

VISVESVARAYA TECHNOLOGICAL UNIVERSITY
BELAGAVI-590014, KARNATAKA



A Mini Project Report

On

“DES SECURITY ALGORITHM”

For the Award of Degree

BACHELOR OF ENGINEERING

IN

COMPUTER SCIENCE & ENGINEERING

Submitted By:

AMIT HEBBI (1SG17CS010)

MANAV PRADHAN (1SG17CS049)

Under the Guidance of:

Mrs. Latha A

Assistant Professor

Mrs. Chaitra K T

Coordinator



Department of Computer Science and Engineering

SAPTHAGIRI COLLEGE OF ENGINEERING

14/5, Chikkasandra, Hesarghatta Main Road Bengaluru – 560057

2019-2020

SAPTHAGIRI COLLEGE OF ENGINEERING

**14/5, Chikkasandra, Hesaraghatta Main Road, Bengaluru –
560057.**

Department of Computer Science and Engineering



CERTIFICATE

Certified that the Mini Project Work on **“DES SECURITY ALGORITHM”** carried out by **AMIT HEBBI (1SG17CS010) & MANAV PRADHAN (1SG17CS049)**, bonafide students of **Sapthagiri College of Engineering**, in partial fulfilment for the award of **Bachelor of Engineering** degree in **Computer Science and Engineering** of **Visvesvaraya Technological University, Belagavi** during the academic year 2019-2020. It is certified that all corrections/suggestions indicated for Internal Assessment have been incorporated in the report deposited in the department library.

Signature of the Guide

Mrs. Latha A
Assistant Professor

Signature of Coordinator

Mrs. Chaitra K T
Assistant Professor

Signature of the HOD

Dr. Kamalakshi Naganna
Professor & HOD

ACKNOWLEDGEMENT

Any achievement does not depend solely on the individual efforts but on the guidance, encouragement and co-operation of intellectuals, elders and friends. A number of personalities, in their own capacities have helped us in carrying out this mini project work. We would like to take this opportunity to thank them all.

We would like to express my profound thanks to **Sri. G Dayanand**, Chairman, Sapthagiri College of Engineering Bangalore, for his continuous support in providing amenities to carry out this Mini Project.

Special Thanks to **Dr. N. Srinivasan**, Director, Sapthagiri College of Engineering Bangalore, for his valuable suggestion.

Also, we would like to express our immense gratitude to **Dr. H Ramakrishna**, Principal, Sapthagiri College of Engineering Bangalore, for his help and inspiration during the tenure of the course.

We also extend our sincere thanks to **Dr. Kamalakshi Naganna**, Professor and Head, Department of Computer Science and Engineering, Sapthagiri College of Engineering, for her constant support.

We would like to express our heartfelt gratitude to **Mrs. Latha A**, Assistant professor and **Mrs. Chaitra K T**, Assistant professor, Department of Computer Science and Engineering, Sapthagiri College of Engineering, for their timely advice on the mini project and regular assistance throughout the work.

We also extend our sincere thanks to all the **Faculty members** and **supporting staff** Department of Computer Science and Engineering, Sapthagiri College of Engineering, for their constant support and encouragement.

Finally, we thank our parents and friends for their moral support.

Amit Hebbi

Manav Pradhan

ABSTRACT

The data which is encrypted by symmetric key method is called ***Data Encryption Standard (DES)***. ***It was*** prepared by IBM in 1974 and declared as national standard in 1997. Government was also using cryptography especially in diplomatic communication and military. ***Without*** cryptography it's difficult to interpret military communication. Cryptography was also used in commercial sector. Federal information Processing Standard (FIPS) was also working and DES. FIPS was integrated with computer security Program with its relevant institution. In structured standards development process Data Encryption Standard has been a forerunner.

TABLE OF CONTENTS

SL. NO.	CHAPTER	PAGE NO
1.	INTRODUCTION	6
2.	HISTORY	7
	2.1 CHRONOLOGY	7
3.	DESIGN	10
	3.1 PRELIMINARY EXAMPLES OF DES	10
	3.2 HOW DES WORKS IN DETAIL	11
	3.3 DES ALGORITHM	16
4.	DES IMPLEMENTATION	18
	4.1 THE DES ALGORITHM STEPS	18
5.	DES MODES OF OPERATION	31
6.	DES ANALYSIS	33
	6.1 PROPERTIES	33
7.	RESULT AND SCREENSHOTS	35
8.	DES APPLICATION	37
	8.1 ADVANTAGES AND DISADVANTAGES	37
	8.2 CRACKING DES	38
	8.3 TRIPLE-DES	39
9.	CONCLUSION	40

CHAPTER 1

INTRODUCTION

The DES (Data Encryption Standard) algorithm is the most widely used encryption algorithm in the world. For many years, and among many people, "secret code making" and DES have been synonymous. And despite the recent coup by the Electronic Frontier Foundation in creating a \$220,000 machine to crack DES-encrypted messages, DES will live on in government and banking for years to come through a life- extending version called "triple-DES."

How does DES work? This article explains the various steps involved in DES-encryption, illustrating each step by means of a simple example. Since the creation of DES, many other algorithms (recipes for changing data) have emerged which are based on design principles similar to DES. Once you understand the basic transformations that take place in DES, you will find it easy to follow the steps involved in these more recent algorithms.

But first a bit of history of how DES came about is appropriate, as well as a look toward the future.

CHAPTER 2

HISTORY

The origins of DES go back to the early 1970s. In 1972, after concluding a study on the US government's computer security needs, the US standards body NBS (National Bureau of Standards)—now named NIST (National Institute of Standards and Technology)—identified a need for a government-wide standard for encrypting unclassified, sensitive information.

