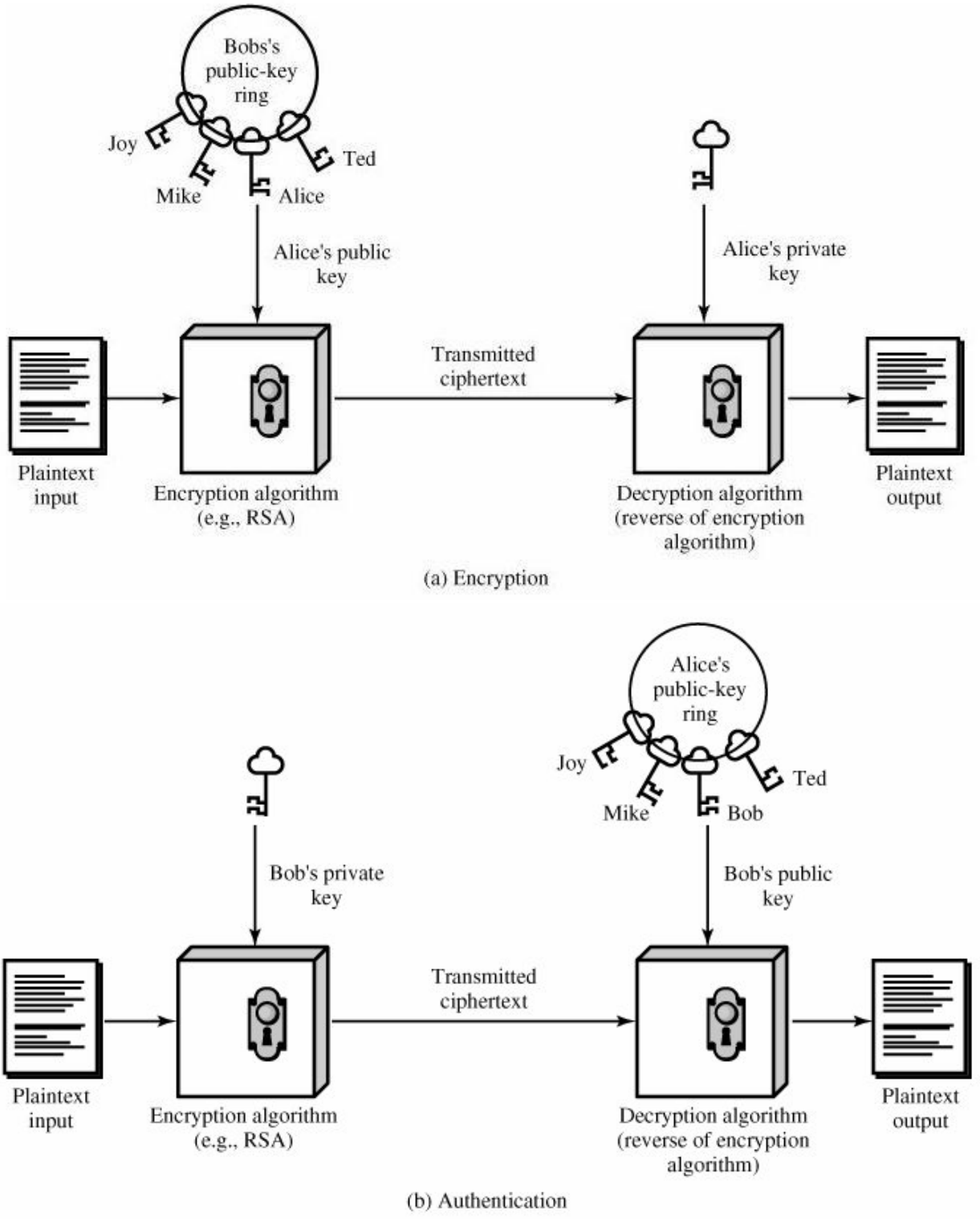

Figure (2.13): Public-Key Cryptography

- ***Plaintex***t: This is the readable message or data that is fed into the algorithm as input.

- ***Encryption algorithm***: The encryption algorithm performs various transformations on the plaintext.

- ***Public and private keys***: This is a pair of keys that have been selected so that if **one is used for encryption, the other is used for decryption**. The exact transformations performed by the algorithm depend on the public or private key that is provided as input.

- ***Cipher text***: This is the **scrambled** message produced as output. It depends on the plaintext and the key. For a given message, two different keys will produce two different cipher texts.
- ***Decryption algorithm***: This algorithm accepts the cipher text and the matching key and produces the original plaintext.
- ***The essential steps are the following:***

1. Each user generates a pair of keys to be used for the encryption and decryption of messages.
2. Each user places one of the two keys in a public register or other accessible file. This is the public key. The companion key is kept private, each user maintains a collection of public keys obtained from others.
3. If Bob wishes to send a confidential message to Alice, Bob encrypts the message using Alice's public key.
4. When Alice receives the message, she decrypts it using her private key. No other recipient can decrypt the message because only Alice knows Alice's private key.

With this approach, all participants have access to public keys, and private keys are generated locally by each participant and therefore need never be distributed. As long as a user's

Private Key remains protected and secret, incoming communication is secure. At any time, a system can change its private key and publish the companion public key to replace its old public key.

In the following Table summarizes some of the important aspects of symmetric and public-key encryption. To discriminate between the two, we refer to the key used in symmetric encryption as a secret key. The two keys used for asymmetric encryption are referred to as the **public key** and the **private key**. Invariably, the private key is kept secret, but it is referred to as a private key rather than a secret key to avoid confusion with symmetric encryption.

Table: Conventional and Public-Key Encryption

Conventional Encryption	***Public-Key Encryption***
Needed to Work:	Needed to Work:
1. The same algorithm with the same key is used for encryption and decryption. 2. The sender and receiver must share the algorithm and the key.	1. One algorithm is used for encryption and decryption with a pair of keys, one for encryption and one for decryption. 2. The sender and receiver must each have one of the matched pair of keys (not the same one).
Needed for Security:	Needed for Security:
1. The key must be kept secret. 2. It must be impossible or at least impractical to decipher a message if no other information is available. 3. Knowledge of the algorithm plus samples of ciphertext must be insufficient to determine the key.	1. One of the two keys must be kept secret. 2. It must be impossible or at least impractical to decipher a message if no other information is available. 3. Knowledge of the algorithm plus one of the keys plus samples of ciphertext must be insufficient to determine the other key.

Let us take a closer look at the essential elements of a public-key encryption scheme, using above Figure (Public) compare with Figure private). There is some source A that produces a message in plaintext, X =[X1, X2... XM,]. The M elements of X are letters in some finite alphabet. The message is intended for destination B. B generates a related pair of keys: a public key, Pub, and a private key, Pub is known only to B, whereas Pub is publicly available and therefore accessible by A.

