ENCRYPTION AND DECRYPTION OF DATA

A THESIS

SUBMITTED IN PARTIAL FULFILLMENT OF THE REQUIREMENT FOR THE AWARD OF THE DEGREE
OF
MASTER OF ENGINEERING
IN
SOFTWARE ENGINEERING
BY

GAURAV VERMA

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

THAPAR INSTITUTE OF ENGINEERING AND TECHNOLOGY(DEEMED UNIVERSITY)PATIALA

CERTIFICATE

This is to certify that the thesis work entitled "Encryption and Decryption of Data" submitted by, Gaurav-Verma in partial fulfillment of the requirement for the award of the degree of Master of Engineering (Software Engineering), TIET(Deemed University),Patiala, is a record of candidates own work carried by him under my guidance.

Gaurav Verma
ME(8003102)

Dr P.K. Bansal
Profeesor
Dept of Computer
Science and Engineering
TIET, Patiala.

Ms. Seema Bawa
Head
Dept of Computer
Science and Engineering
TIET, Patiala.

5/6/2002

Dean
Academic Affairs
TIET, Patiala.

ACKNOWLEDGEMENT

I sincerely wish to express my heartiest gratitude to my supervisor Dr. P.K. Bansal, whose initiatives, keen interest and valuable guidance provide a constant source of inspiration and encouragement to me to carry out this work.

I am also indebted to all the persons who supported me and assisted during my study and thesis

Gaurav Verma
ME(8003102)

ABSTRACT

The selective application of technological and related procedural safeguards is an important responsibility of every organization in providing adequate security to its electronic data systems. This work specifies a cryptographic program which may be used by organizations to protect sensitive data. Protection of data during transmission or while in storage may be necessary to maintain the confidentiality and integrity of the information represented by the data. The program developed in this work, emphasizes on the key and mixing with the data. The program uses 64 bit key length. Initially the bits are subjected to permutation which produces a new pattern of bits and then further undergo modulo-2 addition, which produces a new key different from the key that was initially entered. This system is being made available for use by organizations within the context of a total security program consisting of physical security procedures, good information management practices, and computer system/network access controls.

CONTENTS

Certificate	I
Acknowledgement	II
Abstract	III
Chapter 1 Introduction	1
1.1 Introduction	1
1.2 Problem Specification	5
1.3 Organization of the Thesis	6
Chapter 2 Cryptography	7
2.1 Substitution Cipher	10
2.2 Transposition Cipher	13
2.3 Conclusion	16
Chapter 3 Algorithm for Data Encryption	17
3.1 Components of Algorithm	21
3.1.1 Key	21
3.1.2 Modulo-2 Addition	22
3.1.3 The Cipher Function	22
3.1.4 Pre-Output Block	29
3.1.5 Initial Permutation	29
3.1.6 Decryption	30
3.2 Conclusion	33
Chapter 4 Conclusion and Further Scope	34
References	35
Glossary	36
Appendix	38

CHAPTER 1

INTRODUCTION

1.1 INTRODUCTION

For the first few decades of their existence, computer networks were primarily used by university researchers for sending email, and by corporate employees for sharing printers. Under these conditions, security did not get much attention. Buy now, as million of citizens are using networks for banking, shopping, and filling their tax returns, network security is looming on the horizon as a potentially massive problem.

Network security in its broadest sense is concerned with making sure that nosy people cannot read or worse yet, modify messages intended for other recipients. It is concerned with people trying to access remote services that are not authorized to use. Security also deals with problems of legitimate messages being captured and replayed, and with people trying to deny that they sent certain messages.

Network security problems can be divided into four areas: secrecy, authentication, non-repudiation, and integrity control. Secrecy has to do with keeping information out of the hands of unauthorized user. Authentication deals with determining whom you are talking to before revealing sensitive information or entering into a business deal. Non repudiation deals with signatures: How do you prove that any customer really placed an electronic order for ten million left handed doohickeys at 89 cents when latter claim

that price was 69 cents? Finally, how can you be sure that a message you receive was really the one sent and not something that a malicious adversary modified in transit or concocted?

A suitable methodology for protecting communicated or stored data involves the use of Cryptographic. The Cryptographic systems are generically classified along three independent dimensions:-

1 **The type of operations used for transforming plaintext to ciphertext.** All encryption algorithms are based on two general principles: substitution, in which each element in the plaintext is mapped into another element, and transposition, in which elements in the plaintext are rearranged. The fundamental requirement is that no information be lost. Most systems, referred to as product systems, involves multiple stages of substitution and transpositions.

2 **The number of keys used.**

If both sender and receiver use the same key, the system is referred to as symmetric, single-key, secret-key, or conventional encryption. If the sender and receiver each uses a different key, the system is referred to as asymmetric, two-key, or public-key encryption.

3 The way in which the plaintext is processed.

A block cipher processes the input one block of elements at a time, producing an output block for each input block. A stream cipher processes the input elements continuously, producing one element at a time, as it goes along.

But with the cryptography there are various attacks associated with it. One possible attack is brute-force approach of trying all possible keys. If the key space is very large, this becomes impractical. The ciphertext-only attack is the easiest to defend against because the opponent has the least amount of information to work with. In many cases, however the analyst has more information. The analyst may be able to capture one or more plaintext messages as well as their encryptions. Or the analyst may know that certain plaintext patterns will appear in a message. With this knowledge, the analyst may be able to deduce the key on the basis of the way in which the known plaintext is transformed.

Closely related to the known-plaintext attack is what might be referred to as a probable-word attack. If the opponent is working with encryption of some general prose message, he or she may have little knowledge of what is in the message. However, if the opponent is after some specific information, then parts of the message may be known. For example, if an entire accounting file is being transmitted, the opponent may know the placement of certain key words in the header of the file.

If the analyst is able somehow to get the source system to insert into the system a message chosen by the analyst, then a chosen-plaintext attack is possible. An example of this strategy is differential cryptanalysis. In general, if the analyst is able to choose the message to encrypt, the analyst may deliberately pick patterns that can be expected to reveal the structure of the key.

It is worthy to note that an encryption scheme is secure if the ciphertext generated by the scheme does not contain enough information to determine uniquely the corresponding plaintext, no matter how much ciphertext is available. That is, no matter how much time an opponent has, it is impossible for him or her to decrypt the ciphertext, simply because the required information is not there. But still there is no encryption scheme that is unconditionally secure. But it should be able to meet one or more of the following criteria:

- The cost of breaking the cipher exceeds the value of the encrypted information.
- The time required to break the cipher exceeds the useful lifetime to the information.

An encryption scheme is said to be computationally secure if the foregoing two criteria are met. The rub is that it is very difficult to estimate the amount of effort required to cryptanalyze ciphertext successfully. As a first cut, we can consider the time required to use a brute-force approach which simply involves trying every possible key until an intelligible transformation of the

ciphertext into plaintext is obtained. On average, half of all possible keys must be tried to achieve the success. Keeping in notice of all the foregoing criteria and time required here a program have been developed which uses a 64-bit key. If we assume that it takes 1us then time required for 1encryption/1us will be greater than 1142 years. Since there is no exception that it cannot be broken but making as hard as possible to break is what have been tried in this work.

1.2 PROBLEM SPECIFICATION

The requirements of information security have undergone major changes in the last several decades. Before the widespread use of data processing equipment, the security of information felt to be valuable to an organization was provided primarily by physical and administrative means. But with the usage of the computer, the need for protection of information became all the more important. Further with the introduction of distributed systems and the use of networks and communication facilities for carrying data between computer and computer there is a need to protect data during the transmission. So to solve this kind of a problem there is a need of development of a cryptographic program where the plaintext should be converted to the ciphertext in such a way that all the steps should not be totally dependent on the key and the length of the key should be sufficiently large. The steps of processing the key should be such that, where the outcome of the key(i.e. the key that is input to the program) should be different from what was entered by the user.

1.3 ORGANIZATION OF THE THESIS

The first chapter focuses on how the computer networks came to existence and how more and more people started using networks for information exchange and other purposes. Then the focus is on the network security problems and how to make the information secure .An overview of cryptography is also covered in this chapter.

The second chapter is devoted to cryptography. This chapter explains the importance of cryptography and the various keywords used. It also describes the traditional encryption model and it's limitations. The importance of the key and it's role is also discussed. The various encryption techniques are covered.

The third chapter discusses with the development of the cryptography program. This chapter deals with the development of an algorithm, where all the components of an algorithm are explained along with the related diagrams and examples. Then the emphasis on the development of the key is also covered.

Finally the conclusion and further scope of work is covered in chapter fourth.

CHAPTER 2

CRYPTOGRAPHY

The idea of cryptography is to disguise confidential information in such a way that its meaning is not readable to an unauthorized person. Transforming the text denotes the use of cryptographic methods to protect certain communicated and stored data against theft or misused. These methods generally depends upon the nature of the threat to data security.

The simplest method of encipherment can prevent accidental disclosure, while the most sophisticated technique would provide the maximum protection for sensitive data against penetration of the data security system. The message to be encrypted known as plaintext, are transformed by a function that is parameterized by a key. The output of the encryption process is known as ciphertext. The person who enciphers the message is known as encipher while the person to whom he sends the cryptogram is called the recipient. Normally the operation will depend on a key which the encipher inputs to the algorithm together with the message. The recipient knows the key and this knowledge should enable him to determine the plain text from the ciphertext.

Any person who intercepts a message being transmitted from the encipher to recipient is called interceptor or intruder. An interceptor will not in general know the key and it is the lack of knowledge, which is hoped, will prevent him from knowing the plain text. The art of breaking ciphers is called cryptanalysis and the art of devising ciphers and breaking them is

collectively is known as cryptology. The process of applying a key to translate back from ciphertext to plain text is known as deciphering. The model, which shows encryption and decryption, is shown in Figure 2.1

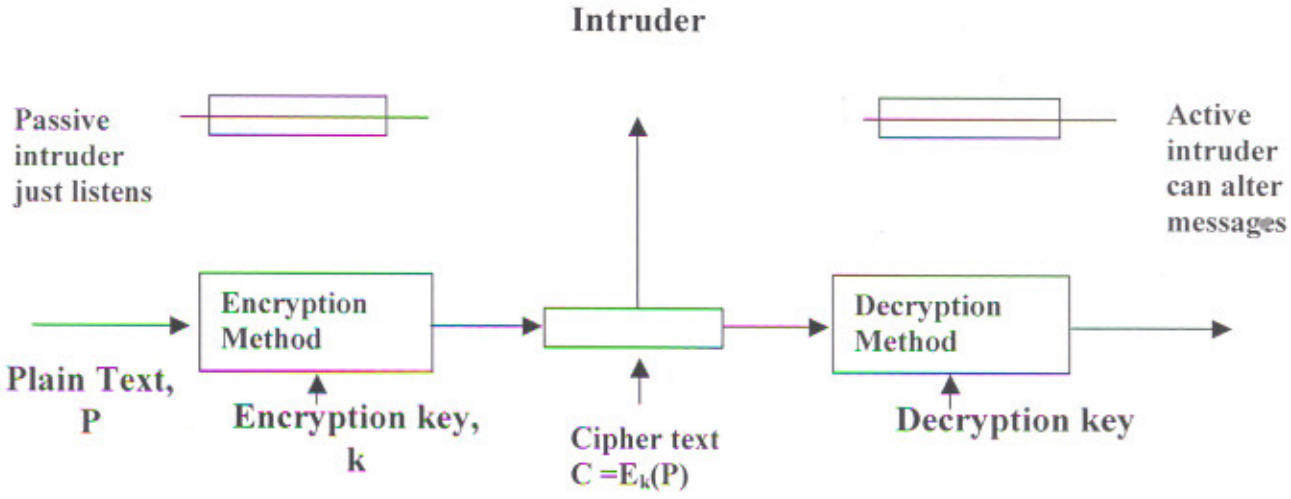

Figure 2.1: The Encryption Model

It will often be useful to have a notation for relating plaintext, ciphertext and keys. We will use $C=E_k(P)$ to mean that the encryption to the plain text P using key K gives ciphertext C. Similarly $P=D_k(C)$ represents decryption of C to get the plain text. It then follows that

$D_k(E_k(P))=P$

The notation suggests that E and D are just mathematical functions. The only tricky part is that both are functions of two parameters out of which one is the key and other is the plain text. A fundamental rule of cryptography is that one must assume that the cryptanalyst knows the general method of

encryption used. In other words the cryptanalyst knows how the encryption method E, of Figure 1 works. The amount of effort necessary to invent, test and install a new method every time the old method is compromised or thought to be compromised has always made it impractical to keep the secret, and thinking it is secret when it does more harm than good.

This is where the key enters. The key consists of a short string that selects one of many potential encryptions. In contrast to the general method, which may only be changed every few years, the key can be changed as often as required. Thus our basic model is a stable and publicly known general method parameterized by a secret and easily changed key. The non-secrecy of the algorithm cannot be emphasized enough. The real secrecy in the key and its length is a major design issue. The longer the key the higher the work factor the cryptanalyst has to deal with. The work factor for breaking the system by exhaustive search of the key space is exponential in the key length.

From the cryptanalyst's point of view, the cryptanalysis problem has three general variations. When he has quantity of ciphertext and no plaintext, he is confronted with the ciphertext problem only. The cryptograms that appear in the puzzle section of newspaper pose this kind of problem. When he has some matched ciphertext and plaintext, the problem becomes known as the known plaintext problem. Finally, when the cryptanalyst has the ability to encrypt pieces of plaintext of his own choosing we have the chosen plaintext problem.

Encryption methods have been divided into two categories: substitution ciphers and transposition ciphers. We will now deal with each of these briefly as background information for modern cryptography.

2.1 Substitution Ciphers

In a substitution cipher, each letter or groups of letters is replaced by another letter or group of letters to disguise it. One of the oldest known ciphers is the Caesar cipher, attributed to Julius Caesar. In this method, a becomes D, b becomes E, c becomes F,…, and z becomes C. For example, attack becomes DWWDFN. A slight generalization of the Caesar cipher allows the ciphertext alphabet to be shifted k letters, instead of always 3. In this case k becomes a key to the general method of circularly shifted alphabets.