Around the same time, engineer Mohamed Atalla in 1972 founded Atalla Corporation and developed the first hardware security module (HSM), the so-called "Atalla Box" which was commercialized in 1973. It protected offline devices with an un-guessable PIN generating key, and was a commercial success. Banks and credit card companies were fearful that Atalla would dominate the market, which spurred the development of an international encryption standard.^[4] The IBM 3624 later adopted a similar PIN verification system to the earlier Atalla system.

On 15 May 1973, after consulting with the NSA, NBS solicited proposals for a cipher that would meet rigorous design criteria. None of the submissions, however, turned out to be suitable. A second request was issued on 27 August 1974. This time, IBM submitted a candidate which was deemed acceptable—a cipher developed during the period 1973–1974 based on an earlier algorithm, Horst Feistel's Lucifer cipher. The team at IBM involved in cipher design and analysis included Feistel, Walter Tuchman, Don Coppersmith, Alan Konheim, Carl Meyer, Mike Matyas, Roy Adler, Edna Grossman, Bill Notz, Lynn Smith, and Bryant Tuckerman.

CHRONOLOGY

Date	Year	Event
15 May	1973	NBS publishes a first request for a standard encryption algorithm
27 August	1974	NBS publishes a second request for encryption algorithms
17 March	1975	DES is published in the <i>Federal Register</i> for comment
August	1976	First workshop on DES

September	1976	Second workshop, discussing mathematical foundation of DES
November	1976	DES is approved as a standard
15 January	1977	DES is published as a FIPS standard FIPS PUB 46
	1983	DES is reaffirmed for the first time
	1986	Videocipher II, a TV satellite scrambling system based upon DES, begins use by HBO
22 January	1988	DES is reaffirmed for the second time as FIPS 46-1, superseding FIPS PUB 46
July	1991	Biham and Shamir rediscover differential cryptanalysis, and apply it to a 15-round DES-like cryptosystem.
	1992	Biham and Shamir report the first theoretical attack with less complexity than brute force: differential cryptanalysis. However, it requires an unrealistic 2^{47} chosen plaintexts.
30 December	1993	DES is reaffirmed for the third time as FIPS 46-2
	1994	The first experimental cryptanalysis of DES is performed using linear cryptanalysis (Matsui, 1994).
June	1997	The DESCHALL Project breaks a message encrypted with DES for the first time in public.
July	1998	The EFF's DES cracker (Deep Crack) breaks a DES key in 56 hours.
January	1999	Together, Deep Crack and distributed.net break a DES key in 22 hours and 15 minutes.
25 October	1999	DES is reaffirmed for the fourth time as FIPS 46-3, which specifies the preferred use of Triple DES, with single DES permitted only in legacy systems.
26 November	2001	The Advanced Encryption Standard is published in FIPS 197
26 May	2002	The AES becomes effective
26 July	2004	The withdrawal of FIPS 46-3 (and a couple of related standards) is proposed in the <i>Federal Register</i>
19 May	2005	NIST withdraws FIPS 46-3 (see Federal Register vol 70, number 96)

April	2006	The FPGA-based parallel machine COPACOBANA of the Universities of Bochum and Kiel, Germany, breaks DES in 9 days at a \$10,000 hardware cost. ^[23] Within a year software improvement reduced the average time to 6.4 days.
Nov.	2008	The successor of COPACOBANA, the RIVYERA machine, reduced the average time to less than a single day.
August	2016	The Open Source password cracking software hashcat added in DES brute force searching on general purpose GPUs. Benchmarking shows a single off the shelf Nvidia GeForce GTX 1080 Ti GPU costing \$1000 USD recovers a key in an average of 15 days (full exhaustive search taking 30 days). Systems have been built with eight GTX 1080 Ti GPUs which can recover a key in an average of under 2 days. ^[24]
July	2017	A chosen-plaintext attack utilizing a rainbow table can recover the DES key for a single specific chosen plaintext <i>1122334455667788</i> in 25 seconds. A new rainbow table has to be calculated per plaintext. A limited set of rainbow tables have been made available for download.

CHAPTER 3

DESIGN

3.1 Some Preliminary Examples of DES

DES works on bits, or binary numbers--the 0s and 1s common to digital computers. Each group of four bits makes up a hexadecimal, or base 16, number. Binary "0001" is equal to the hexadecimal number "1", binary "1000" is equal to the hexadecimal number "8", "1001" is equal to the hexadecimal number "9", "1010" is equal to the hexadecimal number "A", and "1111" is equal to the hexadecimal number "F".

DES works by encrypting groups of 64 message bits, which is the same as 16 hexadecimal numbers. To do the encryption, DES uses "keys" where are also *apparently* 16 hexadecimal numbers long, or *apparently* 64 bits long. However, every 8th key bit is ignored in the DES algorithm, so that the effective key size is 56 bits. But, in any case, 64 bits (16 hexadecimal digits) is the round number upon which DES is organized.

For example, if we take the plaintext message "8787878787878787", and encrypt it with the DES key "0E329232EA6D0D73", we end up with the ciphertext "0000000000000000". If the ciphertext is decrypted with the same secret DES key "0E329232EA6D0D73", the result is the original plaintext "8787878787878787".

This example is neat and orderly because our plaintext was exactly 64 bits long. The same would be true if the plaintext happened to be a multiple of 64 bits. But most messages will not fall into this category. They will not be an exact multiple of 64 bits (that is, an exact multiple of 16 hexadecimal numbers).