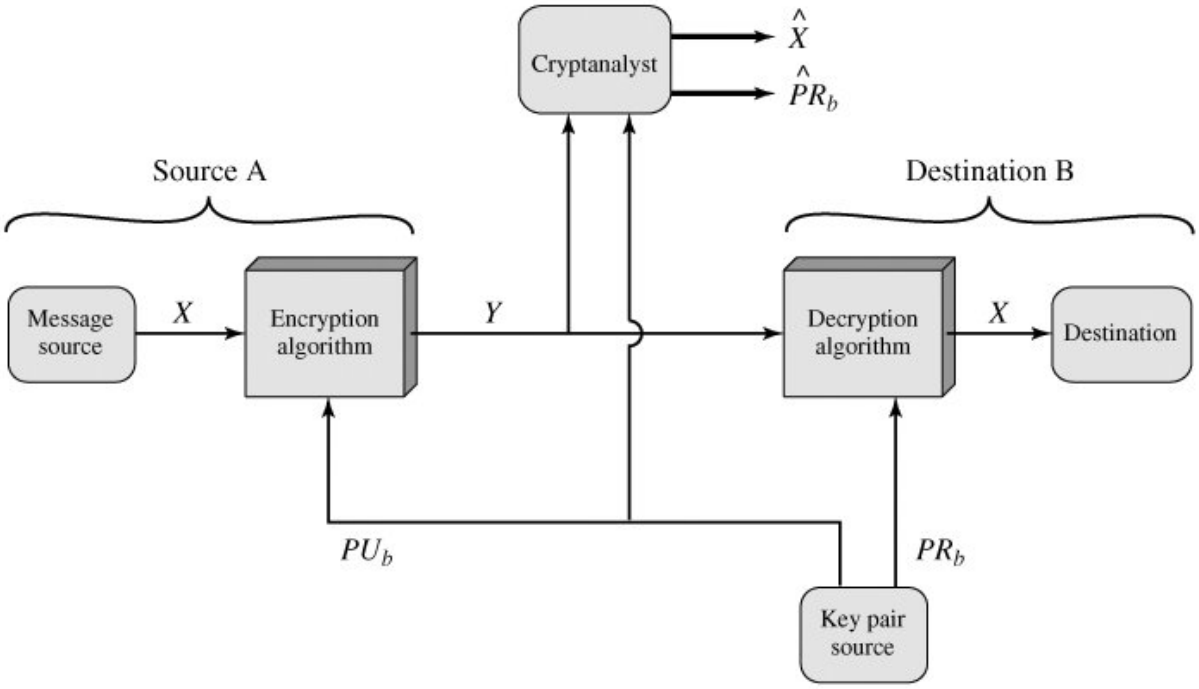

Figure (2.14): Public-Key Cryptosystem: Secrecy

With the message X and the encryption key Pub as input, A forms the cipher text Y = [Y1, Y2... YN]:

Y = E (Pub, X)

The intended receiver, in possession of the matching private key, is able to invert the transformation:

X = D (PRb, Y)

An adversary, observing *Y* and having access to *Pub*, but not having access to *PRb* or *X*, must attempt to recover *X* and/or *PRb*. It is assumed that the adversary does have knowledge of the encryption (E) and decryption (D) algorithms. If the adversary is interested only in this particular message, then the focus of effort is to recover *X* by generating a plaintext estimate *X* ˆ. Often, however, the adversary is interested in being able to read future messages as well, in which case an attempt is made to recover *PRb* by generating an estimate *PR*ˆ*b*. We mentioned earlier that either of the two related keys can be used for encryption, with the other being used for decryption. This enables a rather different cryptographic scheme to be implemented. Whereas the scheme

Illustrated in Figure 2.9 provides confidentiality, Figures 2.8 and 2.10 show the use of public-key encryption to provide authentication:

$Y = \mathrm{E}(PRa, X)$

$X = \mathrm{D}(PUa, Y)$

In this case, A prepares a message to B and encrypts it using A's private key before transmitting it. B can decrypt the message using A's public key. Because the message was encrypted using A's private key, only A could have prepared the message. Therefore, the entire encrypted message serves as a **digital signature**. In addition, it is impossible to alter the message without access to A's private key, so the message is authenticated both in terms of source and in terms of data integrity.

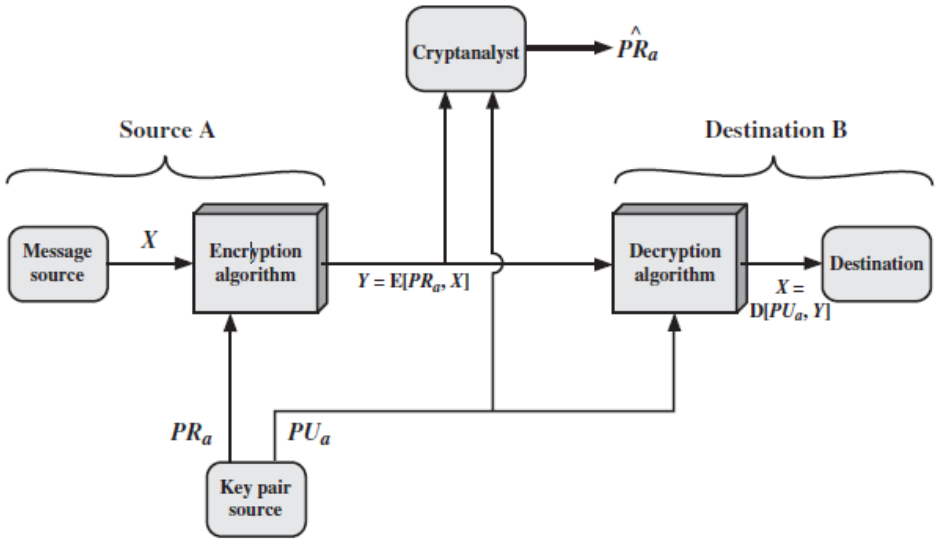

Figure (2.15): public-key cryptosystem: authentication

In the preceding scheme, the entire message is encrypted, which, although validating both author and contents, requires a great deal of storage. Each document must be kept in plaintext to be used for practical purposes. A copy also must be stored in cipher text so that the origin and contents can be verified in case of a dispute. A more efficient way of achieving the same results is to encrypt a small block of bits that is a function of the document. Such a block, called an authenticator, must have the property that it is infeasible to change the document without changing the authenticator. If the authenticator is encrypted with the sender's private key, it serves as a signature that verifies origin, content, and sequencing. Chapter 13 examines this

Technique in detail.

It is important to emphasize that the encryption process depicted in Figures 9.1b and 9.3 does not provide confidentiality. That is, the message being sent is safe from alteration but not from eavesdropping. This is obvious in the case of a signature based on a portion of the message, because the rest of the message is transmitted in the clear. Even in the case of complete encryption, as shown in Figure (2.10), there is no protection of confidentiality because any observer can decrypt the message by using the sender's public key.

It is, however, possible to provide both the authentication function and confidentiality by a double use of the public-key scheme (Figure 2.11):

Z = E (Pub, E (PRa, X))

X = D (PUa, D (PRb, Z))

In this case, we begin as before by encrypting a message, using the sender's private key. This provides the digital signature. Next, we encrypt again, using the receiver's public key. The final cipher text can be decrypted only by the intended receiver, who alone has the matching private key. Thus, confidentiality is provided. The

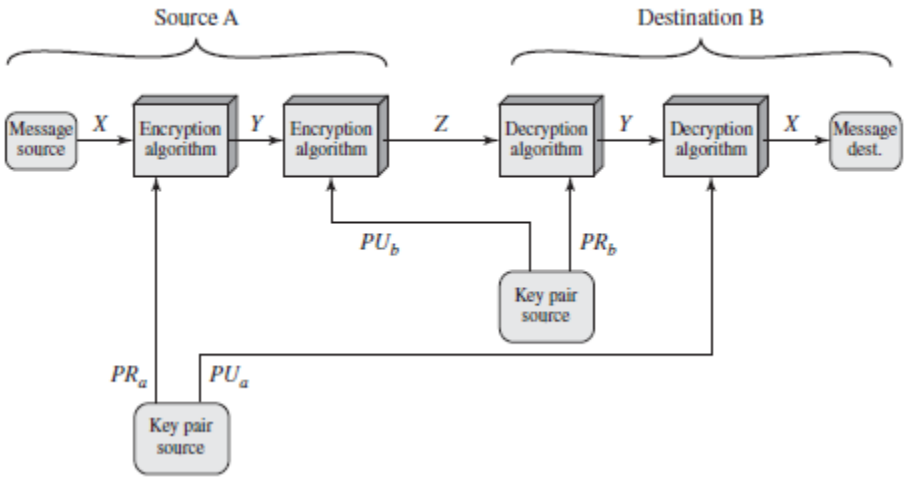

Figure (2.16): public-key cryptosystem: authentication and Secrecy

Disadvantage of this approach is that the public-key algorithm, which is complex, must be exercised four times rather than two in each communication.