The next improvement is called a monoalphabetic substitution, with key being 26–letter string corresponding to the full alphabet. For the key above, the plaintext attack would be transformed into the ciphertext QZZQEA.

At first glance this might appear to be a safe system because although the cryptanalyst knows the general system, he does not know which of the 26!=4x1026 possible keys in use

Nevertheless, given a surprisingly small amount of ciphertext, the cipher can be broken easily. The basic attack takes the advantage of the statistical properties of the natural language. In English, foe example, e is the most

common letter followed by t, o, a, n, etc. The most common two letter combinations, or digrams, are th , in, er, re, and an. The most common three letter combinations, or trigrams, are the, ing, and, and ion.

A cryptanalyst trying to break a monoalphabetic cipher would start out by counting the relative frequencies of all letters in the cipher text. Then he might tentatively assign the most common one to e and the next common one to t. he would then look at trigrams to find a common one of the form tXe, which strongly suggests that X is h. Similarly, if the pattern thYt occurs frequently, the Y probably stands for a. By making guesses at common letters, digrams and trigrams, and knowing about likely patterns of vowels and consonants, the cryptanalyst builds up a tentative plaintext, letter by letter.

So as to improve on the mono alphabetic technique is to use different mono alphabetic substitution as one proceeds through plaintext message. The general name for this approach is Poly alphabetic cipher. This technique has the following features:-

1 A set of related mono alphabetic substitution rules are used.
2 A key determines which particular rule is chosen for a given transformation.

The best-known, and one of the simplest, such algorithm is referred to as the Vigenere cipher. In this scheme, the set of related mono alphabetic substitution rules consists of the 26 Caesar ciphers, with sifts of 0 through 25. each cipher is denoted by a key letter, which is the cipher text letter that substitutes for the plaintext letter.

To aid in understanding the scheme and to aid in its use, a matrix known as the Vigenere tableau is constructed (Table 2.1). Each of the 26 ciphers is laid out horizontally, with the key letter for each cipher to its left. A normal alphabet for the plaintext runs across the top. The process of encryption is simple: Given a key letter x and plaintext y, the cipher text letter is at the intersection of the row labeled x and column labeled y: in this case the ciphertext is V.

To encrypt a message , a key is needed that is as long as the message. Usually, the key is a repeating keyword. For example, if the keyword is deceptive, the message " we are discovered save your self" is encrypted as follows:

Key:	deceptive deceptivedeceptive
Plaintext:	wearediscoveredsaveyourself
Ciphertext:	ZICVTWQNGRZGVTWAVZHCQYGLMGJ

The decryption is equally simple. The key letter again identifies the row. The position of the ciphertext in that row determines the column, and plaintext letter is at the top of that column. The strength of this cipher is that

there are multiple ciphertext letters for each plaintext letter, one for each unique letter of the keyword. Thus the letter frequency information is obscured.

2.2 Transposition Cipher

Substitution cipher preserves the order of the plaintext symbols but disguise them. Transposition cipher, in contrast, reorder the letters but do not disguise them. Figure 2.2 depicts a common transposition cipher, the columnar transposition. The cipher is keyed by a word or phrase not containing any repeated letters. In this example, MEGABUCK is the key. The purpose of the key is to number the columns, column1 being under the key letter closet to the start of the alphabet, and so on. The plaintext is written horizontally, in rows. The ciphertext is read out by columns, starting with the column whose key letter is the lowest.

M E G A B U C K	
7 4 5 1 2 8 3 6	**Plaintext**
p l e a s e t r	pleasetransferonemilliondollarstomyswiss
a n s t e r o n	bankaccountsixtwotwo
e m l l l l o n	
d o l l a r s t	**Ciphertext**
o m y s w l s s	AFLLSKSOSELAWAIATOOSSCTCLNM
b a n k a c c o	OMANTESILYNTWRNNTSOWDPAEED
u n t s l x t w	OBUOERIRICXB
o t w o a b c d	

Figure 2.2 Transposition Cipher

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

Table:2.1 Vigenere tableau

To break a transposition cipher, the cryptanalyst must be aware that he is dealing with a transposition cipher. By looking at the frequency of E, T, A, O, I, etc., it is easy to see that if they fit the normal pattern for plaintext. If so, the cipher is clearly a transposition cipher, because in such a cipher every letter represents itself. The next step is to make a guess at the number of columns. In many cases a probable word or phrase may be guessed at from the context of the message.

For example, suppose that our cryptanalyst suspected the plaintext phrase milliondollars to occur somewhere in the message. Observe that diagrams MO, IL, LL, LA IR and OS occur in the ciphertext as a result of this phase wrapping around. The ciphertext letter O follows the ciphertext letter M because they are separated in the probable phrase by a distance equal to the key length. If a key of length seven had been used, the diagrams MD,IO,LL,LL,IA,OR and NS would have occurred instead. In fact, for each key length, a different set of digrams is produced in the ciphertext.By hunting for the various possibilities, the cryptanalyst can often easily determine the key length.

The remaining steps is to order the columns. When the number of columns, k, is small, each of the k(k-1) column pairs can be examined to see if its digram frequencies match those of English plaintext. The pair with the best match is assumed to be correctly positioned. Now each remaining column is tentatively tried as the successor to this pair. The column whose digram and

trigram frequencies give the best match is tentatively assumed to be correct. The predecessor column is found in the same way. The entire process is continued until a potential ordering is found. Chances are that the plaintext will be recognizable at this point.

2.3 Conclusion

Computer networks are inherently insecure. To keep information secret it must be encrypted. The traditional method of encryption is also not proper where the emphasize is on changing the system and not key. The substitution and transposition methods are also not able to sustain the attacks and the key length could be easily detected.

The next chapter focuses on the development of encryption algorithm that provides an effective means of protection of the data

CHAPTER 3

ALGORITHM FOR DATA ENCRYPTION

The data encryption algorithm incorporates the following steps to encipher a 64-bit block of data using a 64-bit key.

1. A transposition operation referred to as the initial permutation. This transposition does not utilize 64-bit key and operates solely on 64 data bits.
2. A complex key dependent product transformation that uses block ciphering to increase the number of substitutions and recording patterns.
3. A final transposition operation refers to as the inverse initial permutation which is an actual reversal of transformation.

The above steps are summarized in Figure 3.1. The initial permutation and the inverse initial permutation are single bit transpositions. The product transformations require a more comprehensive introduction. In cryptography product transformation is the successive application of substitution and transposition ciphers

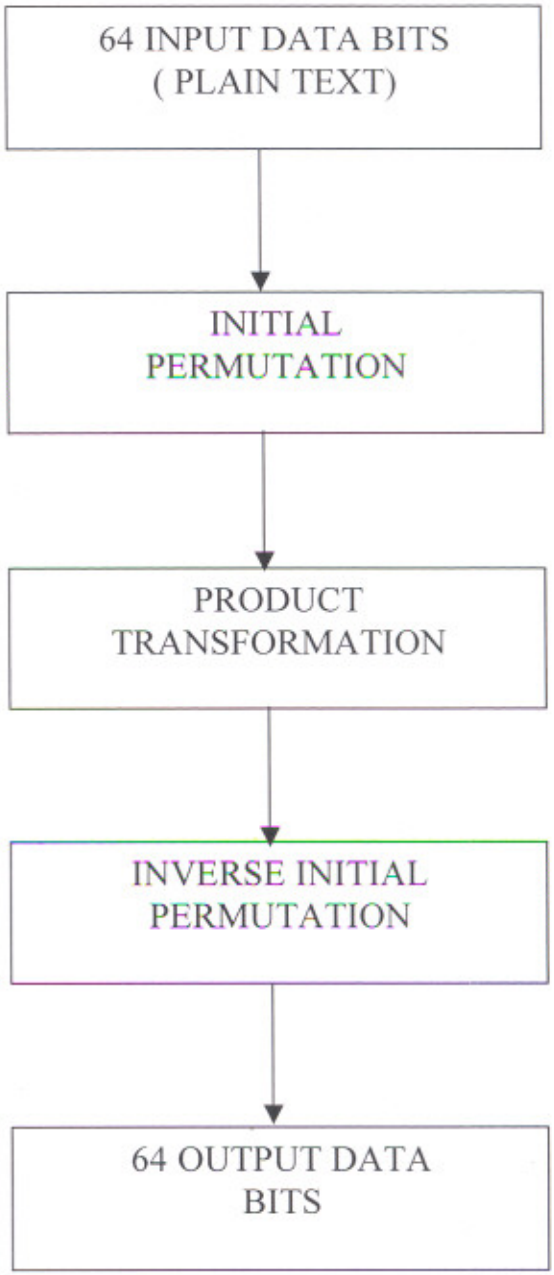

Figure 3.1: Enciphering Process

In the product transformation step of data encryption algorithm, substitution is performed under the control of a cipher key while transpositions are performed according to a fixed sequence.

Figure 3.2 depicts the product transformation, which includes the following transformation

1. The 64-bit block of plain text is divided into two 32-bit blocks denoted by L and R for left and right respectively.
2. The rightmost 32-bits of the input block becomes the leftmost 32-bits of the output block. This is denoted by figure B by an arrow going from R to L.
3. The rightmost 32-bits of the input block denoted hereafter as R go through a selection process yielding a 48-bit data block. This is a fixed selection that is not key dependent.
4. The bits of the key is added to the output of step 3 yielding a 48-bit result.
5. The 48-bit output is divided into groups each consisting of 6 bits that are each subjected to unique substitution ciphers that yield 4-bit groups data concatenated to form a 32-bit output.
6. The 32-bit output of step 5 is permuted to produce a 32-bit block,which is a simple transposition.
7. The 32-bit output of step6 is added to the leftmost 32-bits of the input block denoted by L yielding R which is the rightmost 32-bits of the 64-bit output block.

The last step in the product transformation is the block transformation of the left and right halves of the output.

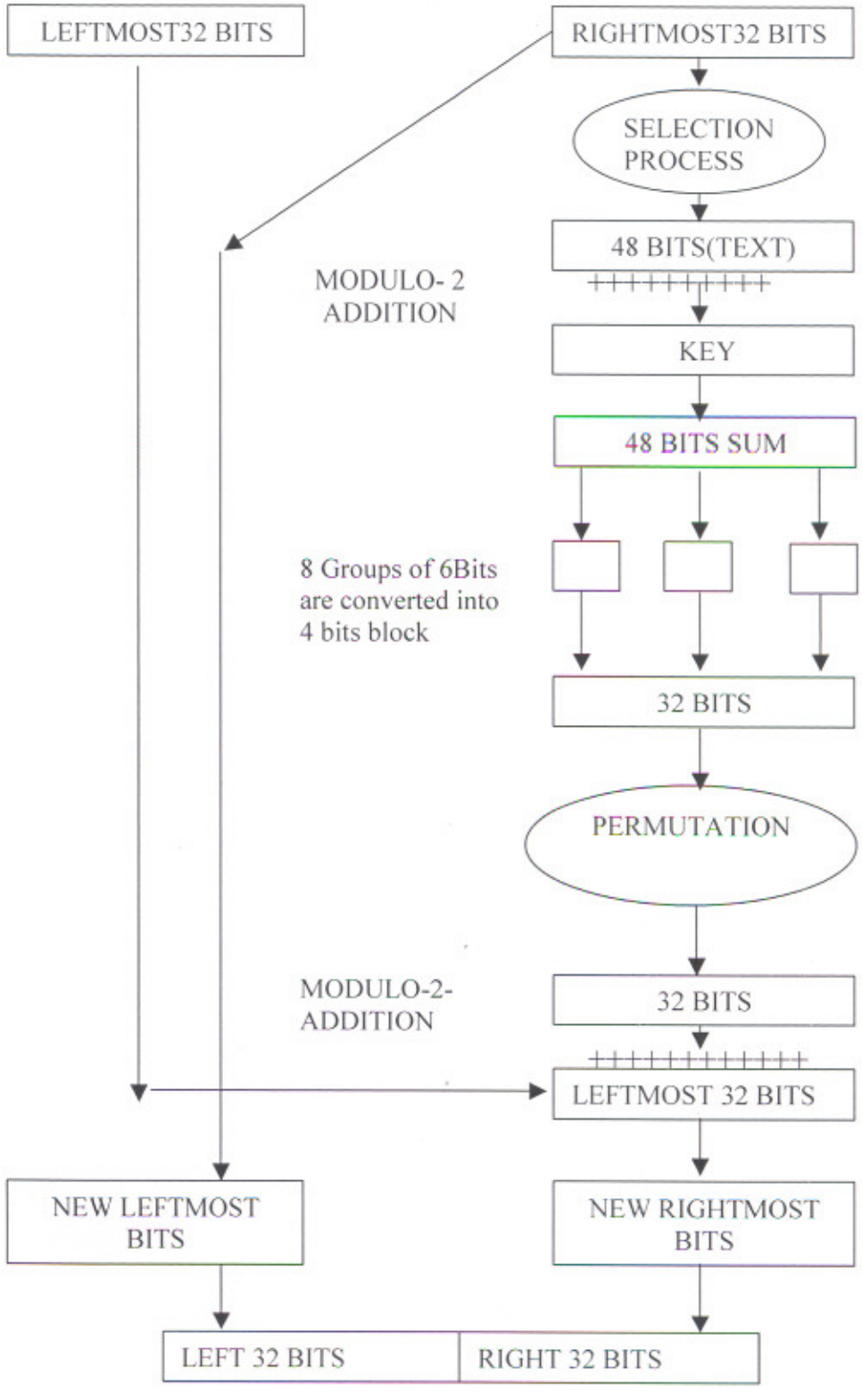

Figure 3.2:Product Transformation

3.1 COMPONENTS OF AN ALGORITHM

One means of describing the standard data encryption algorithm is to present its major components in detailed form

3.1.1 Key

Here I have developed a program that uses a 64 bit key. The key is firstly subjected to an initial permutation (Table 3.1) which is a predefined format developed by me, where emphasize had been laid on the inter mixing of the bits in such a way that the output of this program (i.e. key) is totally different from what was entered by the user. This is done so as to provide an effective protection to the key. Then the output further undergoes modulo-2 addition(Figure 3.3) and this is done in such a manner that the output is a 48 bit key, which is further added to the 48 bit sum with the help of a modulo-2 addition.