For example, take the message "Your lips are smoother than vaseline". This plaintext message is 38 bytes (76 hexadecimal digits) long. So, this message must be padded with some extra bytes at the tail end for the encryption. Once the encrypted message has been decrypted, these extra bytes are thrown away. There are, of course, different padding schemes--different ways to add extra bytes. Here we will just add 0s at the end, so that the total message is a multiple of 8 bytes (or 16 hexadecimal digits, or 64 bits).

The plaintext message "Your lips are smoother than vaseline" is, in hexadecimal,

"596F7572206C6970 732061726520736D 6F6F746865722074 68616E2076617365
6C696E650D0A".

(Note here that the first 72 hexadecimal digits represent the English message, while "0D" is hexadecimal for Carriage Return, and "0A" is hexadecimal for Line Feed, showing that the message file has terminated.) We then pad this message with some 0s on the end, to get a total of 80 hexadecimal digits:

"596F7572206C6970 732061726520736D 6F6F746865722074 68616E2076617365
6C696E650D0A0000".

If we then encrypt this plaintext message 64 bits (16 hexadecimal digits) at a time, using the same DES key "0E329232EA6D0D73" as before, we get the ciphertext:

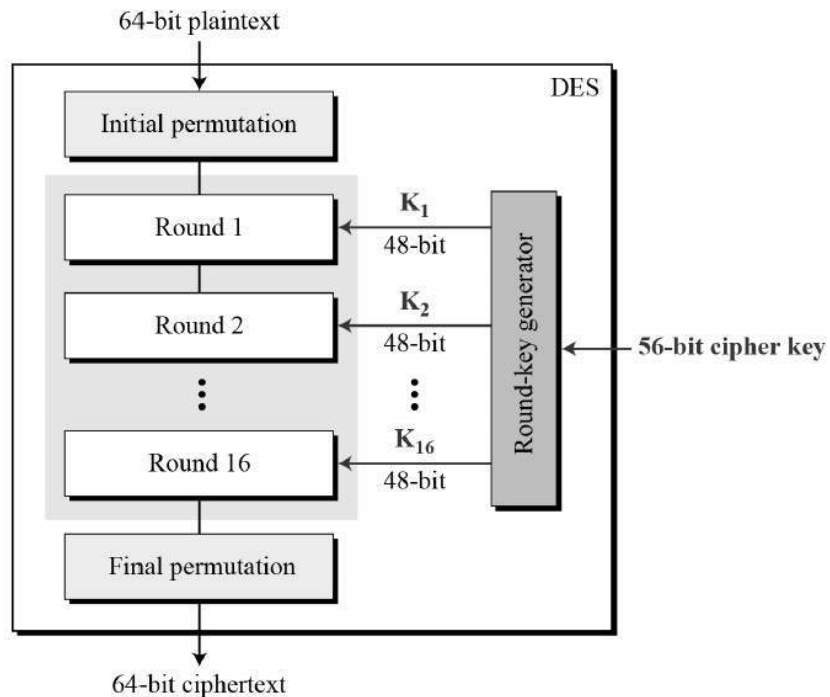
"C0999FDDE378D7ED 727DA00BCA5A84EE 47F269A4D6438190 9DD52F78F5358499
828AC9B453E0E653".

This is the secret code that can be transmitted or stored. Decrypting the ciphertext restores the original message "Your lips are smoother than vaseline". (Think how much better off Bill Clinton would be today, if Monica Lewinsky had used encryption on her Pentagon computer!)

3.2 How DES Works in Detail

DES is a **block cipher**--meaning it operates on plaintext blocks of a given size (64-bits) and returns ciphertext blocks of the same size. Thus DES results in a **permutation** among the 2^{64} (read this as: "2 to the 64th power") possible arrangements of 64 bits, each of which may be either 0 or 1. Each block of 64 bits is divided into two blocks of 32 bits each, a left half block **L** and a right half **R**. (This division is only used in certain operations.)

General Structure of DES is depicted in the following illustration –

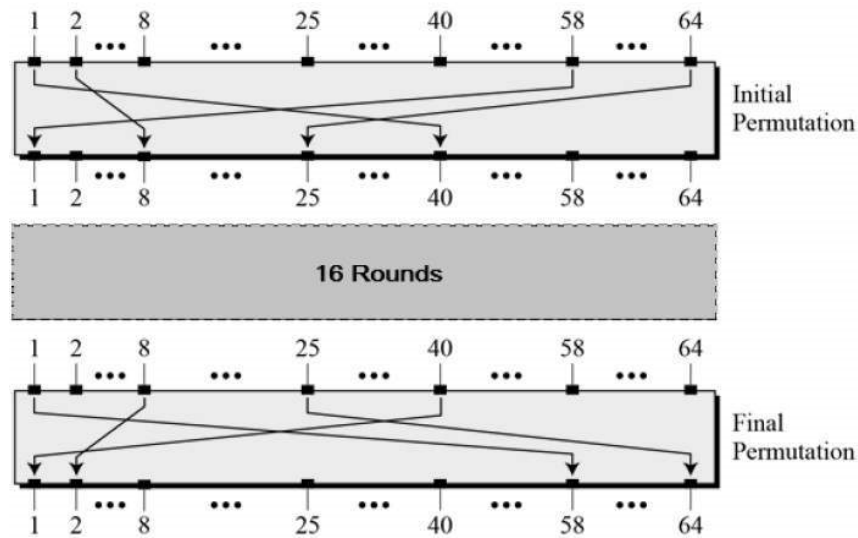


Since DES is based on the Feistel Cipher, all that is required to specify DES is –