2.3.1 DIFFIE-HELLMAN KEY EXCHANGE

The first published public-key algorithm appeared in the seminal paper by Diffie and Hellman that defined public-key cryptography [DIFF76b] and is generally referred to as Diffie-Hellman key exchange.1 a number of commercial products employ this key exchange technique.

The purpose of the algorithm is to enable two users to securely exchange a key that can then be used for subsequent encryption of messages. The algorithm itself is limited to the exchange of secret values.

The Diffie-Hellman algorithm depends for its effectiveness on the difficulty of computing discrete logarithms. Briefly, we can define the discrete logarithm in the following way. Recall from Chapter 8 that a primitive root of a prime number p as one whose powers modulo p generate all the integers from 1 to $p - 1$. That is, if a is a primitive root of the prime number p, then the numbers

$$a \bmod p,\ a^2 \bmod p,\ \ldots,\ a^{p-1} \bmod p$$

are distinct and consist of the integers from 1 through $p - 1$ in some permutation.

For any integer b and a primitive root a of prime number p, we can find a unique exponent i such that

$$b \equiv a^i \pmod{p} \quad \text{where } 0 \le i \le (p - 1)$$

The exponent i is referred to as the **discrete logarithm** of b for the base a, mod p. We express this value as $\text{dlog}_{a,p}(b)$. See Chapter 8 for an extended discussion of discrete logarithms.

The Algorithm

Figure (2.12) summarizes the Diffie-Hellman key exchange algorithm. For this

scheme, there are two publicly known numbers: a prime number q and an integer á that is a primitive root of q. Suppose the users A and B wish to exchange a key. User A selects a random integer $X_A < q$ and computes $Y_A = a^{X_A} \bmod q$. Similarly, user B independently selects a random integer $X_B < q$ and computes $Y_B = a^{X_B} \bmod q$. Each side keeps the X value private and makes the Y value available publicly to the other side. User A computes the key as $K = (Y_B)^{X_A} \bmod q$ and user B computes the key as $K = (Y_A)^{X_B} \bmod q$. These two calculations produce identical results:

$$
\begin{aligned}
K &= (Y_B)^{X_A} \bmod q \\
&= (a^{X_B} \bmod q)^{X_A} \bmod q \\
&= (a^{X_B})^{X_A} \bmod q && \text{by the rules of modular arithmetic} \\
&= a^{X_B X_A} \bmod q \\
&= (a^{X_A})^{X_B} \bmod q \\
&= (a^{X_A} \bmod q)^{X_B} \bmod q \\
&= (Y_A)^{X_B} \bmod q
\end{aligned}
$$

The result is that the two sides have exchanged a secret value. Furthermore, because X_A and X_B are private, an adversary only has the following ingredients to work with: q, a, Y_A, and Y_B. Thus, the adversary is forced to take a discrete logarithm to determine the key. For example, to determine the private key of user B, an adversary must compute

$$X_B = \text{dlog}_{a,q}(Y_B)$$

The adversary can then calculate the key K in the same manner as user B calculates it.

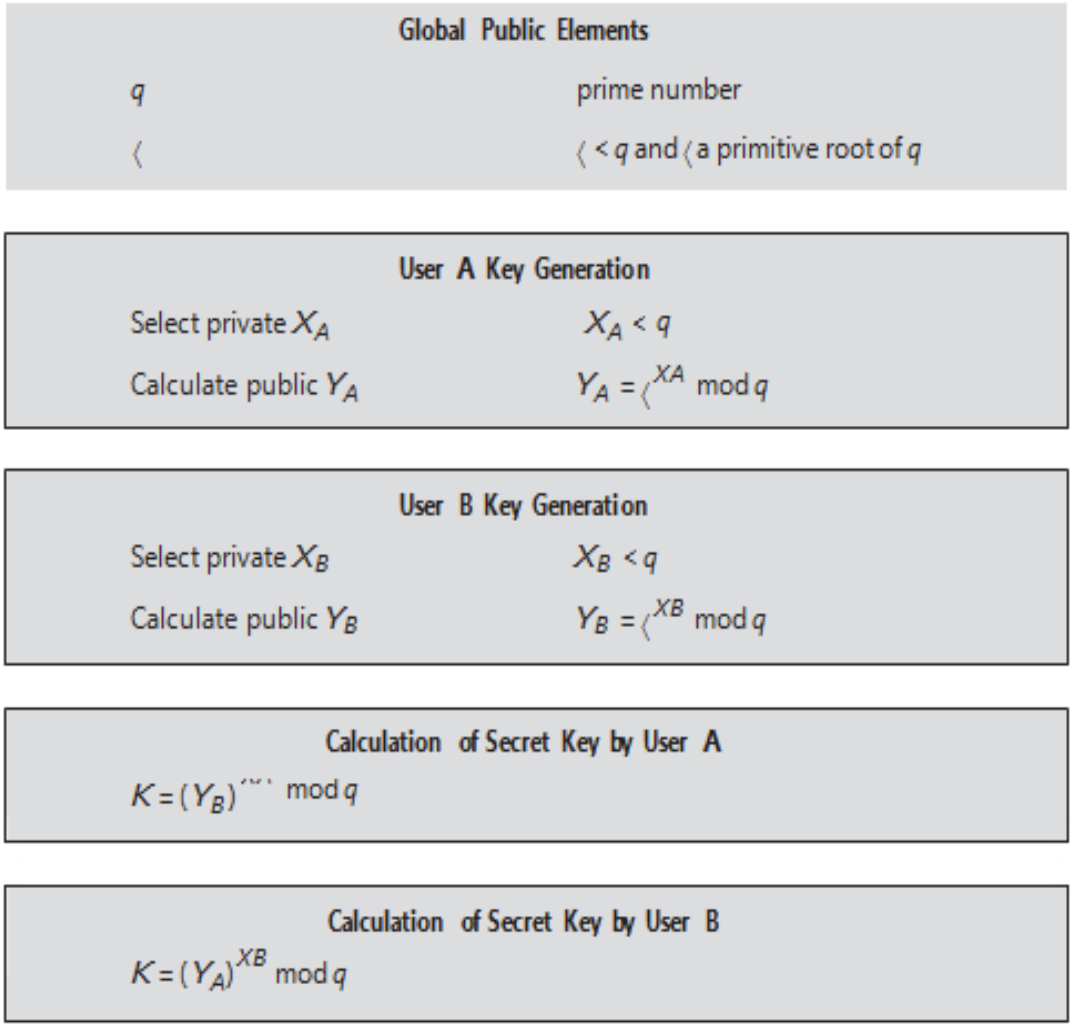

Figure (2.17): The Diffie-Hellman key exchange algorithm

The security of the Diffie-Hellman key exchange lies in the fact that, while it is relatively easy to calculate exponentials modulo a prime, it is very difficult to calculate discrete logarithms. For large primes, the latter task is considered infeasible.

Here is an example. Key exchange is based on the use of the prime number $q = 353$ and a primitive root of 353, in this case $a = 3$. A and B select secret keys $X_A = 97$ and $X_B = 233$, respectively. Each computes its public key:

A computes $Y_A = 3^{97} \bmod 353 = 40$.

B computes $Y_B = 3^{233} \bmod 353 = 248$.

After they exchange public keys, each can compute the common secret key:

A computes $K = (Y_B)^{X_A} \bmod 353 = 248^{97} \bmod 353 = 160$.

B computes $K = (Y_A)^{X_B} \bmod 353 = 40^{233} \bmod 353 = 160$.

We assume an attacker would have available the following information:

$q = 353$; a = 3; $Y_A = 40$; $Y_B = 248$

In this simple example, it would be possible by brute force to determine the secret key 160. In particular, an attacker E can determine the common key by discovering a solution to the equation $3^a \bmod 353 = 40$ or the equation $3^b \bmod 353 = 248$. The brute-force approach is to calculate powers of 3 modulo 353, stopping when the result equals either 40 or 248. The desired answer is reached with the exponent value of 97, which provides $3^{97} \bmod 353 = 40$.