For example consider the following the 64 bit input K:

K_1	K_2	K_3	K_4	K_5	K_6	K_7	K_8
K_9	K_{10}	K_{11}	K_{12}	K_{13}	K_{14}	K_{15}	K_{16}
K_{17}	K_{18}	K_{19}	K_{20}	K_{21}	K_{22}	K_{23}	K_{24}
K_{25}	K_{26}	K_{27}	K_{28}	K_{29}	K_{30}	K_{31}	K_{32}
K_{33}	K_{34}	K_{35}	K_{36}	K_{37}	K_{38}	K_{39}	K_{40}
K_{41}	K_{42}	K_{43}	K_{44}	K_{45}	K_{46}	K_{47}	K_{48}
K_{49}	K_{50}	K_{51}	K_{52}	K_{53}	K_{54}	K_{55}	K_{56}
K_{57}	K_{58}	K_{59}	K_{60}	K_{61}	K_{62}	K_{63}	K_{64}

Where K_i is a binary digit. Then the permutation is as follows:

K_{51}	K_{10}	K_4	K_{16}	K_{61}	K_9	K_2	K_{64}
K_1	K_{35}	K_{29}	K_{13}	K_{42}	K_{37}	K_5	K_{19}
K_{12}	K_{21}	K_{54}	K_{34}	K_{59}	K_{18}	K_{46}	K_{31}
K_{63}	K_{30}	K_{17}	K_{47}	K_{38}	K_{22}	K_8	K_{40}
K_{27}	K_{45}	K_{62}	K_{20}	K_{43}	K_{50}	K_{39}	K_{25}
K_3	K_{60}	K_{23}	K_{55}	K_{33}	K_{26}	K_{52}	K_{48}
K_{32}	K_7	K_{11}	K_{44}	K_{28}	K_{56}	K_{14}	K_{57}
K_{15}	K_{53}	K_{36}	K_{24}	K_{49}	K_{58}	K_6	K_{41}

3.1.2 Modulo- 2 Addition

A bit by bit modulo 2-addition operation is used in several steps of a standard data algorithm

A bit by bit modulo 2-addition is the same as the exclusive or operation defined in boolean algebra.

3.1.3 The Cipher Function

The cipher function comprises of the main operation in the product transformation. Figure 3.4 gives an overview of the cipher function, which combines the following operations:

1. A selection operation E that operates on the argument A of 32 bits and produce a 48-bit result.
2. A modulo-2 addition, which adds the result of the selection operation E and 48-bit key on a bit by bit basis yielding a 48-bit result.
3. A unique set of selection function S that converts the 48-bit result of the modulo addition to a set of 32 bits
4. A permutation operation P that operates on the 32-bit result of the previous operation and produces a 32 bit result.

	Columns							
Rows	0	1	2	3	4	5	6	7
0	51	10	4	16	61	9	2	64
1	1	35	29	13	42	37	5	19
2	12	21	54	34	59	18	46	31
3	63	30	17	47	38	22	8	40
4	27	45	62	20	43	50	39	25
5	3	60	23	55	33	26	52	48
6	32	7	11	44	28	56	14	57
7	15	53	36	24	49	58	6	41

Table 3.1 Initial Permutation

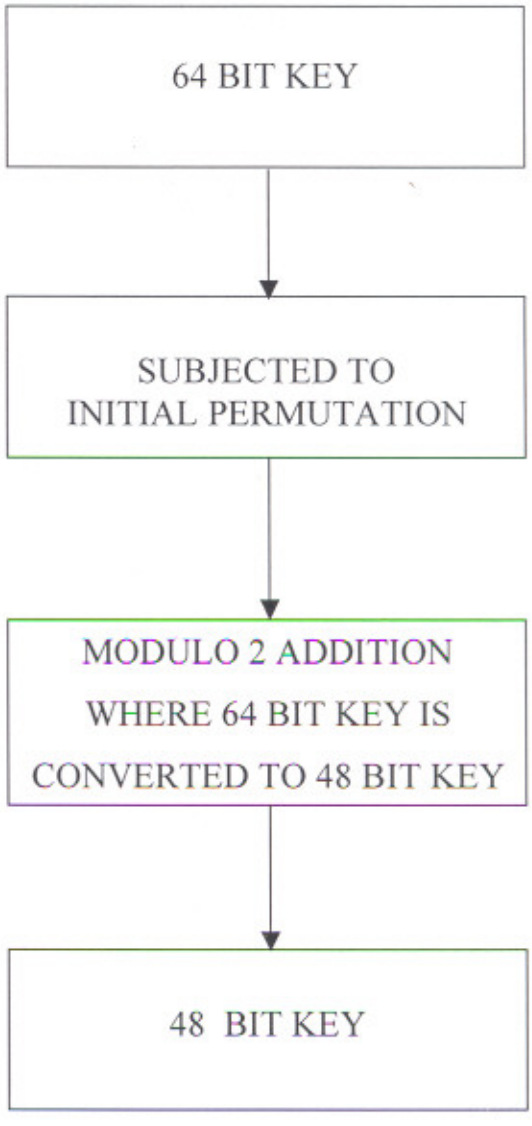

Figure 3.3: Overview of the Key Function

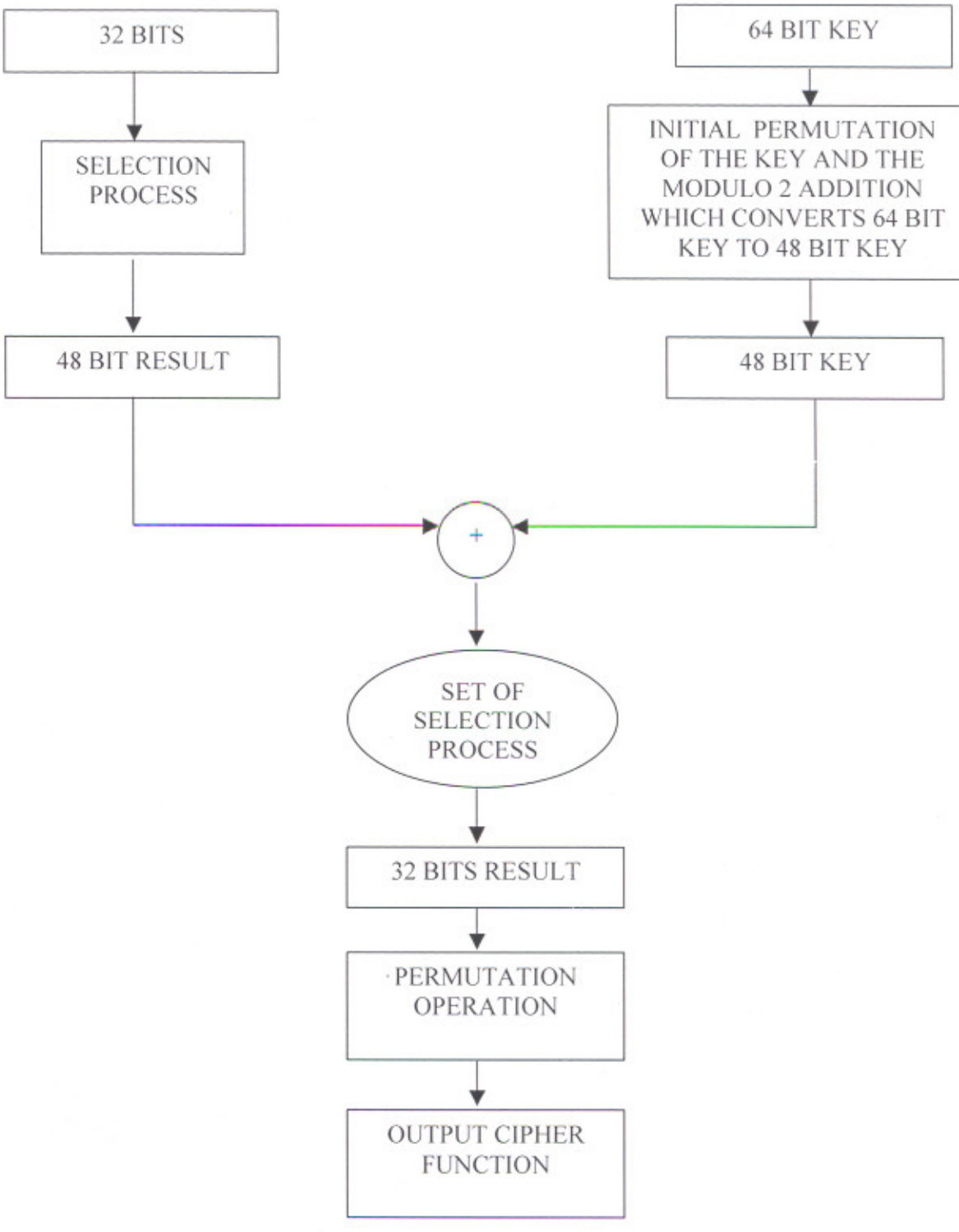

Figure 3.4: Overview of the Cipher Function

The unique set of the selection function S(Figure 3.5)takes a 6-bit block as input and yields a 4-bit result. A selection function is represented as a 4x16 matrix of numbers in Table3.2. We input to the unique set of selection function S_1 to S_8 which is a 48-bit block, denoted symbolically as B_1 to B_8.

1. The first and last bits of B represent a binary number in the range 0 to 3,denoted by 'm'.
2. The middle 4 bits of B represent a binary number in the range 0 to 15 denoted by 'n'.
3. Using zero origin indexing, the number located in the m^{th} row and n^{th} row is selected as a four bit binary block.
4. The result of step 3 is the output of selection function S.

Column Number

Row Number	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	14	4	13	1	2	15	11	8	3	10	6	12	5	9	0	7
1	0	15	7	4	14	2	13	1	10	6	12	11	9	5	3	8
2	4	1	14	8	13	6	2	11	15	12	9	7	3	10	5	0
3	15	12	8	2	4	9	1	7	5	11	3	14	10	0	6	13

Column Number

Row Number	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	15	1	8	14	6	11	3	4	9	7	2	13	12	0	5	10
1	3	13	4	7	15	2	8	14	12	0	1	10	6	9	11	5
2	0	14	7	11	10	4	13	1	5	8	12	6	9	3	2	15
3	13	8	10	1	3	15	4	2	11	6	7	12	0	5	14	9

Column Number

Row Number	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	10	0	9	14	6	3	15	5	1	13	12	7	11	4	2	8
1	13	7	0	9	3	4	6	10	2	8	5	14	12	11	15	1
2	13	6	4	9	8	15	3	0	11	1	2	12	5	10	4	7
3	1	10	13	0	6	9	8	7	4	15	14	3	11	5	2	12

Column Number

Row Number	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	7	13	14	3	0	6	9	10	1	2	8	5	11	12	4	15
1	13	8	11	5	6	15	0	3	4	7	2	12	1	10	14	9
2	10	6	9	0	12	11	7	13	15	1	3	14	5	2	8	4
3	3	15	0	6	10	1	13	8	9	4	5	11	12	7	2	14

Column Number

Row Number	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	2	12	4	1	7	10	11	6	8	5	3	15	13	0	14	9
1	14	11	2	12	4	7	13	1	5	0	15	10	3	9	8	6
2	4	2	1	11	10	13	7	8	15	9	12	5	6	3	0	14
3	11	8	12	7	1	14	2	13	6	15	0	9	10	4	5	3

Column Number

Row Number	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	12	1	10	15	9	2	6	8	0	13	3	4	14	7	5	11
1	10	15	4	2	7	12	9	5	6	1	13	14	0	11	3	8
2	9	14	15	5	2	8	12	3	7	0	4	10	1	13	11	6
3	4	3	2	12	9	5	15	10	11	14	1	7	6	0	8	13

Column Number

Row Number	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	4	11	2	14	15	0	8	13	3	12	9	7	5	10	6	1
1	13	0	11	7	4	9	1	10	14	3	5	12	2	15	8	6
2	1	4	11	13	12	3	7	14	10	15	6	8	0	5	9	2
3	6	11	13	8	1	4	10	7	7	5	0	15	14	2	3	12

Column Number

Row Number	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	13	2	8	4	6	15	11	1	10	9	3	14	5	0	12	7
1	1	15	13	8	10	3	7	4	12	5	6	11	0	14	9	2
2	7	11	4	1	9	12	14	2	0	6	10	13	15	3	5	8
3	2	1	14	7	4	10	8	13	15	12	9	0	3	5	6	11