- Round function
- Key schedule
- Any additional processing – Initial and final permutation

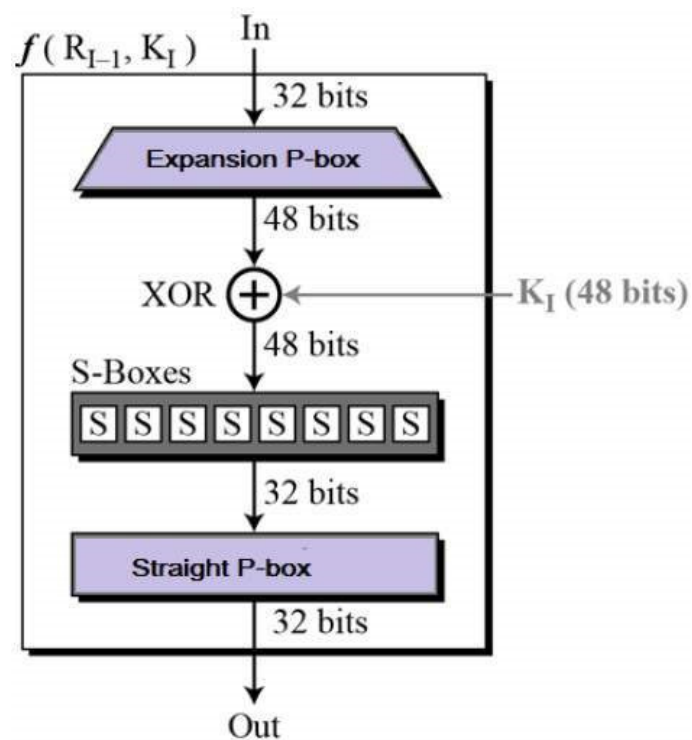
Initial and Final Permutation

The initial and final permutations are straight Permutation boxes (P-boxes) that are inverses of each other. They have no cryptography significance in DES. The initial and final permutations are shown as follows –

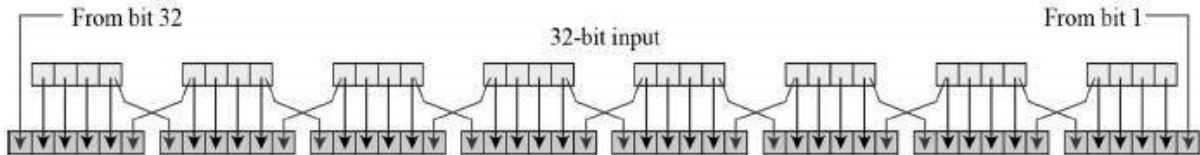


Round Function

The heart of this cipher is the DES function, f . The DES function applies a 48-bit key to the rightmost 32 bits to produce a 32-bit output.



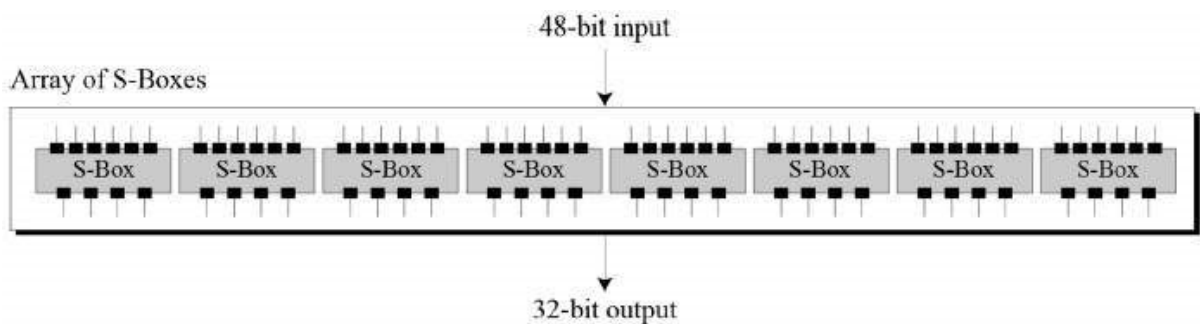
- **Expansion Permutation Box** – Since right input is 32-bit and round key is a 48-bit, we first need to expand right input to 48 bits. Permutation logic is graphically depicted in the following illustration –



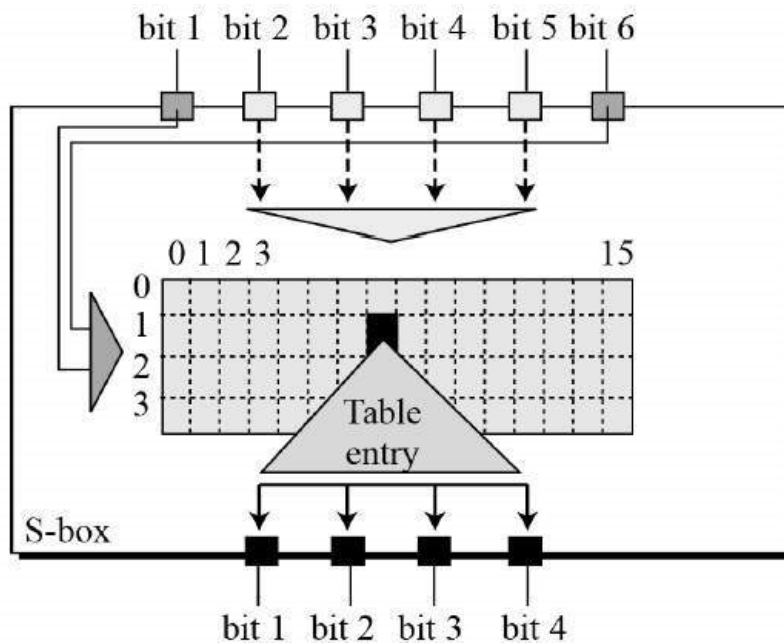
- The graphically depicted permutation logic is generally described as table in DES specification illustrated as shown –