With larger numbers, the problem becomes impractical

Key Exchange Protocols

Figure (2.13) shows a simple protocol that makes use of the Diffie-Hellman calculation. Suppose that user A wishes to set up a connection with user B and use a secret key to encrypt messages on that connection. User A can generate a one-time private key X_A, calculate Y_A, and send that to user B. User B responds by generating a private value X_B, calculating Y_B, and sending Y_B to user A. Both users can now calculate the key. The necessary public values q and a would need to be known ahead of time. Alternatively, user A could pick values for q and a and include those in the first message.

As an example of another use of the Diffie-Hellman algorithm, suppose that a group of users (e.g., all users on a LAN) each generate a long-lasting private value X_i (for user i) and calculate a public value Y_i. These public values, together with global public values for q and □, are stored in some central directory. At any time, user j can access user i's public value, calculate a secret key, and use that to send an encrypted message to user A. If the central directory is trusted, then this form of communication provides both confidentiality and a degree of authentication. Because only i and j can determine the key, no other user can read the message (confidentiality). Recipient i knows that only user j could have created a message using this key (authentication). However, the technique does not protect against replayattacks.

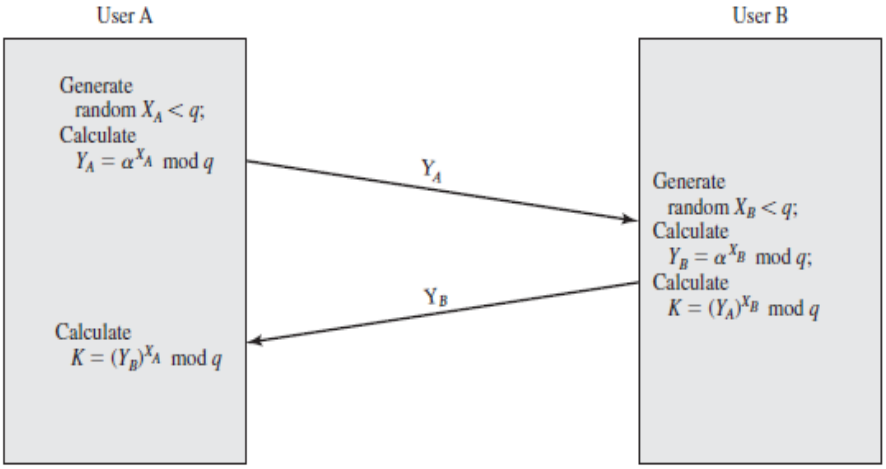

Figure (2.18): The Diffie-Hellman key exchange

CHAPTER THREE

New approach for serpent algorithm

3.1 Introduction

Serpent history this algorithm was developed in 1998 by three researches: Ross Anderson, Lars Knudsen and Eli Biham. Serpent is an open source algorithm that encrypts in blocks, also called symmetric key. Basically only one key is used to encrypt and decrypt messages. We designed serpent to provide users with the highest practical level of assurance that no shortcut attack will be found. To achieve this, we limited ourselves to well understood mechanisms, so that we could rely on the existing experience of block cipher cryptanalysis. We also used twice as many rounds as are sufficient to block all currently known shortcut attack. We believed this to be prudent practice for a cipher that might have a service life of a century or more. It was a finalist in the AES competition. the main different between AES and serpent is that AES has less "rounds" and less encrypting levels making it faster in the encrypting and decrypting processes. Serpent instead is slower but safer and serpent is more secure.

3.2 New approach for serpent algorithm

Serpent is a 32-round SP-network operating on two 256-bit words, thus giving a block size of 512 bits. All values used in the cipher are represented as bit streams. The indices of the bits are counted from 0 to bit 255 in one 256-bit word, 0 to bit 511 in 512-bit blocks, 0 to bit 255 in 256-bit keys, and so on. For internal computation, all values are represented in little-endian, where the first word (word 0) is the least significant word, and the last word is the most significant, and where bit 0 is the least significant bit of word 0. Externally, we write each block as a plain 512-bit hex number. Serpent encrypts a 512-bit plaintext *P* to a 512-bit cipher text *C* in 32 rounds under the control of 33 128-bit sub keys *K*0… *K32*. The user key length is variable, but for the purposes of this submission we fix it at 128, 192 or 256 bits, short keys with less than 256 bits are mapped to full-length keys of 256 bits by appending one “1" bit to the MSB end, followed by

As many “0" bits as required to make up 256 bits. This mapping is designed to map every short key to a full-length key, with no two short keys being equivalent. The serpent algorithm cipher consists of:

- **initial permutation IP /final IP**
- **s-box /invers s-box**
- **linear transformation /invers linear transformation**
- **function feal / invers function feal**
- **shift rows transformation / invers shift rows transformation**

3.2.1 Encryption algorithm

INPUT: 512 bits the plaintext to encrypt, and 512 bits the key

OUTPUT: 512 bits as encrypted cipher text

BEGIN

1. Split the plaintext (512 bits) into 2 part each part 256 bit (L, R), R will processed in IP

and the result from IP will processed in shift (5 bit) from left side , L will processed in shift(5 bit) from right side and swap L with R to get new right and new left

2. New left will mix with key (XOR operation) then pass through function feal and s-box to get final L

3. New right will mix with key (XOR operation) then pass through function shift row transformation and ip invers to get final R