Table 3.2: S-Boxes

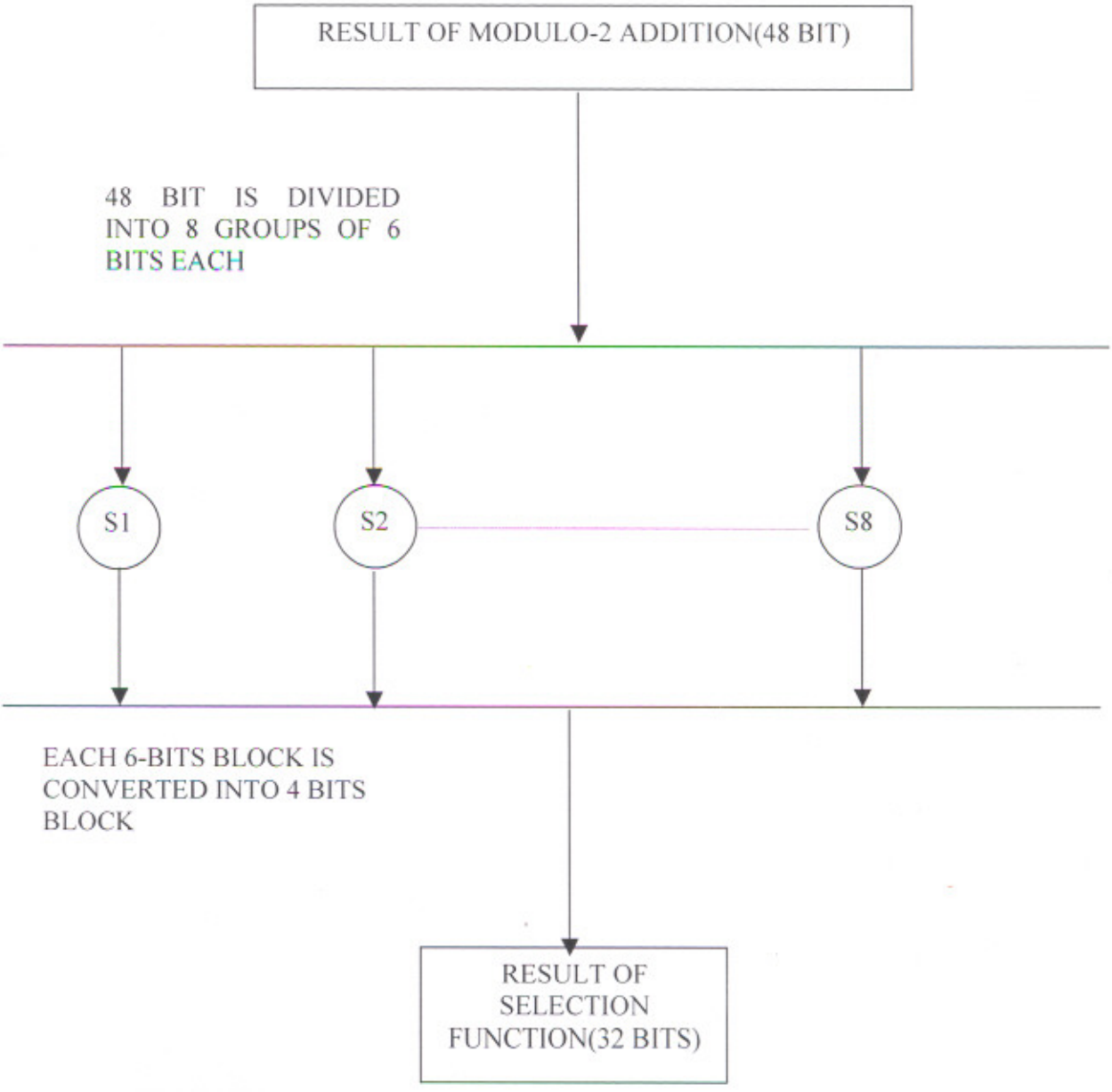

Figure 3.5: Set of Selection Functions

3.1.4 Pre-Output Block

The output of the last iteration in the product transformation goes through a block transformation yielding a 64-bit result is known as the pre-output block. The block transformation is the simple exchange of R and L as depicted in Figure3.6. The pre-output block is comprised of the bits of R followed by the bits of L and constitutes a 64-bit block, whose bits are numbered from 1 to 64 going from left to right.

3.1.5 Initial Permutation

The initial permutation is defined by Table3.3. To see how the initial permutation works , consider the following the 64 bit input M:

M_1	M_2	M_3	M_4	M_5	M_6	M_7	M_8
M_9	M_{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 M_i is a binary digit. Then the permutation 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_9	M_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

Column

Row	0	1	2	3	4	5	6	7
0	58	50	42	34	26	18	10	2
1	60	52	44	36	28	20	12	4
2	62	54	46	38	30	22	14	6
3	64	56	48	40	32	24	16	8
4	57	49	41	33	25	17	9	1
5	59	51	43	35	27	19	11	3
6	61	53	45	37	29	21	13	5
7	63	55	47	39	31	23	15	7

Table 3.3 Initial Permutation

3.1.6 Decryption

Consequently, to decipher it is necessary to apply the very same algorithm to an enciphered message block, taking care that at each iteration of the computation the same block of key bits K is used during decipherment as was used during the encipherment of the block. E.g. the permutation IP^{-1} (Table 3.4) applied to the preoutput block is the inverse of the initial permutation IP applied to the input

Columns

Rows	0	1	2	3	4	5	6	7
0	40	8	48	16	56	24	64	32
1	39	7	47	15	55	23	63	31
2	38	6	46	14	54	22	62	30
3	37	5	45	13	53	21	61	29
4	36	4	44	12	52	20	60	28
5	35	3	43	11	51	19	59	27
6	34	2	42	10	50	18	58	26
7	33	1	41	9	49	17	57	25

Table 3.4 Inverse Initial Permutation

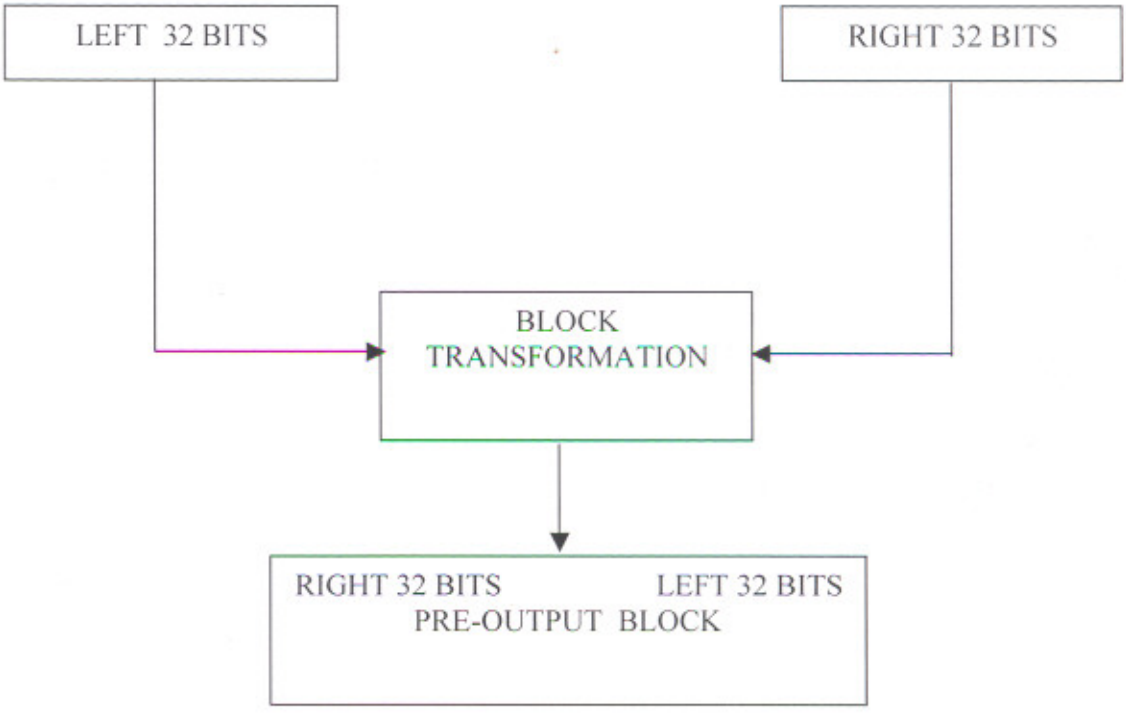

Figure 3.6: Transformation of Bits

3.2 CONCLUSION

The main objective is to evolve a system for protection of data transmitted through electronic means. Here the data which is 64 bit plaintext passes through an initial permutation that rearrange the bits to produce the permuted input. This is followed by a phase consisting of rounds, which involves both permutation and substitution functions. The output of the last round consists of 64 bits that are a function of the input plaintext and the key. Here a 64 bit key is used. Initially the key passes through an initial permutation followed by modulo- 2 addition, which produces a 48 bit key. Then the left and right halves of the output are swapped to produce pre-output. Finally, the pre-output is passed through a permutation that is the inverse of the initial permutation function, to produce a 64-bit cipher text.

CHAPTER 4

Conclusions

Due to the need to ensure that only those intended to view sensitive information can only see the information, and to ensure that the information arrives un-altered, a program have been developed where emphasizes had been made to intermix the data and the key in such a way that it becomes hard to identify the two input data. The intermixed data is subjected to various permutation and substitution functions which are key independent. The program uses a 64 bit key. The key is firstly subjected to an initial permutation and then modulo-2 addition which is a predefined format developed by me, where emphasize had been laid on the inter mixing of the bits in such a way that the output of this program (i.e. key) is totally different from what was entered by the user. This is done so as to provide an effective protection to the key.

The advantage of the program are it's key length i.e. with a 2^{64} possible keys, attacks like brute force attacks are impractical and the average time required for exhaustive key search, assuming it takes 1us will be greater than one thousand four hundred and twenty two years. Other advantage it provides is an effective encryption scheme and encryption schemes can be broken, but making them as hard as possible to break is the job that I have tried to achieve

Further Scope

As there is always a danger of new security threats, the further improvement that can be done is to further increase the length of the key, develop new permutation tables and can change the pattern of modulo-2 addition.

REFERENCES

1 Andrew S. Tanenbaum," Computer Networks",Prenctice-Hall,1998.

2 Dimitri Bertsekes and Robert Galleger, "Data Networks", Prenctice-Hall,1994.

3 Harry Katzan, "Computer Data Security", Petrocelli Book,1999.

4 John Y Hsu, "Computer Networks", Artech House,1998

5 William Stallings:" Cryptography and Network Security", Prentice Hall, Second Edition,2000.

6 http://www.itl.nist.gov

7 http://raphael.math.uic.edu

8 http://www.hem.passegn.se/tan01

9 http://searchsecurity.techtarget.com

GLOSSARY

Cipher: An algorithm for encryption and decryption . a cipher replaces a piece of information(an element in plaintext) with another object, with the intent to conceal meaning. Typically , the replacement rule is governed by a secret key.

Ciphertext: The output of an encryption algorithm; the encrypted form of a message or data.

Code: An unvarying rule for replacing a piece of information(e.g. letter, word, phrase) with another object, not necessarily of the same sort. Generraly , there is no intent to conceal meaning.

Cryptology: The study of secure communications, which encompasses both cryptography and cryptanalysis.

Cryptanalysis: The branch of cryptology dealing with the breaking of a cipher to recover information, or forging encrypted information that will be accepted as authentic.

Cryptography: The branch of cryptology dealing with design of algorithm for encryption and decryption, intended to ensure the secrecy and/or authenticity of message.

Cryptology: The study of secure communications, which encompasses both cryptography and cryptanalysis.

Decryption: The translation of encrypted text or data into original text or data.

Digram: A two letter sequence. In English and other languages, the relative frequency of various digrams in plaintext can be used in the cryptanalysis of some ciphers. Also called digraph.

Encryption: The conversion of plaintext or data into unintelligible form by means of a reversible translation, based on a translation table or algorithm. Also called enciphering.