32	01	02	03	04	05
04	05	06	07	08	09
08	09	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	31	31	32	01

- **XOR (Whitener).** – After the expansion permutation, DES does XOR operation on the expanded right section and the round key. The round key is used only in this operation.
- **Substitution Boxes.** – The S-boxes carry out the real mixing (confusion). DES uses 8 S-boxes, each with a 6-bit input and a 4-bit output. Refer the following illustration –



- The S-box rule is illustrated below –

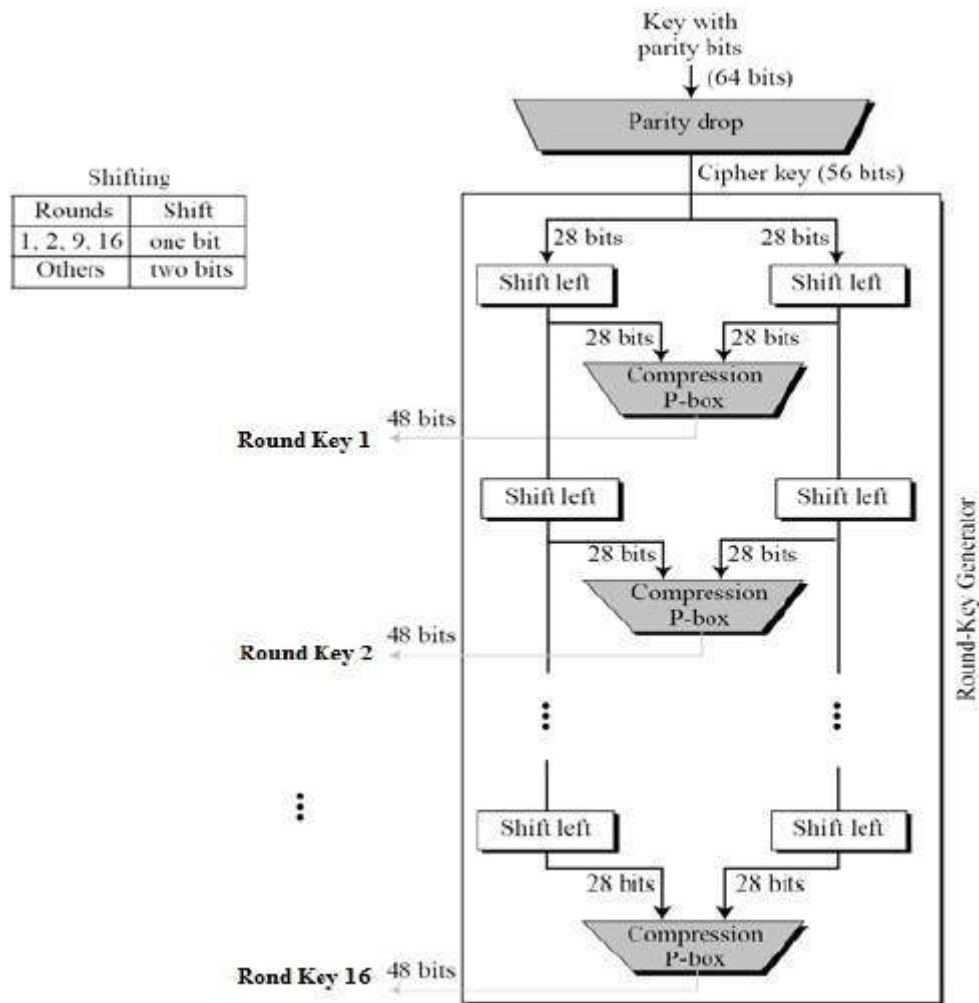


- There are total of eight S-box tables. The output of all eight s-boxes is then combined in to 32-bit section.
- Straight Permutation** – The 32-bit output of S-boxes is then subjected to the straight permutation with rule shown in the following illustration:

16	07	20	21	29	12	28	17
01	15	23	26	05	18	31	10
02	08	24	14	32	27	03	09
19	13	30	06	22	11	04	25

Key Generation

The round-key generator creates sixteen 48-bit keys out of a 56-bit cipher key. The process of key generation is depicted in the following illustration –



The logic for Parity drops, shifting, and Compression P-box is given in the DES description.

3.3 DES Algorithm

In the encryption process, the data is first divided into 64-bit long blocks. Then, each block undergoes the following operations:

1. Initial permutation rearranges bits in a certain, predefined way. This step does not enhance the security of algorithm. It was introduced to make passing data into encryption machines easier, at the times when the cipher was invented.
2. The input data is divided into two 32-bit parts: the left one and the right one.
3. 56 bits are selected from the 64-bit key (Permutation PC-1). They are then divided into two 28-bit parts.
4. Sixteen rounds of the following operations (so called Feistel functions) are then performed:

1. Both halves of key are rotated left by one or two bits (specified for each round). Then 48 subkey bits are selected by Permutation PC-2.
 2. The right half of data is expanded to 48 bits using the Expansion Permutation.
 3. The expanded half of data is combined using XOR operation with the 48-bit subkey chosen earlier.
 4. The combined data is divided into eight 6-bit pieces. Each part is then an input to one of the S-Boxes (the first 6-bit part is the input to the first S-Box, the second 6-bit part enters the second S-Box, and so on). The first and the last bits stand for the row, and the rest of bits define the column of an S-Box table. After determining the location in the table, the value is read and converted to binary format. The output from each S-Box is 4-bit long, so the output from all S-Boxes is 32-bit long. Each S-box has a different structure.
 5. The output bits from S-Boxes are combined, and they undergo P-Box Permutation.
 6. Then, the bits of the changed right side are added to the bits of the left side.
 7. The modified left half of data becomes a new right half, and the previous right half becomes a new left side.
5. After all, sixteen rounds, the left and the right halves of data are combined using the XOR operation.
 6. The Final Permutation is performed.