4. Marge final right with final left, then pass through linear transformation

END

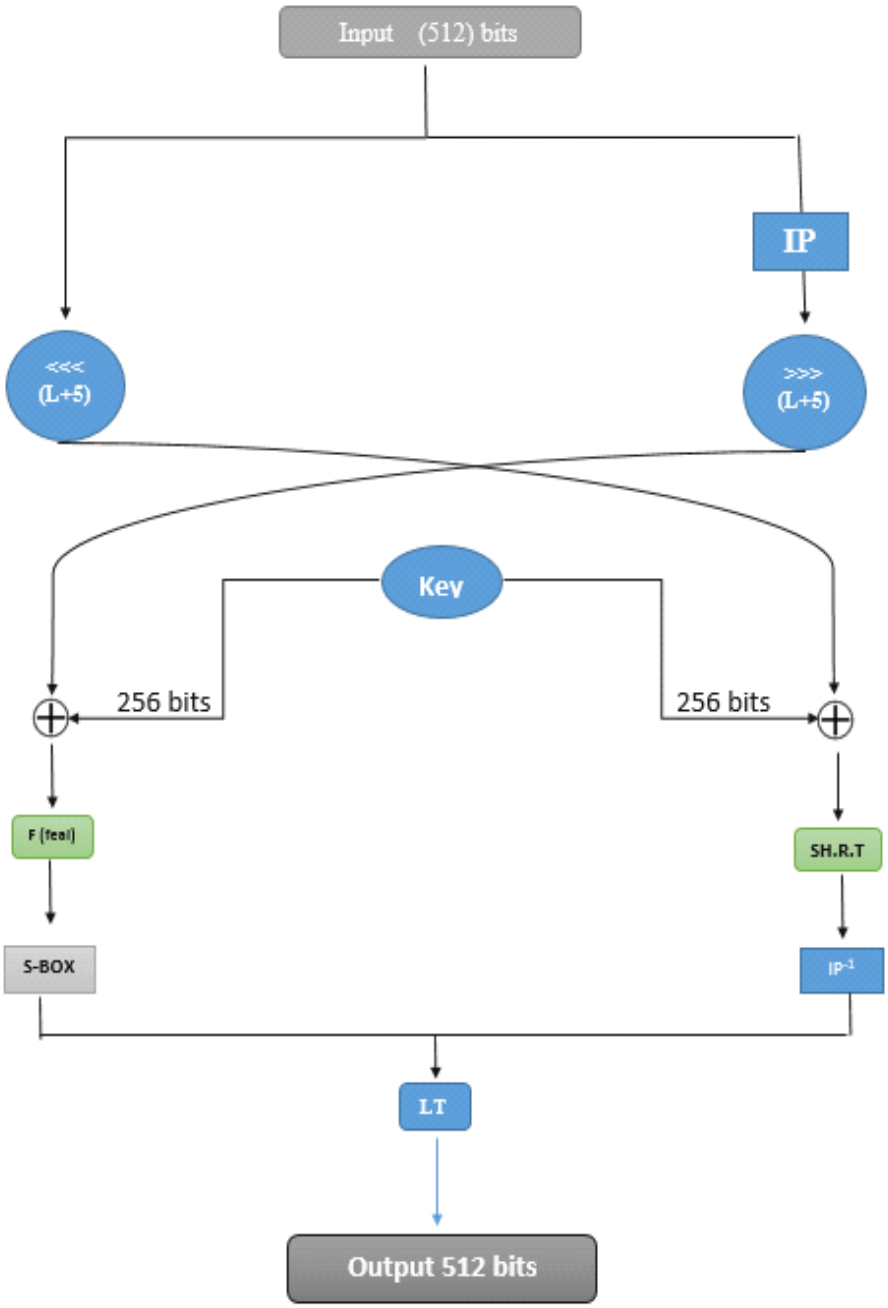

Figure (3.1): encryption of serpent

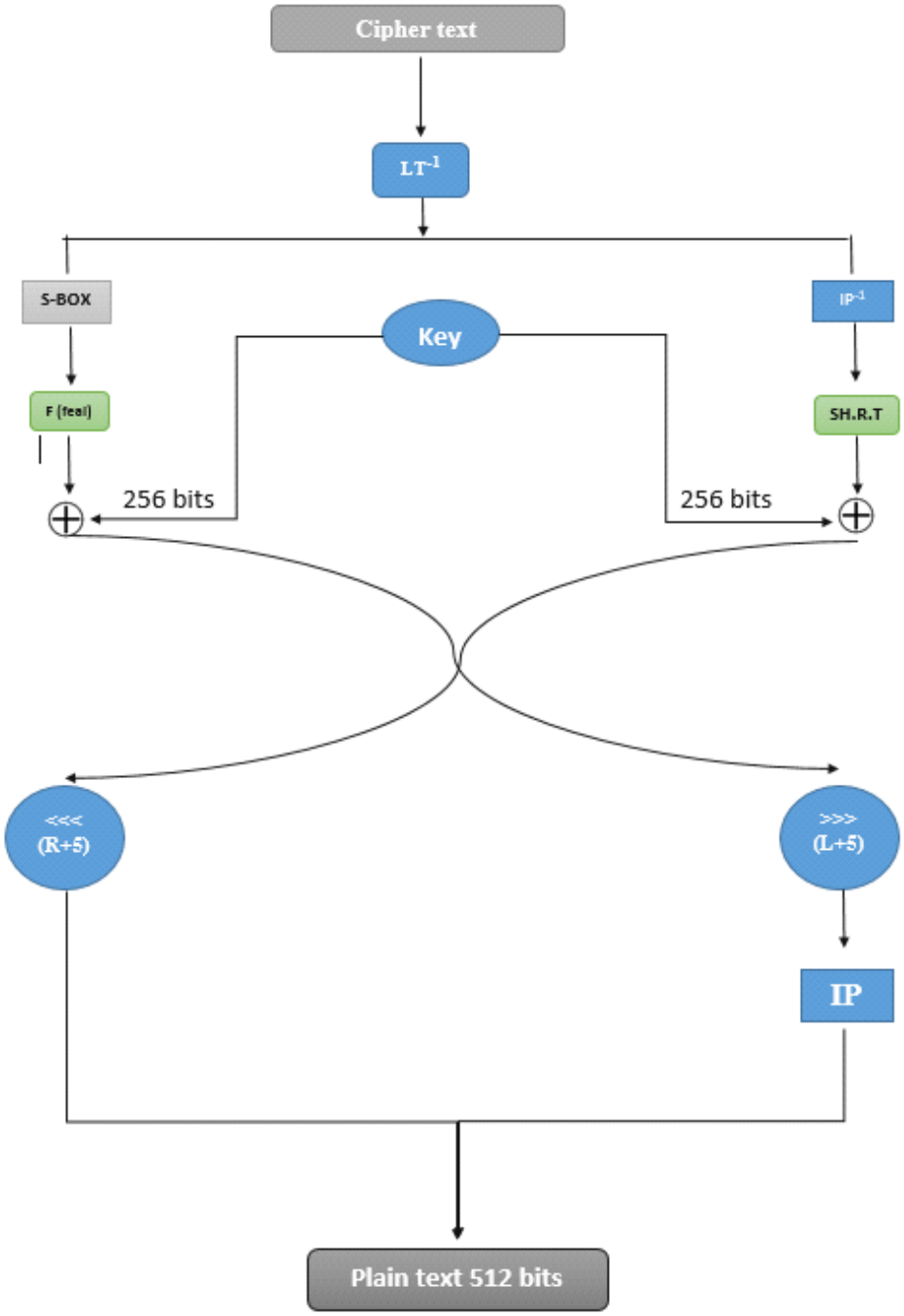

Figure (3.2): decryption of serpent

•the initial permutation

The initial and final permutations do not have any cryptographic. They are used to simplify an optimized implementation of the cipher. Both these two permutations and the linear transformation are to add more complexity. Can explain the initial permutation and final permutation in the following tables:

Table (3.1): The initial and final permutations

Initial permutation

{58, 50, 42, 34, 26, 18, 10, 2, 60, 52, 44, 36, 28, 20, 12, 4, 62, 54, 46, 38, 30, 22, 14, 6, 64, 56, 48, 40, 32, 24, 16, 8, 57, 49, 41, 33, 25, 17, 9, 1, 59, 51, 43, 35, 27, 19, 11, 3, 61, 53, 45, 37, 29, 21, 13, 5, 63, 55, 47, 39, 31, 23, 15, 7, 122, 114, 106, 98, 90, 82, 74, 66, 124, 116, 108, 100, 92, 84, 76, 68, 126, 118, 110, 102, 94, 86, 78, 70, 128, 120, 112, 104, 96, 88, 80, 72, 121, 113, 105, 97, 89, 81, 73, 65, 123, 115, 107, 99, 91, 83, 75, 67, 125, 117, 109, 101, 93, 85, 77, 69, 127, 119, 111, 103, 95, 87, 79, 71}

Invers initial permutation

{40, 8, 48, 16, 56, 24, 64, 32, 39, 7, 47, 15, 55, 23, 63, 31, 38, 6, 46, 14, 54, 22, 62, 30, 37, 5, 45, 13, 53, 21, 61, 29, 36, 4, 44, 12, 52, 20, 60, 28, 35, 3, 43, 11, 51, 19, 59, 27, 34, 2, 42, 10, 50, 18, 58, 26, 33, 1, 41, 9, 49, 17, 57, 25, 104, 72, 112, 80, 120, 88, 128, 96, 103, 71, 111, 79, 119, 87, 127, 95, 102, 70, 110, 78, 118, 86, 126, 94, 101, 69, 109, 77, 117, 85, 125, 93, 100, 68, 108, 76, 116, 84, 124, 92, 99, 67, 107, 75, 115, 83, 123, 91, 98, 66, 106, 74, 114, 82, 122, 90, 97, 65, 105, 73, 113, 81, 121, 89}

•the s-box

The set of eight S-boxes is used four times. Thus after using *S*7 in round 7, we use *S*0 again in round 8, then *S*1 in round 9, and so on. The last round *R*31 is slightly different from the others, apply *S*7 on ^*B*31 *XOR* ^*K*31, and XOR the result with ^*K*32 rather than applying the linear transformation. The result ^*B*32 is then permuted by *FP*, giving the cipher text.