APPENDIX

Source Code

```
#include<stdio.h>
#include<conio.h>

int bd2(int,int);
int bd4(int,int,int,int);
int db1(int);
int db2(int);
int db3(int);
int db4(int);


main()
{
clrscr();

int i,m,data[128],data1[128];
printf("Enter 128 bit Message\n");
for(i=1;i<=128;i++)
{
scanf("%d",&m);
data[i]=m;
}

int a[64],b[64];
for(i=1;i<65;i++)
{
a[i]=data[i];
}


for(i=65;i<129;i++)
{
b[i-64]=data[i];
}


/*
INTIAL PERMUTATION
*/

int a1[32],ar[32];
a1[1]=a[58];      a1[2]=a[50];      a1[3]=a[42];      a1[4]=a[34];

a1[5]=a[26];      a1[6]=a[18];      a1[7]=a[10];      a1[8]=a[2];

a1[9]=a[60];      a1[10]=a[52];     a1[11]=a[44];     a1[12]=a[36];

a1[13]=a[28];     a1[14]=a[20];     a1[15]=a[12];     a1[16]=a[4];

a1[17]=a[62];     a1[18]=a[54];     a1[19]=a[46];     a1[20]=a[38];

a1[21]=a[30];     a1[22]=a[22];     a1[23]=a[14];     a1[24]=a[6];
```

```
al[25]=a[64];     al[26]=a[56];     al[27]=a[48];     al[28]=a[40];
al[29]=a[32];     al[30]=a[24];     al[31]=a[16];     al[32]=a[8];
ar[1]=a[57];      ar[2]=a[49];      ar[3]=a[41];      ar[4]=a[33];
ar[5]=a[25];      ar[6]=a[17];      ar[7]=a[9]; ar[8]=a[1];
ar[9]=a[59];      ar[10]=a[51];     ar[11]=a[43];     ar[12]=a[35];
ar[13]=a[27];     ar[14]=a[19];     ar[15]=a[11];     ar[16]=a[3];
ar[17]=a[61];     ar[18]=a[53];     ar[19]=a[45];     ar[20]=a[37];
ar[21]=a[29];     ar[22]=a[21];     ar[23]=a[13];     ar[24]=a[5];
ar[25]=a[63];     ar[26]=a[55];     ar[27]=a[47];     ar[28]=a[39];
ar[29]=a[31];     ar[30]=a[23];     ar[31]=a[15];     ar[32]=a[7];
printf("\ntesting one%d%d",al[24],al[31]);
/*
THE 32 BITS OF RIGHT BLOCK ARE CONVERTED INTO 48 BITS
*/
int ae[48];
ae[1]=ar[32];     ae[2]=ar[1];      ae[3]=ar[2];      ae[4]=ar[3];
ae[5]=ar[4];      ae[6]=ar[5];      ae[7]=ar[4];      ae[8]=ar[5];
ae[9]=ar[6];      ae[10]=ar[7];     ae[11]=ar[8];     ae[12]=ar[9];
ae[13]=ar[8];     ae[14]=ar[9];     ae[15]=ar[10];    ae[16]=ar[11];
ae[17]=ar[12];    ae[18]=ar[13];    ae[19]=ar[12];    ae[20]=ar[13];
ae[21]=ar[14];    ae[22]=ar[15];    ae[23]=ar[16];    ae[24]=ar[17];
ae[25]=ar[16];    ae[26]=ar[17];    ae[27]=ar[18];    ae[28]=ar[19];
ae[29]=ar[20];    ae[30]=ar[21];    ae[31]=ar[20];    ae[32]=ar[21];
ae[33]=ar[22];    ae[34]=ar[23];    ae[35]=ar[24];    ae[36]=ar[25];
ae[37]=ar[24];    ae[38]=ar[25];    ae[39]=ar[26];    ae[40]=ar[27];
ae[41]=ar[28];    ae[42]=ar[29];    ae[43]=ar[28];    ae[44]=ar[29];
ae[45]=ar[30];    ae[46]=ar[31];    ae[47]=ar[32];    ae[48]=ar[1];

/*
KEY IS ENTERED HERE
*/
```

```
printf("\n");

int ak[64],ay,ap2[64],ap1[64];
printf("\nEnter the key\n");
for(i=1;i<65;i++)
{
scanf("%d",&ay);
ak[i]=ay;
}


ap2[1]=ak[51]; ap2[2]=ak[10]; ap2[3]=ak[4]; ap2[4]=ak[16];
ap2[5]=ak[61];

ap2[6]=ak[9];  ap2[7]=ak[2];  ap2[8]=ak[64];

ap2[9]=ak[1];
ap2[10]=ak[35];ap2[11]=ak[29];ap2[12]=ak[13];ap2[13]=ak[42];

ap2[14]=ak[37];ap2[15]=ak[5];ap2[16]=ak[19];

ap2[17]=ak[12];ap2[18]=ak[21];ap2[19]=ak[54];ap2[20]=ak[34];ap2[21]=ak[
59];

ap2[22]=ak[18];ap2[23]=ak[46];ap2[24]=ak[31];

ap2[25]=ak[63];ap2[26]=ak[30];ap2[27]=ak[17];ap2[28]=ak[47];ap2[29]=ak[
38];

ap2[30]=ak[22];ap2[31]=ak[8];ap2[32]=ak[40];

ap2[33]=ak[27];ap2[34]=ak[45];ap2[35]=ak[62];ap2[36]=ak[20];ap2[37]=ak[
43];

ap2[38]=ak[50];ap2[39]=ak[39];ap2[40]=ak[25];

ap2[41]=ak[3];ap2[42]=ak[60];ap2[43]=ak[23];ap2[44]=ak[55];ap2[45]=ak[3
3];

ap2[46]=ak[26];ap2[47]=ak[52];ap2[48]=ak[48];

ap2[49]=ak[32];ap2[50]=ak[7];ap2[51]=ak[11];ap2[52]=ak[44];ap2[53]=ak[2
8];

ap2[54]=ak[56];ap2[55]=ak[14];ap2[56]=ak[57];

ap2[57]=ak[15];ap2[58]=ak[53];ap2[59]=ak[36];ap2[60]=ak[24];ap2[61]=ak[
49];

ap2[62]=ak[58];ap2[63]=ak[6];ap2[64]=ak[41];
```

```
ap1[1]=ap2[51]^ap2[10];
ap1[2]=ap2[4];ap1[3]=ap2[16];ap1[4]=ap2[61]^ap2[59];

ap1[5]=ap2[9]^ap2[2];ap1[6]=ap2[64];ap1[7]=ap2[35];ap1[8]=ap2[29];

ap1[9]=ap2[17];ap1[10]=ap2[42]^ap2[37];ap1[11]=ap2[50]^ap2[26];ap1[12]=
ap2[31];

ap1[13]=ap2[15]^ap2[53];ap1[14]=ap2[19];ap1[15]=ap2[12]^ap2[21];ap1[16]
=ap2[54];

ap1[17]=ap2[5]; ap1[18]=ap2[3];ap1[19]=ap2[6]^ap2[32];
ap1[20]=ap2[11]^ap2[34];

ap1[21]=ap2[18];ap1[22]=ap2[7];
ap1[23]=ap2[27]^ap2[63];ap1[24]=ap2[36];

ap1[25]=ap2[13];ap1[26]=ap2[47];ap1[27]=ap2[44]^ap2[23];
ap1[28]=ap2[8]^ap2[43];

ap1[29]=ap2[20]; ap1[30]=ap2[62]^ap2[39];
ap1[31]=ap2[45]^ap2[24];ap1[32]=ap2[14];

ap1[33]=ap2[55];ap1[34]=ap2[28];ap1[35]=ap2[25];ap1[36]=ap2[56];

ap1[37]=ap2[1];ap1[38]=ap2[41];ap1[39]=ap2[57];ap1[40]=ap2[33];

ap1[41]=ap2[41];ap1[42]=ap2[60];ap1[43]=ap2[40];ap1[44]=ap2[52];

ap1[45]=ap2[48]^ap2[38]; ap1[46]=ap2[41]^ap2[56];
ap1[47]=ap2[49];ap1[48]=ap2[33]^ap2[58];
```

```
/*
MODULO 2 ADDITION OF KEY AND E BLOCK*/

int as[48];
for(i=1;i<49;i++)
{
as[i]=ae[i]^ap1[i];
}
```

```
/*
48 BIT S BLOCK INTO 8 BLOCKS OF 6 BITS EACH
*/
```

```
int as1[6],as2[6],as3[6],as4[6],as5[6],as6[6],as7[6],as8[6];


for(i=1;i<=6;i++)
as1[i]=as[i];


for(i=7;i<=12;i++)
as2[i-6]=as[i];


for(i=13;i<=18;i++)
as3[i-12]=as[i];


for(i=19;i<=24;i++)
as4[i-18]=as[i];


for(i=25;i<=30;i++)
as5[i-24]=as[i];

for(i=31;i<=36;i++)
as6[i-30]=as[i];

for(i=37;i<=42;i++)
as7[i-36]=as[i];

for(i=43;i<=48;i++)
as8[i-42]=as[i];

/*
S1 BLOCK IS CONVERTED INTO 4BIT
*/

int adlr=0,adlc=0,adsl=0,acs[32];
adlr=bd2(as1[1],as1[6]);
adlc=bd4(as1[2],as1[3],as1[4],as1[5]);

int sal[4][16]={
{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}
};

adsl=sal[adlr][adlc];
acs[1]=db1(adsl);
acs[2]=db2(adsl);
acs[3]=db3(adsl);
acs[4]=db4(adsl);

/*
S2 BLOCK IS CONVERTED INTO 4BIT
*/
```

```
int ad2r=0,ad2c=0;
ad2r=bd2(as2[1],as2[6]);
ad2c=bd4(as2[2],as2[3],as2[4],as2[5]);

int sa2[4][16]={
{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}
};

int  ads2=0;
ads2=sa2[ad2r][ad2c];
acs[5]=db1(ads2);
acs[6]=db2(ads2);
acs[7]=db3(ads2);
acs[8]=db4(ads2);
/*
S3 CONVERTED INTO 4BITS
*/
int ad3r=0,ad3c=0;
ad3r=bd2(as3[1],as3[6]);
ad3c=bd4(as3[2],as3[3],as3[4],as3[5]);
int sa3[4][16]={
{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}
};

int ads3=0;
ads3=sa3[ad3r][ad3c];
acs[9]=db1(ads3);
acs[10]=db2(ads3);
acs[11]=db3(ads3);
acs[12]=db4(ads3);


/* S4 IS CONVERTED TO 4 BITS
*/
int ad4r=0,ad4c=0;
ad4r=bd2(as4[1],as4[6]);
ad4c=bd4(as4[2],as4[3],as4[4],as4[5]);

int sa4[4][16]={
{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}
};

int ads4=0;
ads4=sa4[ad4r][ad4c];
acs[13]=db1(ads4);
acs[14]=db2(ads4);
```

```
acs[15]=db3(ads4);
acs[16]=db4(ads4);

/* S5 IS CONVERTED INTO 4 BITS*/

int ad5r=0,ad5c=0;
ad5r=bd2(as5[1],as5[6]);
ad5c=bd4(as5[2],as5[3],as5[4],as5[5]);
int sa5[4][16]={
{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}
};

int ads5=0;
ads5=sa5[ad5r][ad5c];
acs[17]=db1(ads5);
acs[18]=db2(ads5);
acs[19]=db3(ads5);
acs[20]=db4(ads5);

/* S6 IS CONVERTED INTO 4 BITS*/

int ad6r=0,ad6c=0;
ad6r=bd2(as6[1],as6[6]);
ad6c=bd4(as6[2],as6[3],as6[4],as6[5]);

int sa6[4][16]={
{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}
};

int ads6=0;
ads6=sa6[ad6r][ad6c];
acs[21]=db1(ads6);
acs[22]=db2(ads6);
acs[23]=db3(ads6);
acs[24]=db4(ads6);

/* S7 IS CONVERTED INTO 4 BITS */

int ad7r=0,ad7c=0;
ad7r=bd2(as7[1],as7[6]);
ad7c=bd4(as7[2],as7[3],as7[4],as7[5]);
int sa7[4][16]={
{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}
};

int ads7=0;
ads7=sa7[ad7r][ad7c];
```

```
acs[25]=db1(ads7);
acs[26]=db2(ads7);
acs[27]=db3(ads7);
acs[28]=db4(ads7);

/*
S8 IS CONVERTED INTO 8 BITS
*/
int ad8r=0,ad8c=0;
ad8r=bd2(as8[1],as8[6]);
ad8c=bd4(as8[2],as8[3],as8[4],as8[5]);

int sa8[4][16]={
{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},
};

int ads8=0;
ads8=sa8[ad8r][ad8c];
acs[29]=db1(ads8);
acs[30]=db2(ads8);
acs[31]=db3(ads8);
acs[32]=db4(ads8);
printf("\t");
printf("\nTesting....ad8r=%dad8c=%d",ad8r,ad8c);
printf("ads8=%d",ads8);
printf("\t");



/*printf("\n checking acs\n");
for(i=29;i<33;i++)
{
printf("%d",acs[i]);
printf("\t");
} */

/*
COMBINING 8 BITS OF 4 BITS TO GET 32 BITS DATA
*/

int ap[32];

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


/*
```

```
MODULO 2 ADDITION OF RIGHT AND LEFT BITS
*/
int al1[32],ar1[32];
for(i=1;i<33;i++)
{
ar1[i]=a1[i]^ap[i];
al1[i]=ar[i];
}



/*
PREOUTPUT BLOCK WHICH IS EXCHANGE OF RIGHT AND LEFT BITS
*/

int apb[64];
for(i=1;i<33;i++){
apb[i]=ar1[i];
apb[32+i]=ar[i];
}


 /*
 HERE THE OUTPUT OF CIPHER FUNCTION BLOCK IS THE INPUT TO
 THE INVERSE PERMUTATION BLOCK
 */

int aip[64];

aip[1]=apb[40];aip[2]=apb[8];aip[3]=apb[48];aip[4]=apb[16];

aip[5]=apb[56];aip[6]=apb[24];aip[7]=apb[64];aip[8]=apb[32];

aip[9]=apb[39];aip[10]=apb[7];aip[11]=apb[47];aip[12]=apb[15];

aip[13]=apb[55];aip[14]=apb[23];aip[15]=apb[63];aip[16]=apb[31];

aip[17]=apb[38];aip[18]=apb[6];aip[19]=apb[46];aip[20]=apb[14];

aip[21]=apb[54];aip[22]=apb[22];aip[23]=apb[62];aip[24]=apb[30];

aip[25]=apb[37];aip[26]=apb[5];aip[27]=apb[45];aip[28]=apb[13];

aip[29]=apb[53];aip[30]=apb[21];aip[31]=apb[61];aip[32]=apb[29];