During decryption, the same set of operations is performed but in reverse order. The subkeys are also selected in reverse order (compared to encryption).

CHAPTER 4

DES Implementation

Example: Let **M** be the plain text message **M** = 0123456789ABCDEF, where **M** is in hexadecimal (base 16) format. Rewriting **M** in binary format, we get the 64-bit block of text:

M = 0000 0001 0010 0011 0100 0101 0110 0111 1000 1001 1010 1011 1100 1101 1110 1111

L = 0000 0001 0010 0011 0100 0101 0110 0111

R = 1000 1001 1010 1011 1100 1101 1110 1111

The first bit of **M** is "0". The last bit is "1". We read from left to right.

DES operates on the 64-bit blocks using *key* sizes of 56- bits. The keys are actually stored as being 64 bits long, but every 8th bit in the key is not used (i.e. bits numbered 8, 16, 24, 32, 40, 48, 56, and 64). However, we will nevertheless number the bits from 1 to 64, going left to right, in the following calculations. But, as you will see, the eight bits just mentioned get eliminated when we create subkeys.

Example: Let **K** be the hexadecimal key **K** = 133457799BBCDFF1. This gives us as the binary key (setting 1 = 0001, 3 = 0011, etc., and grouping together every eight bits, of which the last one in each group will be unused):

K = 00010011 00110100 01010111 01111001 10011011 10111100 11011111 11110001

4.1 The DES algorithm uses the following steps:

Step 1: Create 16 subkeys, each of which is 48-bits long.

The 64-bit key is permuted according to the following table, **PC-1**. Since the first entry in the table is "57", this means that the 57th bit of the original key **K** becomes the first bit of the permuted key **K+**. The 49th bit of the original key becomes the second bit of the permuted key. The 4th bit of the original key is the last bit of the permuted key. Note only 56 bits of the original key appear in the permuted key.

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

Example: From the original 64-bit key

K = 00010011 00110100 01010111 01111001 10011011 10111100 11011111 11110001

we get the 56-bit permutation

K⁺ = 1111000 0110011 0010101 0101111 0101010 1011001 1001111 0001111

Next, split this key into left and right halves, **C**₀ and **D**₀, where each half has 28 bits.

Example: From the permuted key **K**⁺, we get

C₀ = 1111000 0110011 0010101 0101111

D₀ = 0101010 1011001 1001111 0001111

With **C**₀ and **D**₀ defined, we now create sixteen blocks **C**_{*n*} and **D**_{*n*}, 1 ≤ *n* ≤ 16. Each pair of blocks **C**_{*n*} and **D**_{*n*} is formed from the previous pair **C**_{*n-1*} and **D**_{*n-1*}, respectively, for *n* = 1, 2, ..., 16, using the following schedule of "left shifts" of the previous block. To do a left shift, move each bit one place to the left, except for the first bit, which is cycled to the end of the block.

Iteration Number	Number of Left Shifts
---------------------	--------------------------

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

This means, for example, C_3 and D_3 are obtained from C_2 and D_2 , respectively, by two left shifts, and C_{16} and D_{16} are obtained from C_{15} and D_{15} , respectively, by one left shift. In all cases, by a single left shift is meant a rotation of the bits one place to the left, so that after one left shift the bits in the 28 positions are the bits that were previously in positions 2, 3..., 28, 1.

Example: From original pair C_0 and D_0 we obtain:

$$C_0 = 1111000011001100101010101111$$
$$D_0 = 01010101011001100111110001111$$
$$C_1 = 1110000110011001010101011111$$
$$D_1 = 10101010110011001111100011110$$
$$C_2 = 1100001100110010101010111111$$
$$D_2 = 01010101100110011111000111101$$

$C_3 = 0000110011001010101011111111$

$D_3 = 0101011001100111100011110101$

$C_4 = 0011001100101010101111111100$

$D_4 = 0101100110011110001111010101$

$C_5 = 1100110010101010111111110000$

$D_5 = 0110011001111000111101010101$

$C_6 = 0011001010101011111111000011$

$D_6 = 1001100111100011110101010101$

$C_7 = 1100101010101111111100001100$

$D_7 = 0110011110001111010101010110$

$C_8 = 0010101010111111110000110011$

$D_8 = 1001111000111101010101011001$

$C_9 = 0101010101111111100001100110$

$D_9 = 0011110001111010101010110011$

$C_{10} = 0101010111111110000110011001$

$D_{10} = 1111000111101010101011001100$

$C_{11} = 0101011111111000011001100101$

$D_{11} = 1100011110101010101100110011$

$C_{12} = 0101111111100001100110010101$

$D_{12} = 0001111010101010110011001111$

$C_{13} = 0111111110000110011001010101$

$D_{13} = 0111101010101011001100111100$

$C_{14} = 1111111000011001100101010101$

$D_{14} = 1110101010101100110011110001$

$C_{15} = 1111100001100110010101010111$ $D_{15} = 1010101010110011001111000111$ $C_{16} = 1111000011001100101010101111$ $D_{16} = 0101010101100110011110001111$

We now form the keys K_n , for $1 \leq n \leq 16$, by applying the following permutation table to each of the concatenated pairs $C_n D_n$. Each pair has 56 bits, but **PC-2** only uses 48 of these.