Thus the 32 rounds use 8 different S-boxes each of which maps four input bits to four output bits. Each S-box is used in precisely four rounds, and in each of these it is used 32 times in parallel. Can explain the in the following table:

Table (3.2): the s-box and invers s-box

Table s-box

S0 = {3, 8, 15, 1, 10, 6, 5, 11, 14, 13, 4, 2, 7, 0, 9, 12}

S1 = {15, 12, 2, 7, 9, 0, 5, 10, 1, 11, 14, 8, 6, 13, 3, 4}

S2 = {8, 6, 7, 9, 3, 12, 10, 15, 13, 1, 14, 4, 0, 11, 5, 2}

S3 = {0, 15, 11, 8, 12, 9, 6, 3, 13, 1, 2, 4, 10, 7, 5, 14}

S4 = {1, 15, 8, 3, 12, 0, 11, 6, 2, 5, 4, 10, 9, 14, 7, 13}

S5 = {15, 5, 2, 11, 4, 10, 9, 12, 0, 3, 14, 8, 13, 6, 7, 1}

S6 = {7, 2, 12, 5, 8, 4, 6, 11, 14, 9, 1, 15, 13, 3, 10, 0}

S7 = {1, 13, 15, 0, 14, 8, 2, 11, 7, 4, 12, 10, 9, 3, 5, 6}

Table invers s-box

IS0 = {13, 3, 11, 0, 10, 6, 5, 12, 1, 14, 4, 7, 15, 9, 8, 2}

IS1 = {5, 8, 2, 14, 15, 6, 12, 3, 11, 4, 7, 9, 1, 13, 10, 0}

IS2 = {12, 9, 15, 4, 11, 14, 1, 2, 0, 3, 6, 13, 5, 8, 10, 7}

IS3 = {0, 9, 10, 7, 11, 14, 6, 13, 3, 5, 12, 2, 4, 8, 15, 1}

IS4 = {5, 0, 8, 3, 10, 9, 7, 14, 2, 12, 11, 6, 4, 15, 13, 1}

IS5 = {8, 15, 2, 9, 4, 1, 13, 14, 11, 6, 5, 3, 7, 12, 10, 0}

IS6 = {15, 10, 1, 13, 5, 3, 6, 0, 4, 9, 14, 7, 2, 12, 8, 11}

IS7 = {3, 0, 6, 13, 9, 14, 15, 8, 5, 12, 11, 7, 10, 1, 4, 2}

- **linear transformation**

Linear Transformation input by four 32 bit words X0,X1,X2,X3, with X3 being the most significant (left most) 32 bits of the input , the linear transformation can be described by the following sequence of operations:

X0, X1, X2, X3:=Si (Bi (XOR) Ki)

X0:= X0 <<< 13

X2:= X2 <<< 3

X1:= X1 (XOR) X0 (XOR) X2

X3:= X3 (XOR) X2 (XOR) (X0 <<3)

X1:=X1 <<< 1

X3:=X3 <<< 7

X0:= X0 (XOR) X1 (XOR) X3

X2:= X2 (XOR) X3 (XOR) (X1 <<7)

X0:=X0 <<< 5

X2:=X2 <<< 22

Bi+1:= X0, X1, X2, X3

Where <<< denotes rotation, and << denotes shift. In the last round, this linear transformation is replaced by an additional key mixing: B32 = S7 (B31 (XOR) K31) XOR K32 .can explain in the following figure.

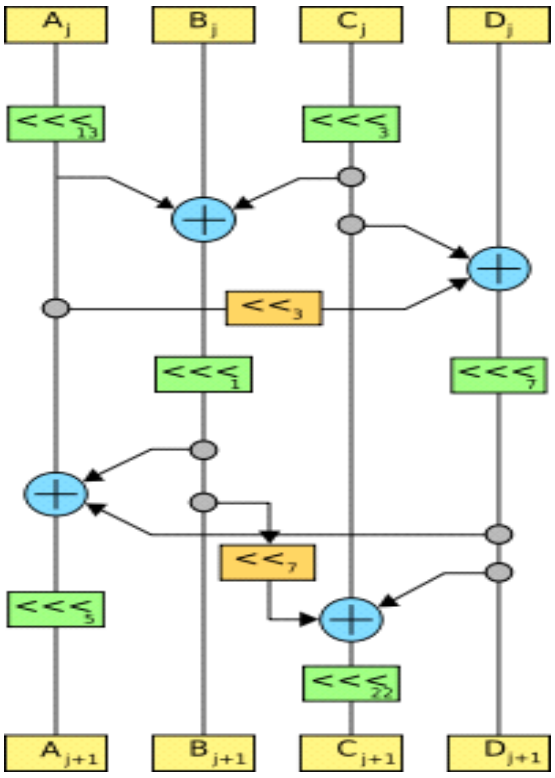

Figure (3.3) linear transformation

- **shift rows transformation**:

The purpose of the step shift rows is to spread the bytes of each. Input column to different output column. This step is a linear diffusion process when operates on individual rows and each row of the array is rotated by a certain number of byte positions.

- **forward shift rows transformations**:

The forward shift row transformation is called shift rows, this operation rotates the 2nd, 3rd and 4th row of the matrix. The first row (row 0) is not shifted, and the remaining rows proceed as follows, for the second row (row1) a 1-byte circular left shift is performed, for the third row 2-byte circular left shift is performed and for the fourth row 3-byte circular left shift is performed, see figure (3.4)

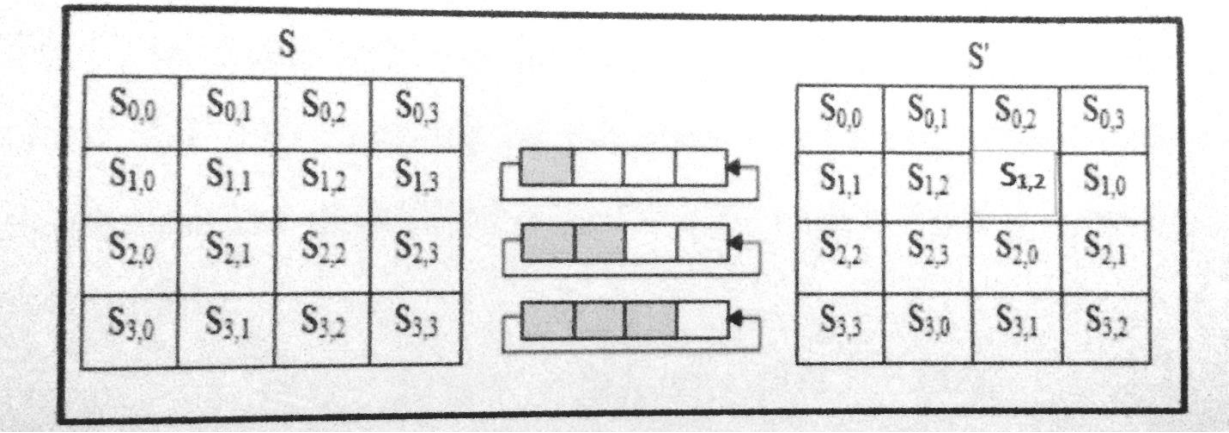

Figure (3.4): shift rows transformation

•Invers shift rows transformations:

For decryption, the corresponding step shifts the rows in exactly the opposite fashion. The first row is left unchanged, the second row is shifted to the right by one byte, the third row to the right by two bytes and the last row to the right by three bytes, and all shifts are being circular, as show in figure (3.5)