aip[33]=apb[36];aip[34]=apb[4];aip[35]=apb[44];aip[36]=apb[12];

aip[37]=apb[52];aip[38]=apb[20];aip[39]=apb[60];aip[40]=apb[28];

aip[41]=apb[35];aip[42]=apb[3];aip[43]=apb[43];aip[44]=apb[11];

aip[45]=apb[51];aip[46]=apb[19];aip[47]=apb[59];aip[48]=apb[27];

aip[49]=apb[34];aip[50]=apb[2];aip[51]=apb[42];aip[52]=apb[10];

aip[53]=apb[50];aip[54]=apb[18];aip[55]=apb[58];aip[56]=apb[26];
```

```
aip[57]=apb[33];aip[58]=apb[1];aip[59]=apb[41];aip[60]=apb[9];

aip[61]=apb[49];aip[62]=apb[17];aip[63]=apb[57];aip[64]=apb[25];


printf("\n Cipher function of 64 bit data \n");
for(i=1;i<65;i++)
{
printf("%d",aip[i]);
printf("\t");
}


int b1[32],br[32];

b1[1]=b[58];      b1[2]=b[50];      b1[3]=b[42];      b1[4]=b[34];

b1[5]=b[26];      b1[6]=b[18];      b1[7]=b[10];      b1[8]=b[2];

b1[9]=b[60];      b1[10]=b[52];     b1[11]=b[44];     b1[12]=b[36];

b1[13]=b[28];     b1[14]=b[20];     b1[15]=b[12];     b1[16]=b[4];

b1[17]=b[62];     b1[18]=b[54];     b1[19]=b[46];     b1[20]=b[38];

b1[21]=b[30];     b1[22]=b[22];     b1[23]=b[14];     b1[24]=b[6];

b1[25]=b[64];     b1[26]=b[56];     b1[27]=b[48];     b1[28]=b[40];

b1[29]=b[32];     b1[30]=b[24];     b1[31]=b[16];     b1[32]=b[8];

br[1]=b[57];      br[2]=b[49];      br[3]=b[41];      br[4]=b[33];

br[5]=b[25];      br[6]=b[17];      br[7]=b[9]; br[8]=b[1];

br[9]=b[59];      br[10]=b[51];     br[11]=b[43];     br[12]=b[35];

br[13]=b[27];     br[14]=b[19];     br[15]=b[11];     br[16]=b[3];

br[17]=b[61];     br[18]=b[53];     br[19]=b[45];     br[20]=b[37];

br[21]=b[29];     br[22]=b[21];     br[23]=b[13];     br[24]=b[5];

br[25]=b[63];     br[26]=b[55];     br[27]=b[47];     br[28]=b[39];

br[29]=b[31];     br[30]=b[23];     br[31]=b[15];     br[32]=b[7];
/*
32 BITS OF RIGHT BLOCK GOES THROUGH THE CIPHER FUNCTION WHICH CONVERTS
32 BITS INTO 48 BITS
*/

int be[48];

be[1]=br[32];     be[2]=br[1];      be[3]=br[2];      be[4]=br[3];
```

```
be[5]=br[4];      be[6]=br[5];      be[7]=br[4];      be[8]=br[5];

be[9]=br[6];      be[10]=br[7];     be[11]=br[8];     be[12]=br[9];

be[13]=br[8];     be[14]=br[9];     be[15]=br[10];    be[16]=br[11];

be[17]=br[12];    be[18]=br[13];    be[19]=br[12];    be[20]=br[13];

be[21]=br[14];    be[22]=br[15];    be[23]=br[16];    be[24]=br[17];

be[25]=br[16];    be[26]=br[17];    be[27]=br[18];    be[28]=br[19];

be[29]=br[20];    be[30]=br[21];    be[31]=br[20];    be[32]=br[21];

be[33]=br[22];    be[34]=br[23];    be[35]=br[24];    be[36]=br[25];

be[37]=br[24];    be[38]=br[25];    be[39]=br[26];    be[40]=br[27];

be[41]=br[28];    be[42]=br[29];    be[43]=br[28];    be[44]=br[29];

be[45]=br[30];    be[46]=br[31];    be[47]=br[32];    be[48]=br[1];


/*
KEY
*/

printf("\n");
int bk[64],by,bp2[64],bp1[48];
printf("\n Enter the key for b\n");
for(i=1;i<65;i++)
{
scanf("%d",&by);
bk[i]=by;
}

bp2[1]=bk[51]; bp2[2]=bk[10]; bp2[3]=bk[4];     bp2[4]=bk[16];
      bp2[5]=bk[61];

bp2[6]=bk[9];  bp2[7]=bk[2];  bp2[8]=bk[64];

bp2[9]=bk[1];
bp2[10]=bk[35];bp2[11]=bk[29];bp2[12]=bk[13];bp2[13]=bk[42];

bp2[14]=bk[37]; bp2[15]=bk[5];bp2[16]=bk[19];

bp2[17]=bk[12];
bp2[18]=bk[21];bp2[19]=bk[54];bp2[20]=bk[34];bp2[21]=bk[59];

bp2[22]=bk[18]; bp2[23]=bk[46];bp2[24]=bk[31];

bp2[25]=bk[63];
bp2[26]=bk[30];bp2[27]=bk[17];bp2[28]=bk[47];bp2[29]=bk[38];

bp2[30]=bk[22];bp2[31]=bk[8];bp2[32]=bk[40];
```

```
bp2[33]=bk[27];bp2[34]=bk[45];bp2[35]=bk[62];bp2[36]=bk[20];bp2[37]=bk[
43];

bp2[38]=bk[50];bp2[39]=bk[39];bp2[40]=bk[25];

bp2[41]=bk[30];bp2[42]=bk[60];bp2[43]=bk[23];bp2[44]=bk[55];bp2[45]=bk[
33];

bp2[46]=bk[26];bp2[47]=bk[52];bp2[48]=bk[48];

bp2[49]=bk[32];bp2[50]=bk[7];bp2[51]=bk[11];bp2[52]=bk[44];bp2[53]=bk[2
8];

bp2[54]=bk[56]; bp2[55]=bk[14];bp2[56]=bk[57];

bp2[57]=bk[15];bp2[58]=bk[53];bp2[59]=bk[36];bp2[60]=bk[24];bp2[61]=bk[
49];

bp2[62]=bk[58]; bp2[63]=bk[6];bp2[64]=bk[41];


bp1[1]=bp2[51]^bp2[10];
bp1[2]=bp2[4];bp1[3]=bp2[16];bp1[4]=bp2[61]^bp2[59];

bp1[5]=bp2[9]^bp2[2];bp1[6]=bp2[64];bp1[7]=bp2[35];bp1[8]=bp2[29];

bp1[9]=bp2[17];bp1[10]=bp2[42]^bp2[37];bp1[11]=bp2[50]^bp2[26];bp1[12]=
bp2[31];

bp1[13]=bp2[15]^bp2[53];bp1[14]=bp2[19];bp1[15]=bp2[12]^bp2[21];bp1[16]
=bp2[54];

bp1[17]=bp2[5]; bp1[18]=bp2[3];bp1[19]=bp2[6]^bp2[32];
bp1[20]=bp2[11]^bp2[34];

bp1[21]=bp2[18];bp1[22]=bp2[7];bp1[23]=bp2[27]^bp2[63];bp1[24]=bp2[36];

bp1[25]=bp2[13];bp1[26]=bp2[47];bp1[27]=bp2[44]^bp2[23];bp1[28]=bp2[8]^
bp2[43];

bp1[29]=bp2[20];bp1[30]=bp2[62]^bp2[39];bp1[31]=bp2[45]^bp2[24];bp1[32]
=bp2[14];

bp1[33]=bp2[55];bp1[34]=bp2[28];bp1[35]=bp2[25];bp1[36]=bp2[56];

bp1[37]=bp2[1];bp1[38]=bp2[41];bp1[39]=bp2[57];bp1[40]=bp2[33];

bp1[41]=bp2[41];bp1[42]=bp2[60];bp1[43]=bp2[40];bp1[44]=bp2[52];

bp1[45]=bp2[48]^bp2[38]; bp1[46]=bp2[41]^bp2[56];
bp1[47]=bp2[49];bp1[48]=bp2[33]^bp2[58];


/* MODULO 2 ADDITION OF KEY AND E BLOCK*/
int bs[48];
```

```
for(i=1;i<49;i++)
{
bs[i]=be[i]^bp1[i];
}

/* 48 BIT S BLOCK IS CONVERTED INTO 8 BLOCKS OF 6 BITS*/


int bs1[6],bs2[6],bs3[6],bs4[6],bs5[6],bs6[6],bs7[6],bs8[6];


for(i=1;i<=6;i++)
bs1[i]=bs[i];


for(i=7;i<=12;i++)
bs2[i-6]=bs[i];


for(i=13;i<=18;i++)
bs3[i-12]=bs[i];


for(i=19;i<=24;i++)
bs4[i-18]=bs[i];


for(i=25;i<=30;i++)
bs5[i-24]=bs[i];

for(i=31;i<=36;i++)
bs6[i-30]=bs[i];

for(i=37;i<=42;i++)
bs7[i-36]=bs[i];

for(i=43;i<=48;i++)
bs8[i-42]=bs[i];


/*
S1 BLOCK IS CONVERTED INTO 4BIT
*/

int bdlr,bdlc,bds1,bcs[32];
bdlr=bd2(bs1[1],bs1[6]);
bdlc=bd4(bs1[2],bs1[3],bs1[4],bs1[5]);


bds1=sa1[bdlr][bdlc];
bcs[1]=db1(bds1);
bcs[2]=db2(bds1);
bcs[3]=db3(bds1);
bcs[4]=db4(bds1);

/*
```

```
S2 block is converted into 4bit
*/

int bd2r,bd2c;
bd2r=bd2(bs2[1],bs2[6]);
bd2c=bd4(bs2[2],bs2[3],bs2[4],bs2[5]);


int  bds2;
bds2=sa2[bd2r][bd2c];
bcs[5]=db1(bds2);
bcs[6]=db2(bds2);
bcs[7]=db3(bds2);
bcs[8]=db4(bds2);
/*
S3 CONVERTED INTO 4BITS
*/


int bd3r,bd3c;
bd3r=bd2(bs3[1],bs3[6]);
bd3c=bd4(bs3[2],bs3[3],bs3[4],bs3[5]);

int bds3;
bds3=sa3[bd3r][bd3c];
bcs[9]=db1(bds3);
bcs[10]=db2(bds3);
bcs[11]=db3(bds3);
bcs[12]=db4(bds3);


/*
S4 IS CONVERTED TO 4 BITS
*/
int bd4r,bd4c;
bd4r=bd2(bs4[1],bs4[6]);
bd4c=bd4(bs4[2],bs4[3],bs4[4],bs4[5]);

int bds4;
bds4=sa4[bd4r][bd4c];
bcs[13]=db1(bds4);
bcs[14]=db2(bds4);
bcs[15]=db3(bds4);
bcs[16]=db4(bds4);

/* S5 IS CONVERTED INTO 4 BITS*/

int bd5r,bd5c;
bd5r=bd2(bs5[1],bs5[6]);
bd5c=bd4(bs5[2],bs5[3],bs5[4],bs5[5]);

int bds5;
bds5=sa5[bd5r][bd5c];
bcs[17]=db1(bds5);
bcs[18]=db2(bds5);
bcs[19]=db3(bds5);
```

```
bcs[20]=db4(bds5);

/* S6 IS CONVERTED INTO 4 BITS*/

int bd6r,bd6c;
bd6r=bd2(bs6[1],bs6[6]);
bd6c=bd4(bs6[2],bs6[3],bs6[4],bs6[5]);


int bds6;
bds6=sa6[bd6r][bd6c];
bcs[21]=db1(bds6);
bcs[22]=db2(bds6);
bcs[23]=db3(bds6);
bcs[24]=db4(bds6);

/* S7 IS CONVERTED INTO 4 BITS */

int bd7r,bd7c;
bd7r=bd2(bs7[1],bs7[6]);
bd7c=bd4(bs7[2],bs7[3],bs7[4],bs7[5]);

int bds7;
bds7=sa7[bd7r][bd7c];
bcs[25]=db1(bds7);
bcs[26]=db2(bds7);
bcs[27]=db3(bds7);
bcs[28]=db4(bds7);

/*
S8 IS CONVERTED INTO 8 BITS
*/
int bd8r=0,bd8c=0;
bd8r=bd2(bs8[1],bs8[6]);
bd8c=bd4(bs8[2],bs8[3],bs8[4],bs8[5]);

int bds8;
bds8=sa8[bd8r][bd8c];
bcs[29]=db1(bds8);
bcs[30]=db2(bds8);
bcs[31]=db3(bds8);
bcs[32]=db4(bds8);

/*
COMBINING 8 BLOCKS OF 4 BIT DATA TO GET 32 BIT DATA
*/

int bp[32];

bp[1]=bcs[16];bp[2]=bcs[7];bp[3]=bcs[20];bp[4]=bcs[21];

bp[5]=bcs[29];bp[6]=bcs[12];bp[7]=bcs[28];bp[8]=bcs[17];

bp[9]=bcs[1];bp[10]=bcs[15];bp[11]=bcs[23];bp[12]=bcs[26];

bp[13]=bcs[5];bp[14]=bcs[18];bp[15]=bcs[31];bp[16]=bcs[10];
```

```
bp[17]=bcs[2];bp[18]=bcs[8];bp[19]=bcs[24];bp[20]=bcs[14];

bp[21]=bcs[32];bp[22]=bcs[27];bp[23]=bcs[3];bp[24]=bcs[9];

bp[25]=bcs[19];bp[26]=bcs[13];bp[27]=bcs[30];bp[28]=bcs[6];

bp[29]=bcs[22];bp[30]=bcs[11];bp[31]=bcs[4];bp[32]=bcs[25];

/*
MODULO 2 ADDITION OF RIGHT AND LEFT BITS
*/
int bl1[32],br1[32];

for(i=1;i<33;i++)
{
br1[i]=bl[i]^bp[i];
bl1[i]=br[i];
}

/*
PREOUTPUT BLOCK WHICH IS EXCHANGE OF RIGHT AND LEFT BITS
*/

int bpb[64];
for(i=1;i<33;i++){
bpb[i]=br1[i];
bpb[32+i]=br[i];
}


 /*
 THE OUTPUT OF CIPHER FUNCTION BLOCK IS THE INPUT TO
 THE INVERSE PERMUTATION BLOCK
 */

int bip[64];

bip[1]=bpb[40];bip[2]=bpb[8];bip[3]=bpb[48];bip[4]=bpb[16];

bip[5]=bpb[56];bip[6]=bpb[24];bip[7]=bpb[64];bip[8]=bpb[32];

bip[9]=bpb[39];bip[10]=bpb[7];bip[11]=bpb[47];bip[12]=bpb[15];

bip[13]=bpb[55];bip[14]=bpb[23];bip[15]=bpb[63];bip[16]=bpb[31];

bip[17]=bpb[38];bip[18]=bpb[6];bip[19]=bpb[46];bip[20]=bpb[14];

bip[21]=bpb[54];bip[22]=bpb[22];bip[23]=bpb[62];bip[24]=bpb[30];

bip[25]=bpb[37];bip[26]=bpb[5];bip[27]=bpb[45];bip[28]=bpb[13];

bip[29]=bpb[53];bip[30]=bpb[21];bip[31]=bpb[61];bip[32]=bpb[29];

bip[33]=bpb[36];bip[34]=bpb[4];bip[35]=bpb[44];bip[36]=bpb[12];
```

```
bip[37]=bpb[52];bip[38]=bpb[20];bip[39]=bpb[60];bip[40]=bpb[28];

bip[41]=bpb[35];bip[42]=bpb[3];bip[43]=bpb[43];bip[44]=bpb[11];