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

Therefore, the first bit of K_n is the 14th bit of $C_n D_n$, the second bit the 17th, and so on, ending with the 48th bit of K_n being the 32th bit of $C_n D_n$.

Example: For the first key we have $C_1 D_1 = 1110000 \ 1100110 \ 0101010 \ 1011111 \ 1010101 \ 0110011 \ 0011110 \ 0011110$

which, after we apply the permutation **PC-2**, becomes

 $K_1 = 000110 \ 110000 \ 001011 \ 101111 \ 111111 \ 000111 \ 000001 \ 110010$

For the other keys we have

 $K_2 = 011110 \ 011010 \ 111011 \ 011001 \ 110110 \ 111100 \ 100111 \ 100101$ $K_3 = 010101 \ 011111 \ 110010 \ 001010 \ 010000 \ 101100 \ 111110 \ 011001$ $K_4 = 011100 \ 101010 \ 110111 \ 010110 \ 110110 \ 110011 \ 010100 \ 011101$ $K_5 = 011111 \ 001110 \ 110000 \ 000111 \ 111010 \ 110101 \ 001110 \ 101000$

$K_6 = 011000\ 111010\ 010100\ 111110\ 010100\ 000111\ 101100\ 101111$
 $K_7 = 111011\ 001000\ 010010\ 110111\ 111101\ 100001\ 100010\ 111100$
 $K_8 = 111101\ 111000\ 101000\ 111010\ 110000\ 010011\ 101111\ 111011$
 $K_9 = 111000\ 001101\ 101111\ 101011\ 111011\ 011110\ 011110\ 000001$
 $K_{10} = 101100\ 011111\ 001101\ 000111\ 101110\ 100100\ 011001\ 001111$
 $K_{11} = 001000\ 010101\ 111111\ 010011\ 110111\ 101101\ 001110\ 000110$
 $K_{12} = 011101\ 010111\ 000111\ 110101\ 100101\ 000110\ 011111\ 101001$
 $K_{13} = 100101\ 111100\ 010111\ 010001\ 111110\ 101011\ 101001\ 000001$
 $K_{14} = 010111\ 110100\ 001110\ 110111\ 111100\ 101110\ 011100\ 111010$
 $K_{15} = 101111\ 111001\ 000110\ 001101\ 001111\ 010011\ 111100\ 001010$
 $K_{16} = 110010\ 110011\ 110110\ 001011\ 000011\ 100001\ 011111\ 110101$

So much for the subkeys. Now we look at the message itself.

Step 2: Encode each 64-bit block of data.

There is an *initial permutation* **IP** of the 64 bits of the message data **M**. This rearranges the bits according to the following table, where the entries in the table show the new arrangement of the bits from their initial order. The 58th bit of **M** becomes the first bit of **IP**. The 50th bit of **M** becomes the second bit of **IP**. The 7th bit of **M** is the last bit of **IP**.

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

Example: Applying the initial permutation to the block of text **M**, given previously, we get

M = 0000 0001 0010 0011 0100 0101 0110 0111 1000 1001 1010 1011 1100 1101 1110 1111

IP = 1100 1100 0000 0000 1100 1100 1111 1111 1111 0000 1010 1010 1111 0000 1010 1010

Here the 58th bit of **M** is "1", which becomes the first bit of **IP**. The 50th bit of **M** is "1", which becomes the second bit of **IP**. The 7th bit of **M** is "0", which becomes the last bit of **IP**.

Next divide the permuted block **IP** into a left half **L₀** of 32 bits, and a right half **R₀** of 32 bits.

Example: From **IP**, we get **L₀** and **R₀**

L₀ = 1100 1100 0000 0000 1100 1100 1111 1111

R₀ = 1111 0000 1010 1010 1111 0000 1010 1010

We now proceed through 16 iterations, for $1 \leq n \leq 16$, using a function **f** which operates on two blocks--a data block of 32 bits and a key **K_n** of 48 bits--to produce a block of 32 bits. **Let + denote XOR addition, (bit-by-bit addition modulo 2).** Then for **n** going from 1 to 16 we calculate

$$L_n = R_{n-1}$$

$$R_n = L_{n-1} + f(R_{n-1}, K_n)$$

This results in a final block, for $n = 16$, of **L₁₆R₁₆**. That is, in each iteration, we take the right 32 bits of the previous result and make them the left 32 bits of the current step. For the right 32 bits in the current step, we XOR the left 32 bits of the previous step with the calculation **f**.

Example: For $n = 1$, we have

K₁ = 000110 110000 001011 101111 111111 000111 000001 110010

L₁ = **R₀** = 1111 0000 1010 1010 1111 0000 1010 1010

$$R_1 = L_0 + f(R_0, K_1)$$

It remains to explain how the function **f** works. To calculate **f**, we first expand each block **R_{n-1}** from 32 bits to 48 bits. This is done by using a selection table that repeats some of the bits in **R_{n-1}**. We'll call the use of this selection table the function **E**. Thus **E(R_{n-1})** has a 32-bit input block, and a 48-bit output block.