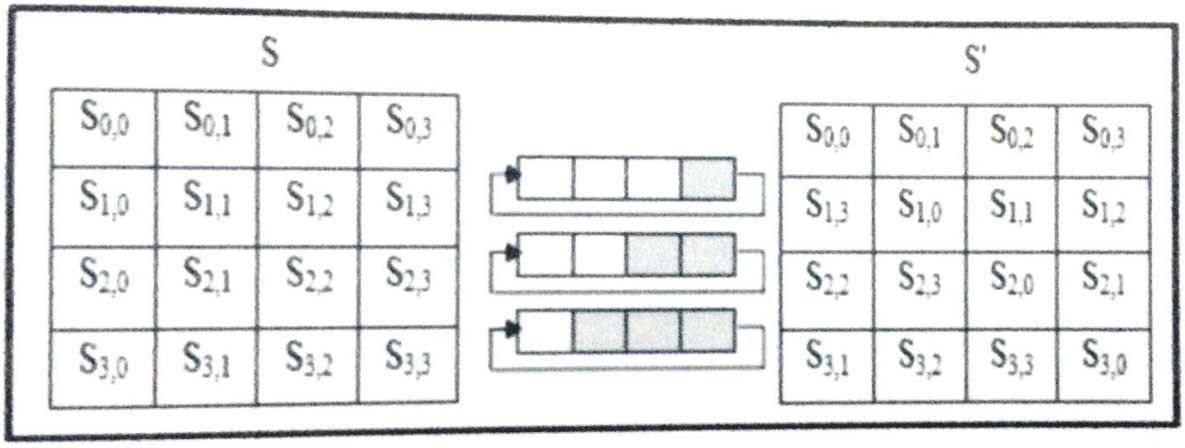

Figure (3.5): invers shift rows transformation

Function feal

Function f takes the 32 bits of data and 16 bits of key material and mixes them together. First the data block

Is broken up into 8 bit chunks, then the chunks are XORed and substituted with each other. Figure (3.6) is

A block diagram of function. The two function S0 and S1 are defined as:

S0 (a, b) =rotate left two bits ((a+b) mod 256)

S1 (a, b) =rotate left two bits ((a+b+1) mod 256)

The same algorithm can be used for decryption. The only difference is: when decrypting, the key material must be used in the reverse order.

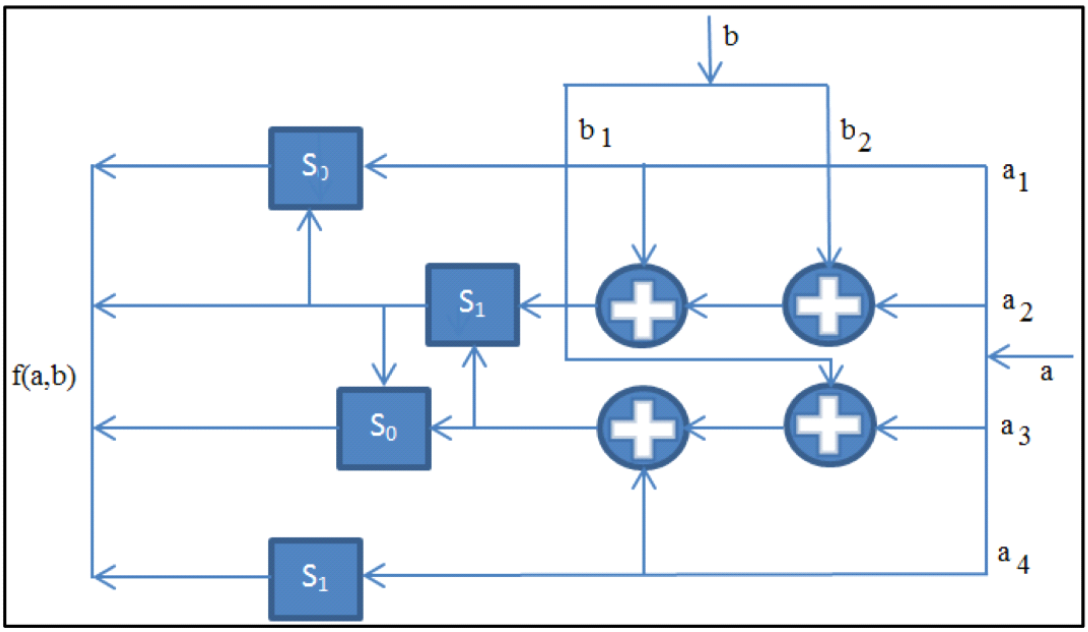

Figure (3.6) function feal

- **Time of encryption and decryption**

Text file size	operation	New approach in sec.
1kb	Encryption	6
	Decryption	6
2kb	Encryption	41
	Decryption	41
3kb	Encryption	82
	Decryption	82

The proposal algorithm implementation in visual studio 2010.net, the input data as text file with static key ,and the propoal consisted of three pages the master home page and the running pag of encryption and running page of decryption ,can explain in the following interface:

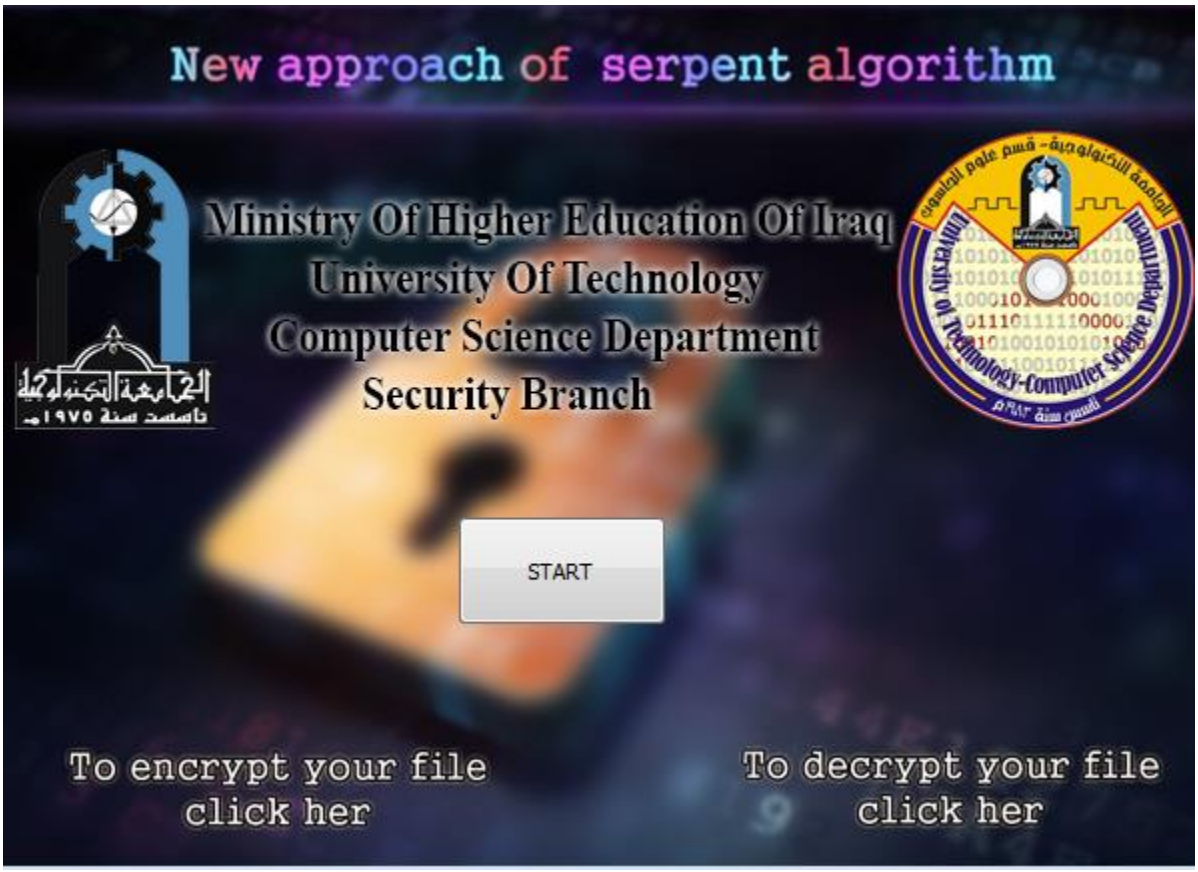

Figure (3.7): the master home page

This page consist of one command to move to page of encryption

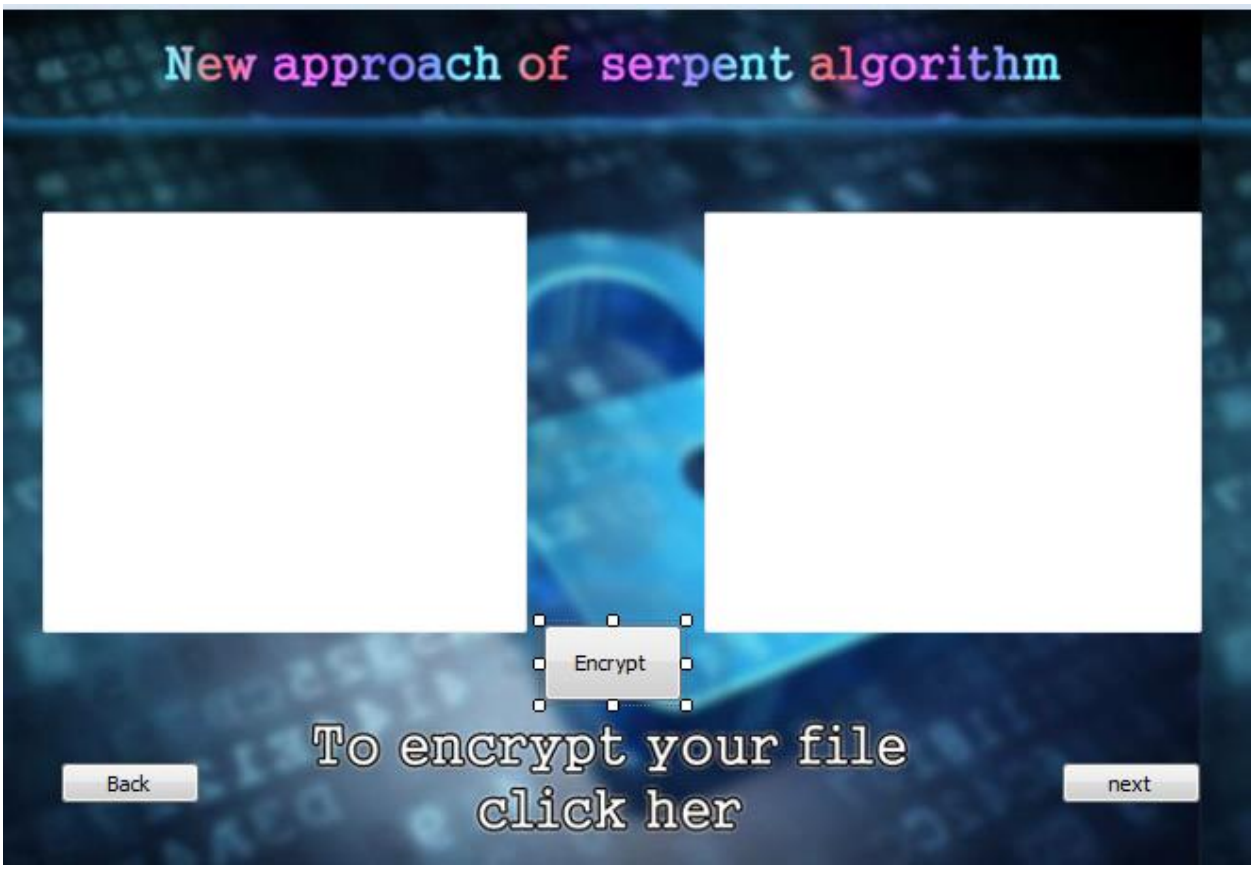

Figure (3.8): the running page of encryption algorithm

In the running page of encryption algorithm we have the three command (encrypt, back and next), the encrypt command will encrypt the text file ,the back command to back to the master home page and the next command to move to the decryption page

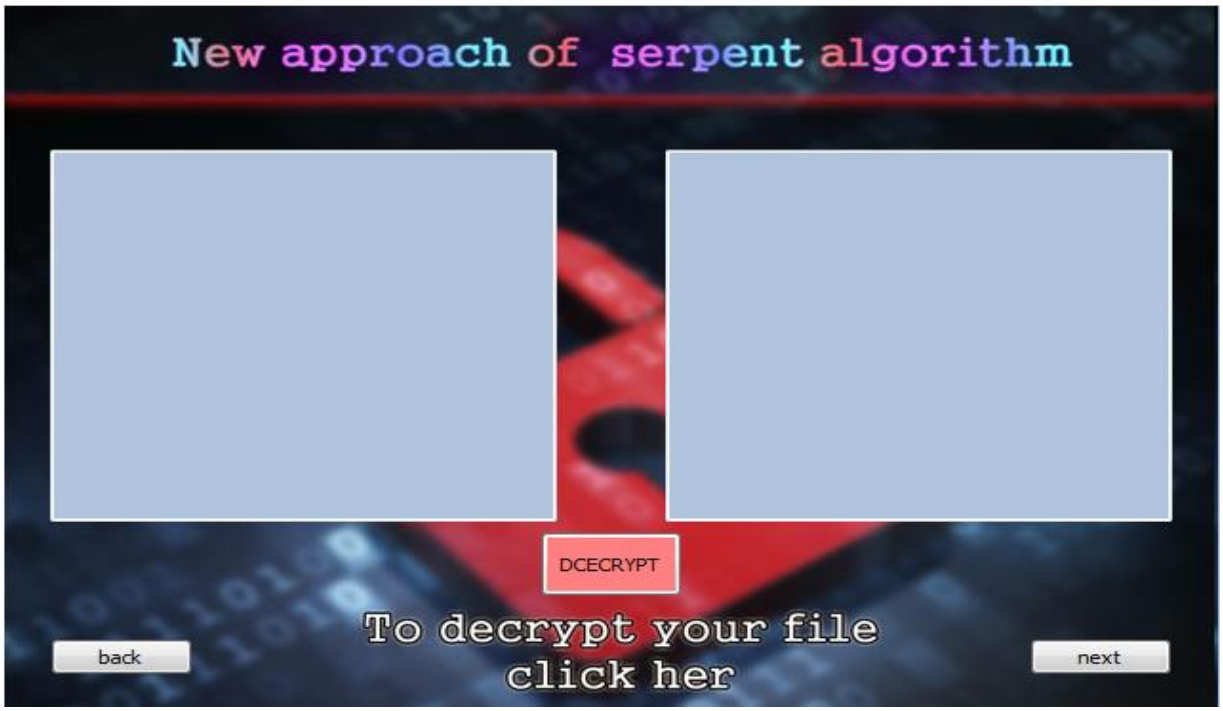

Figure (3.8): the running page of decryption algorithm

In the running page of decryption algorithm we have the three command (decrypt, back and next), the decrypt command will decrypt the text file, the back command to back to the encryption page and the next command to move to the decryption page

CHAPTER FOUR

Conclusions

4.1 Conclusions

1. The original linear transformation take more time in execution, in new proposal this problem has been processing through split the input into two blocks.

2. The SBOX is add more complex to the algorithm because no one can suggest what is the value of s-box (s-box apply the principle confusion)

3. The initial permutation add complex to the algorithm (IP apply the principle diffusion)

4. The function feal add complex to the algorithm (f-feal apply the principle diffusion)

4.2 suggestion

1. We suggest new method to generator secret key using in new serpent

2. adding mathematical to the algorithm to more strong in future

References

- BK03_Internet Security - Cryptographic Principles, Algorithms and Protocols
- BK06_Complexity and Cryptography - An Introduction
- Cryptography and Network Security Principles and Practice, 5th Edition
- BK07_Computer Security and Cryptography