bip[45]=bpb[51];bip[46]=bpb[19];bip[47]=bpb[59];bip[48]=bpb[27];

bip[49]=bpb[34];bip[50]=bpb[2];bip[51]=bpb[42];bip[52]=bpb[10];

bip[53]=bpb[50];bip[54]=bpb[18];bip[55]=bpb[58];bip[56]=bpb[26];

bip[57]=bpb[33];bip[58]=bpb[1];bip[59]=bpb[41];bip[60]=bpb[9];

bip[61]=bpb[49];bip[62]=bpb[17];bip[63]=bpb[57];bip[64]=bpb[25];

printf("\n the cipher function of 64 bit data of b \n");
for(i=1;i<65;i++)
{
printf("%d",bip[i]);
printf("\t");
}

/* THE DICPHER FUNCTION */

for(i=1;i<65;i++)
{
a[i]=aip[i];
b[i]=bip[i];
}


al[1]=a[58];      al[2]=a[50];      al[3]=a[42];      al[4]=a[34];

al[5]=a[26];      al[6]=a[18];      al[7]=a[10];      al[8]=a[2];

al[9]=a[60];      al[10]=a[52];     al[11]=a[44];     al[12]=a[36];

al[13]=a[28];     al[14]=a[20];     al[15]=a[12];     al[16]=a[4];

al[17]=a[62];     al[18]=a[54];     al[19]=a[46];     al[20]=a[38];

al[21]=a[30];     al[22]=a[22];     al[23]=a[14];     al[24]=a[6];

al[25]=a[64];     al[26]=a[56];     al[27]=a[48];     al[28]=a[40];

al[29]=a[32];     al[30]=a[24];     al[31]=a[16];     al[32]=a[8];

ar[1]=a[57];      ar[2]=a[49];      ar[3]=a[41];      ar[4]=a[33];

ar[5]=a[25];      ar[6]=a[17];      ar[7]=a[9]; ar[8]=a[1];

ar[9]=a[59];      ar[10]=a[51];     ar[11]=a[43];     ar[12]=a[35];

ar[13]=a[27];     ar[14]=a[19];     ar[15]=a[11];     ar[16]=a[3];

ar[17]=a[61];     ar[18]=a[53];     ar[19]=a[45];     ar[20]=a[37];
```

```
ar[21]=a[29];     ar[22]=a[21];     ar[23]=a[13];     ar[24]=a[5];
ar[25]=a[63];     ar[26]=a[55];     ar[27]=a[47];     ar[28]=a[39];
ar[29]=a[31];     ar[30]=a[23];     ar[31]=a[15];     ar[32]=a[7];
/* 32 bits are converted into 48 bits*/


ae[1]=ar[32];     ae[2]=ar[1];      ae[3]=ar[2];      ae[4]=ar[3];
ae[5]=ar[4];      ae[6]=ar[5];      ae[7]=ar[4];      ae[8]=ar[5];
ae[9]=ar[6];      ae[10]=ar[7];     ae[11]=ar[8];     ae[12]=ar[9];
ae[13]=ar[8];     ae[14]=ar[9];     ae[15]=ar[10];    ae[16]=ar[11];
ae[17]=ar[12];    ae[18]=ar[13];    ae[19]=ar[12];    ae[20]=ar[13];
ae[21]=ar[14];    ae[22]=ar[15];    ae[23]=ar[16];    ae[24]=ar[17];
ae[25]=ar[16];    ae[26]=ar[17];    ae[27]=ar[18];    ae[28]=ar[19];
ae[29]=ar[20];    ae[30]=ar[21];    ae[31]=ar[20];    ae[32]=ar[21];
ae[33]=ar[22];    ae[34]=ar[23];    ae[35]=ar[24];    ae[36]=ar[25];
ae[37]=ar[24];    ae[38]=ar[25];    ae[39]=ar[26];    ae[40]=ar[27];
ae[41]=ar[28];    ae[42]=ar[29];    ae[43]=ar[28];    ae[44]=ar[29];
ae[45]=ar[30];    ae[46]=ar[31];    ae[47]=ar[32];    ae[48]=ar[1];


/*KEY IS TAKEN */
printf("\n");
printf("\nEnter the key for a same as a prev\n");

for(i=1;i<65;i++){
scanf("%d",&ay);
ak[i]=ay;
}
ap2[1]=ak[51]; ap2[2]=ak[10]; ap2[3]=ak[4]; ap2[4]=ak[16];
ap2[5]=ak[61];

ap2[6]=ak[9];  ap2[7]=ak[2];  ap2[8]=ak[64];

ap2[9]=ak[1];
ap2[10]=ak[35];ap2[11]=ak[29];ap2[12]=ak[13];ap2[13]=ak[42];

ap2[14]=ak[37];ap2[15]=ak[5];ap2[16]=ak[19];

ap2[17]=ak[12];ap2[18]=ak[21];ap2[19]=ak[54];ap2[20]=ak[34];ap2[21]=ak[
59];
```

```
ap2[22]=ak[18];ap2[23]=ak[46];ap2[24]=ak[31];

ap2[25]=ak[63];ap2[26]=ak[30];ap2[27]=ak[17];ap2[28]=ak[47];ap2[29]=ak[
38];

ap2[30]=ak[22];ap2[31]=ak[8];ap2[32]=ak[40];

ap2[33]=ak[27];ap2[34]=ak[45];ap2[35]=ak[62];ap2[36]=ak[20];ap2[37]=ak[
43];

ap2[38]=ak[50];ap2[39]=ak[39];ap2[40]=ak[25];

ap2[41]=ak[3];ap2[42]=ak[60];ap2[43]=ak[23];ap2[44]=ak[55];ap2[45]=ak[3
3];

ap2[46]=ak[26];ap2[47]=ak[52];ap2[48]=ak[48];

ap2[49]=ak[32];ap2[50]=ak[7];ap2[51]=ak[11];ap2[52]=ak[44];ap2[53]=ak[2
8];

ap2[54]=ak[56];ap2[55]=ak[14];ap2[56]=ak[57];

ap2[57]=ak[15];ap2[58]=ak[53];ap2[59]=ak[36];ap2[60]=ak[24];ap2[61]=ak[
49];

ap2[62]=ak[58];ap2[63]=ak[6];ap2[64]=ak[41];



ap1[1]=ap2[51]^ap2[10];
ap1[2]=ap2[4];ap1[3]=ap2[16];ap1[4]=ap2[61]^ap2[59];

ap1[5]=ap2[9]^ap2[2];ap1[6]=ap2[64];ap1[7]=ap2[35];ap1[8]=ap2[29];

ap1[9]=ap2[17];ap1[10]=ap2[42]^ap2[37];ap1[11]=ap2[50]^ap2[26];ap1[12]=
ap2[31];

ap1[13]=ap2[15]^ap2[53];ap1[14]=ap2[19];ap1[15]=ap2[12]^ap2[21];ap1[16]
=ap2[54];

ap1[17]=ap2[5]; ap1[18]=ap2[3];ap1[19]=ap2[6]^ap2[32];
ap1[20]=ap2[11]^ap2[34];

ap1[21]=ap2[18];ap1[22]=ap2[7];
ap1[23]=ap2[27]^ap2[63];ap1[24]=ap2[36];

ap1[25]=ap2[13];ap1[26]=ap2[47];ap1[27]=ap2[44]^ap2[23];
ap1[28]=ap2[8]^ap2[43];

ap1[29]=ap2[20]; ap1[30]=ap2[62]^ap2[39];
ap1[31]=ap2[45]^ap2[24];ap1[32]=ap2[14];

ap1[33]=ap2[55];ap1[34]=ap2[28];ap1[35]=ap2[25];ap1[36]=ap2[56];
```

```
ap1[37]=ap2[1];ap1[38]=ap2[41];ap1[39]=ap2[57];ap1[40]=ap2[33];

ap1[41]=ap2[41];ap1[42]=ap2[60];ap1[43]=ap2[40];ap1[44]=ap2[52];

ap1[45]=ap2[48]^ap2[38]; ap1[46]=ap2[41]^ap2[56];
ap1[47]=ap2[49];ap1[48]=ap2[33]^ap2[58];



for(i=1;i<49;i++)
{
as[i]=ae[i]^ap1[i];
}


/* now 48 bit s block into 8 blocks of 6 bits each
*/



for(i=1;i<=6;i++)
as1[i]=as[i];


for(i=7;i<=12;i++)
as2[i-6]=as[i];


for(i=13;i<=18;i++)
as3[i-12]=as[i];


for(i=19;i<=24;i++)
as4[i-18]=as[i];


for(i=25;i<=30;i++)
as5[i-24]=as[i];

for(i=31;i<=36;i++)
as6[i-30]=as[i];

for(i=37;i<=42;i++)
as7[i-36]=as[i];

for(i=43;i<=48;i++)
as8[i-42]=as[i];

/*
s1 block is converted into 4bit
*/
adlr=0,adlc=0,ads1=0,acs[4]=0;

adlr=bd2(as1[1],as1[6]);
adlc=bd4(as1[2],as1[3],as1[4],as1[5]);
```

```
ads1=sa1[adlr][adlc];
acs[1]=db1(ads1);
acs[2]=db2(ads1);
acs[3]=db3(ads1);
acs[4]=db4(ads1);

printf("\ntest......acs[4]%d",acs[4]);
/*
s2 block is converted into 4bit
*/

ad2r=0,ad2c=0,ads2=0;
ad2r=bd2(as2[1],as2[6]);
ad2c=bd4(as2[2],as2[3],as2[4],as2[5]);
ads2=sa2[ad2r][ad2c];
acs[5]=db1(ads2);
acs[6]=db2(ads2);
acs[7]=db3(ads2);
acs[8]=db4(ads2);

/*
s3 converted into 4bits
*/
/*int s3[4][16]={
{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}
};
  */



ad3r=0,ad3c=0,ads3=0;
ad3r=bd2(as3[1],as3[6]);
ad3c=bd4(as3[2],as3[3],as3[4],as3[5]);
ads3=sa3[ad3r][ad3c];
acs[9]=db1(ads3);
acs[10]=db2(ads3);
acs[11]=db3(ads3);
acs[12]=db4(ads3);


/* s4 is converted to 4 bits
*/

int s4[4][16]={
{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}
};

ad4r=0,ad4c=0,ads4=0;
```

```
ad4r=bd2(as4[1],as4[6]);
ad4c=bd4(as4[2],as4[3],as4[4],as4[5]);
ads4=sa4[ad4r][ad4c];
acs[13]=db1(ads4);
acs[14]=db2(ads4);
acs[15]=db3(ads4);
acs[16]=db4(ads4);

//printf("\n testing %d",acs[16]);
/* s5 is converted into 4 bits*/

int s5[4][16]={
{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}
};



ad5r=0,ad5c=0,ads5=0;
ad5r=bd2(as5[1],as5[6]);
ad5c=bd4(as5[2],as5[3],as5[4],as5[5]);
ads5=sa5[ad5r][ad5c];
acs[17]=db1(ads5);
acs[18]=db2(ads5);
acs[19]=db3(ads5);
acs[20]=db4(ads5);

/* s6 is converted into 4 bits*/



ad6r=0,ad6c=0,ads6=0;
ad6r=bd2(as6[1],as6[6]);
ad6c=bd4(as6[2],as6[3],as6[4],as6[5]);
ads6=sa6[ad6r][ad6c];
acs[21]=db1(ads6);
acs[22]=db2(ads6);
acs[23]=db3(ads6);
acs[24]=db4(ads6);

/* s7 is converted into 4 bits */
ad7r=0,ad7c=0,ads7=0;

ad7r=bd2(as7[1],as7[6]);
ad7c=bd4(as7[2],as7[3],as7[4],as7[5]);
ads7=sa7[ad7r][ad7c];
acs[25]=db1(ads7);
acs[26]=db2(ads7);
acs[27]=db3(ads7);
acs[28]=db4(ads7);

/*
s8 is converted into 8 bits
```

```
*/

ad8r=0,ad8c=0,ads8=0;

int saa8[4][16]={
{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},
};




ad8r=bd2(as8[1],as8[6]);
ad8c=bd4(as8[2],as8[3],as8[4],as8[5]);
ads8=sa8[ad8r][ad8c];
acs[29]=db1(ads8);
acs[30]=db2(ads8);
acs[31]=db3(ads8);
acs[32]=db4(ads8);


/*
combining 8 bits of 4 bits to get 32 bits data
*/


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

/*
MODULO 2 ADDITION OF RIGHT AND LEFT BITS
*/
printf("\n testingaddai[24] %d%d%d",a1[24],ap[24],acs[9]);
for(i=1;i<33;i++)
{
ar1[i]=a1[i]^ap[i];
a11[i]=ar[i];
}
/*
preoutput block which is exchange of right and left bits
*/


for(i=1;i<33;i++){
apb[i]=ar1[i];
apb[32+i]=ar[i];
}
```

```
 /*
 here the output of cipher function block is the input to
 the inverse permutation block
 */
printf("\n test apb %d",apb[31]);


aip[1]=apb[40];aip[2]=apb[8];aip[3]=apb[48];aip[4]=apb[16];

aip[5]=apb[56];aip[6]=apb[24];aip[7]=apb[64];aip[8]=apb[32];

aip[9]=apb[39];aip[10]=apb[7];aip[11]=apb[47];aip[12]=apb[15];

aip[13]=apb[55];aip[14]=apb[23];aip[15]=apb[63];aip[16]=apb[31];

aip[17]=apb[38];aip[18]=apb[6];aip[19]=apb[46];aip[20]=apb[14];

aip[21]=apb[54];aip[22]=apb[22];aip[23]=apb[62];aip[24]=apb[30];

aip[25]=apb[37];aip[26]=apb[5];aip[27]=apb[45];aip[28]=apb[13];

aip[29]=apb[53];aip[30]=apb[21];aip[31]=apb[61];aip[32]=apb[29];

aip[33]=apb[36];aip[34]=apb[4];aip[35]=apb[44];aip[36]=apb[12];

aip[37]=apb[52];aip[38]=apb[20];aip[39]=apb[60];aip[40]=apb[28];

aip[41]=apb[35];aip[42]=apb[3];aip[43]=apb[43];aip[44]=apb[11];

aip[45]=apb[51];aip[46]=apb[19];aip[47]=apb[59];aip[48]=apb[27];

aip[49]=apb[34];aip[50]=apb[2];aip[51]=apb[42];aip[52]=apb[10];

aip[53]=apb[50];aip[54]=apb[18];aip[55]=apb[58];aip[56]=apb[26];

aip[57]=apb[33];aip[58]=apb[1];aip[59]=apb[41];aip[60]=apb[9];

aip[61]=apb[49];aip[62]=apb[17];aip[63]=apb[57];aip[64]=apb[25];

printf("\n re convert of a\n");
for(i=1;i<65;i++)
{
printf("%d",aip[i]);
printf("\t");
}
/* 64 bit data for block b*/


b1[1]=b[58];      b1[2]=b[50];      b1[3]=b[42];      b1[4]=b[34];

b1[5]=b[26];      b1[6]=b[18];      b1[7]=b[10];      b1[8]=b[2];

b1[9]=b[60];      b1[10]=b[52];     b1[11]=b[44];     b1[12]=b[36];

b1[13]=b[28];     b1[14]=b[20];     b1[15]=b[12];     b1[16]=b[4];
```

```
bl[17]=b[62];    bl[18]=b[54];    bl[19]=b[46];    bl[20]=b[38];
bl[21]=b[30];    bl[22]=b[22];    bl[23]=b[14];    bl[24]=b[6];
bl[25]=b[64];    bl[26]=b[56];    bl[27]=b[48];    bl[28]=b[40];
bl[29]=b[32];    bl[30]=b[24];    bl[31]=b[16];    bl[32]=b[8];
br[1]=b[57];     br[2]=b[49];     br[3]=b[41];     br[4]=b[33];
br[5]=b[25];     br[6]=b[17];     br[7]=b[9]; br[8]=b[1];
br[9]=b[59];     br[10]=b[51];    br[11]=b[43];    br[12]=b[35];
br[13]=b[27];    br[14]=b[19];    br[15]=b[11];    br[16]=b[3];
br[17]=b[61];    br[18]=b[53];    br[19]=b[45];    br[20]=b[37];
br[21]=b[29];    br[22]=b[21];    br[23]=b[13];    br[24]=b[5];
br[25]=b[63];    br[26]=b[55];    br[27]=b[47];    br[28]=b[39];
br[29]=b[31];    br[30]=b[23];    br[31]=b[15];    br[32]=b[7];

/*
32 bits of right block goes through the cipher function which converts
32 bits into 48 bits
*/

be[1]=br[32];    be[2]=br[1];     be[3]=br[2];     be[4]=br[3];
be[5]=br[4];     be[6]=br[5];     be[7]=br[4];     be[8]=br[5];
be[9]=br[6];     be[10]=br[7];    be[11]=br[8];    be[12]=br[9];
be[13]=br[8];    be[14]=br[9];    be[15]=br[10];   be[16]=br[11];
be[17]=br[12];   be[18]=br[13];   be[19]=br[12];   be[20]=br[13];
be[21]=br[14];   be[22]=br[15];   be[23]=br[16];   be[24]=br[17];
be[25]=br[16];   be[26]=br[17];   be[27]=br[18];   be[28]=br[19];
be[29]=br[20];   be[30]=br[21];   be[31]=br[20];   be[32]=br[21];
be[33]=br[22];   be[34]=br[23];   be[35]=br[24];   be[36]=br[25];
be[37]=br[24];   be[38]=br[25];   be[39]=br[26];   be[40]=br[27];
be[41]=br[28];   be[42]=br[29];   be[43]=br[28];   be[44]=br[29];
be[45]=br[30];   be[46]=br[31];   be[47]=br[32];   be[48]=br[1];
```

```
printf("\n");
printf("Enter the key for b as a prev\n");


for(i=1;i<65;i++)
{
scanf("%d",&by);
bk[i]=by;
}


bp2[1]=bk[51]; bp2[2]=bk[10]; bp2[3]=bk[4];     bp2[4]=bk[16];
      bp2[5]=bk[61];

bp2[6]=bk[9];  bp2[7]=bk[2];  bp2[8]=bk[64];

bp2[9]=bk[1];
bp2[10]=bk[35];bp2[11]=bk[29];bp2[12]=bk[13];bp2[13]=bk[42];

bp2[14]=bk[37]; bp2[15]=bk[5];bp2[16]=bk[19];

bp2[17]=bk[12];
bp2[18]=bk[21];bp2[19]=bk[54];bp2[20]=bk[34];bp2[21]=bk[59];

bp2[22]=bk[18]; bp2[23]=bk[46];bp2[24]=bk[31];

bp2[25]=bk[63];
bp2[26]=bk[30];bp2[27]=bk[17];bp2[28]=bk[47];bp2[29]=bk[38];

bp2[30]=bk[22];bp2[31]=bk[8];bp2[32]=bk[40];

bp2[33]=bk[27];bp2[34]=bk[45];bp2[35]=bk[62];bp2[36]=bk[20];bp2[37]=bk[
43];

bp2[38]=bk[50];bp2[39]=bk[39];bp2[40]=bk[25];

bp2[41]=bk[30];bp2[42]=bk[60];bp2[43]=bk[23];bp2[44]=bk[55];bp2[45]=bk[
33];

bp2[46]=bk[26];bp2[47]=bk[52];bp2[48]=bk[48];

bp2[49]=bk[32];bp2[50]=bk[7];bp2[51]=bk[11];bp2[52]=bk[44];bp2[53]=bk[2
8];

bp2[54]=bk[56]; bp2[55]=bk[14];bp2[56]=bk[57];

bp2[57]=bk[15];bp2[58]=bk[53];bp2[59]=bk[36];bp2[60]=bk[24];bp2[61]=bk[
49];

bp2[62]=bk[58]; bp2[63]=bk[6];bp2[64]=bk[41];



bp1[1]=bp2[51]^bp2[10];
bp1[2]=bp2[4];bp1[3]=bp2[16];bp1[4]=bp2[61]^bp2[59];
```

```
bp1[5]=bp2[9]^bp2[2];bp1[6]=bp2[64];bp1[7]=bp2[35];bp1[8]=bp2[29];

bp1[9]=bp2[17];bp1[10]=bp2[42]^bp2[37];bp1[11]=bp2[50]^bp2[26];bp1[12]=
bp2[31];

bp1[13]=bp2[15]^bp2[53];bp1[14]=bp2[19];bp1[15]=bp2[12]^bp2[21];bp1[16]
=bp2[54];

bp1[17]=bp2[5]; bp1[18]=bp2[3];bp1[19]=bp2[6]^bp2[32];
bp1[20]=bp2[11]^bp2[34];

bp1[21]=bp2[18];bp1[22]=bp2[7];bp1[23]=bp2[27]^bp2[63];bp1[24]=bp2[36];

bp1[25]=bp2[13];bp1[26]=bp2[47];bp1[27]=bp2[44]^bp2[23];bp1[28]=bp2[8]^
bp2[43];

bp1[29]=bp2[20];bp1[30]=bp2[62]^bp2[39];bp1[31]=bp2[45]^bp2[24];bp1[32]
=bp2[14];

bp1[33]=bp2[55];bp1[34]=bp2[28];bp1[35]=bp2[25];bp1[36]=bp2[56];

bp1[37]=bp2[1];bp1[38]=bp2[41];bp1[39]=bp2[57];bp1[40]=bp2[33];

bp1[41]=bp2[41];bp1[42]=bp2[60];bp1[43]=bp2[40];bp1[44]=bp2[52];

bp1[45]=bp2[48]^bp2[38]; bp1[46]=bp2[41]^bp2[56];
bp1[47]=bp2[49];bp1[48]=bp2[33]^bp2[58];


/* modulo 2 addition of key and e block*/


for(i=1;i<49;i++)
bs[i]=be[i]^bp1[i];

/* 48 bit s block is converted into 8 blocks of 6 bits*/


for(i=1;i<=6;i++)
bs1[i]=bs[i];


for(i=7;i<=12;i++)
bs2[i-6]=bs[i];


for(i=13;i<=18;i++)
bs3[i-12]=bs[i];


for(i=19;i<=24;i++)
bs4[i-18]=bs[i];


for(i=25;i<=30;i++)
```

```
bs5[i-24]=bs[i];

for(i=31;i<=36;i++)
bs6[i-30]=bs[i];

for(i=37;i<=42;i++)
bs7[i-36]=bs[i];

for(i=43;i<=48;i++)
bs8[i-42]=bs[i];


/*
s1 block is converted into 4bit
*/

bd1r=bd2(bs1[1],bs1[6]);
bd1c=bd4(bs1[2],bs1[3],bs1[4],bs1[5]);
bds1=sa1[bd1r][bd1c];
bcs[1]=db1(bds1);
bcs[2]=db2(bds1);
bcs[3]=db3(bds1);
bcs[4]=db4(bds1);

/*
s2 block is converted into 4bit
*/


bd2r=bd2(bs2[1],bs2[6]);
bd2c=bd4(bs2[2],bs2[3],bs2[4],bs2[5]);
bds2=sa2[bd2r][bd2c];
bcs[5]=db1(bds2);
bcs[6]=db2(bds2);
bcs[7]=db3(bds2);
bcs[8]=db4(bds2);
/*
s3 converted into 4bits
*/

bd3r=bd2(bs3[1],bs3[6]);
bd3c=bd4(bs3[2],bs3[3],bs3[4],bs3[5]);
bds3=sa3[bd3r][bd3c];
bcs[9]=db1(bds3);
bcs[10]=db2(bds3);
bcs[11]=db3(bds3);
bcs[12]=db4(bds3);


/* s4 is converted to 4 bits
*/
bd4r=bd2(bs4[1],bs4[6]);
bd4c=bd4(bs4[2],bs4[3],bs4[4],bs4[5]);
bds4=sa4[bd4r][bd4c];
bcs[13]=db1(bds4);
bcs[14]=db2(bds4);
```

```
bcs[15]=db3(bds4);
bcs[16]=db4(bds4);

/* s5 is converted into 4 bits*/

bd5r=bd2(bs5[1],bs5[6]);
bd5c=bd4(bs5[2],bs5[3],bs5[4],bs5[5]);
bds5=sa5[bd5r][bd5c];
bcs[17]=db1(bds5);
bcs[18]=db2(bds5);
bcs[19]=db3(bds5);
bcs[20]=db4(bds5);

/* s6 is converted into 4 bits*/
bd6r=bd2(bs6[1],bs6[6]);
bd6c=bd4(bs6[2],bs6[3],bs6[4],bs6[5]);
bds6=sa6[bd6r][bd6c];
bcs[21]=db1(bds6);
bcs[22]=db2(bds6);
bcs[23]=db3(bds6);
bcs[24]=db4(bds6);

/* s7 is converted into 4 bits */

bd7r=bd2(bs7[1],bs7[6]);
bd7c=bd4(bs7[2],bs7[3],bs7[4],bs7[5]);
bds7=sa7[bd7r][bd7c];
bcs[25]=db1(bds7);
bcs[26]=db2(bds7);
bcs[27]=db3(bds7);
bcs[28]=db4(bds7);

/*
s8 is converted into 8 bits
*/
bd8r=bd2(bs8[1],bs8[6]);
bd8c=bd4(bs8[2],bs8[3],bs8[4],bs8[5]);
bds8=sa8[bd8r][bd8c];

bcs[29]=db1(bds8);
bcs[30]=db2(bds8);
bcs[31]=db3(bds8);
bcs[32]=db4(bds8);

/*
combining 8 blocks of 4 bit data to get 32 bit data
*/

bp[1]=bcs[16];bp[2]=bcs[7];bp[3]=bcs[20];bp[4]=bcs[21];

bp[5]=bcs[29];bp[6]=bcs[12];bp[7]=bcs[28];bp[8]=bcs[17];

bp[9]=bcs[1];bp[10]=bcs[15];bp[11]=bcs[23];bp[12]=bcs[26];

bp[13]=bcs[5];bp[14]=bcs[18];bp[15]=bcs[31];bp[16]=bcs[10];
```

```
bp[17]=bcs[2];bp[18]=bcs[8];bp[19]=bcs[24];bp[20]=bcs[14];

bp[21]=bcs[32];bp[22]=bcs[27];bp[23]=bcs[3];bp[24]=bcs[9];

bp[25]=bcs[19];bp[26]=bcs[13];bp[27]=bcs[30];bp[28]=bcs[6];

bp[29]=bcs[22];bp[30]=bcs[11];bp[31]=bcs[4];bp[32]=bcs[25];


/*
MODULO 2 ADDITION OF RIGHT AND LEFT BITS
*/


for(i=1;i<33;i++)
{
br1[i]=bl[i]^bp[i];
bl1[i]=br[i];
}
/*
preoutput block which is exchange of right and left bits
*/


for(i=1;i<33;i++){
bpb[i]=br1[i];
bpb[32+i]=br[i];
}
 /*
 here the output of cipher function block is the input to
 the inverse permutation block

 */


bip[1]=bpb[40];bip[2]=bpb[8];bip[3]=bpb[48];bip[4]=bpb[16];

bip[5]=bpb[56];bip[6]=bpb[24];bip[7]=bpb[64];bip[8]=bpb[32];

bip[9]=bpb[39];bip[10]=bpb[7];bip[11]=bpb[47];bip[12]=bpb[15];

bip[13]=bpb[55];bip[14]=bpb[23];bip[15]=bpb[63];bip[16]=bpb[31];

bip[17]=bpb[38];bip[18]=bpb[6];bip[19]=bpb[46];bip[20]=bpb[14];

bip[21]=bpb[54];bip[22]=bpb[22];bip[23]=bpb[62];bip[24]=bpb[30];

bip[25]=bpb[37];bip[26]=bpb[5];bip[27]=bpb[45];bip[28]=bpb[13];

bip[29]=bpb[53];bip[30]=bpb[21];bip[31]=bpb[61];bip[32]=bpb[29];

bip[33]=bpb[36];bip[34]=bpb[4];bip[35]=bpb[44];bip[36]=bpb[12];

bip[37]=bpb[52];bip[38]=bpb[20];bip[39]=bpb[60];bip[40]=bpb[28];
```

```
bip[41]=bpb[35];bip[42]=bpb[3];bip[43]=bpb[43];bip[44]=bpb[11];

bip[45]=bpb[51];bip[46]=bpb[19];bip[47]=bpb[59];bip[48]=bpb[27];

bip[49]=bpb[34];bip[50]=bpb[2];bip[51]=bpb[42];bip[52]=bpb[10];


bip[53]=bpb[50];bip[54]=bpb[18];bip[55]=bpb[58];bip[56]=bpb[26];

bip[57]=bpb[33];bip[58]=bpb[1];bip[59]=bpb[41];bip[60]=bpb[9];

bip[61]=bpb[49];bip[62]=bpb[17];bip[63]=bpb[57];bip[64]=bpb[25];

printf(" this is the decipher function of 64 bit data of b block\n");
for(i=1;i<65;i++)
{
printf("%d",bip[i]);
printf("\t");
}
printf("\n THE DECIPHER MESSAGE IS \n");

for(i=1;i<=64;i++)
{
data1[i]=aip[i];
data1[64+i]=bip[i];
}

for(i=1;i<=128;i++)
{
printf("%d",data1[i]);
printf("\t");
}
getch();
return 0;
}
```

```
int bd2(int a ,int b)
{
int s;
s=a*2+b;
return(s);
}

int bd4(int a, int b, int c, int d)
{
int s;
s=a*8+b*4+c*2+d;
return(s);
}
```

```
int db1(int z)
{
int b,m;
m=0x1 << 3;
b=(z&m) ? 1:0;
return(b);
}

int db2(int z)
{
int b,m;
m=0x1 <<3;
m>>=1;
b=(z&m)?1:0;
return(b);
}

int db3(int z)
{
int b,m;
m=0x1<<3;
m>>=2;
b=(z&m)?1:0;

return(b);
}

int db4(int z)
{
int b,m;
m=0x1<<4;
m>>=2;
b=(z&m)?1:0;

return(b);
}
```