Let \mathbf{E} be such that the 48 bits of its output, written as 8 blocks of 6 bits each, are obtained by selecting the bits in its inputs in order according to the following table:

E BIT-SELECTION TABLE

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

Thus, the first three bits of $\mathbf{E}(\mathbf{R}_{n-1})$ are the bits in positions 32, 1 and 2 of \mathbf{R}_{n-1} while the last 2 bits of $\mathbf{E}(\mathbf{R}_{n-1})$ are the bits in positions 32 and 1.

Example: We calculate $\mathbf{E}(\mathbf{R}_0)$ from \mathbf{R}_0 as follows:

$\mathbf{R}_0 = 1111\ 0000\ 1010\ 1010\ 1111\ 0000\ 1010\ 1010$

$\mathbf{E}(\mathbf{R}_0) = 011110\ 100001\ 010101\ 010101\ 011110\ 100001\ 010101\ 010101$

(Note that each block of 4 original bits has been expanded to a block of 6 output bits.)

Next in the f calculation, we XOR the output $\mathbf{E}(\mathbf{R}_{n-1})$ with the key \mathbf{K}_n :

$\mathbf{K}_n + \mathbf{E}(\mathbf{R}_{n-1})$.

Example: For \mathbf{K}_1 , $\mathbf{E}(\mathbf{R}_0)$, we have

$\mathbf{K}_1 = 000110\ 110000\ 001011\ 101111\ 111111\ 000111\ 000001\ 110010$

$\mathbf{E}(\mathbf{R}_0) = 011110\ 100001\ 010101\ 010101\ 011110\ 100001\ 010101\ 010101$

$\mathbf{K}_1 + \mathbf{E}(\mathbf{R}_0) = 011000\ 010001\ 011110\ 111010\ 100001\ 100110\ 010100\ 100111$.

We have not yet finished calculating the function f . To this point we have expanded \mathbf{R}_{n-1} from 32 bits to 48 bits, using the selection table, and XORed the result with the key \mathbf{K}_n . We now have 48 bits, or eight groups of six bits. We now do something strange with each group of six

bits: we use them as addresses in tables called "**S boxes**". Each group of six bits will give us an address in a different **S** box. Located at that address will be a 4-bit number. This 4-bit number will replace the original 6 bits. The net result is that the eight groups of 6 bits are transformed into eight groups of 4 bits (the 4-bit outputs from the **S** boxes) for 32 bits total.

Write the previous result, which is 48 bits, in the form:

$$K_n + E(R_{n-1}) = B_1B_2B_3B_4B_5B_6B_7B_8,$$

where each B_i is a group of six bits. We now calculate

$S_1(B_1)S_2(B_2)S_3(B_3)S_4(B_4)S_5(B_5)S_6(B_6)S_7(B_7)S_8(B_8)$ where $S_i(B_i)$ refers to the output of the i -th **S** box.

To repeat, each of the functions S_1, S_2, \dots, S_8 , takes a 6-bit block as input and yields a 4-bit block as output. The table to determine S_1 is shown and explained below:

S1																
Column Number																
Row																
No.	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

If S_1 is the function defined in this table and B is a block of 6 bits, then $S_1(B)$ is determined as follows: The first and last bits of B represent in base 2 a number in the decimal range 0 to 3 (or binary 00 to 11). Let that number be i . The middle 4 bits of B represent in base 2 a number in the decimal range 0 to 15 (binary 0000 to 1111). Let that number be j . Look up in the table the number in the i -th row and j -th column. It is a number in the range 0 to 15 and is uniquely represented by a 4-bit block. That block is the output $S_1(B)$ of S_1 for the input B . For example, for input block $B = 011011$ the first bit is "0" and the last bit "1" giving 01 as the row. This is

row 1. The middle four bits are "1101". This is the binary equivalent of decimal 13, so the column is column number 13. In row 1, column 13 appears 5. This determines the output; 5 is binary 0101, so that the output is 0101. Hence $S_1(011011) = 0101$.

The tables defining the functions S_1, \dots, S_8 are the following:

S1

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

S2

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

S3

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

S4

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

S5

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

S6

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

S7

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

S8

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

Example: For the first round, we obtain as the output of the eight **S** boxes:

$$K_I + E(R_0) = 011000\ 010001\ 011110\ 111010\ 100001\ 100110\ 010100\ 100111.$$

$$S_1(B_1)S_2(B_2)S_3(B_3)S_4(B_4)S_5(B_5)S_6(B_6)S_7(B_7)S_8(B_8) = 0101\ 1100\ 1000\ 0010\ 1011\ 0101\ 1001\ 0111$$

The final stage in the calculation of f is to do a permutation **P** of the **S**-box output to obtain the final value of f :

$$f = P(S_1(B_1)S_2(B_2)...S_8(B_8))$$

The permutation **P** is defined in the following table. **P** yields a 32-bit output from a 32-bit input by permuting the bits of the input block.

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

Example: From the output of the eight **S** boxes:

$$S_1(B_1)S_2(B_2)S_3(B_3)S_4(B_4)S_5(B_5)S_6(B_6)S_7(B_7)S_8(B_8) = 0101\ 1100\ 1000\ 0010\ 1011\ 0101\ 1001\ 0111$$

we get

$$f = 0010\ 0011\ 0100\ 1010\ 1010\ 1001\ 1011\ 1011$$

$$R_1 = L_0 + f(R_0, K_1)$$

$$= 1100\ 1100\ 0000\ 0000\ 1100\ 1100\ 1111\ 1111$$

$$+ 0010\ 0011\ 0100\ 1010\ 1010\ 1001\ 1011\ 1011$$

$$= 1110\ 1111\ 0100\ 1010\ 0110\ 0101\ 0100\ 0100$$

In the next round, we will have $L_2 = R_1$, which is the block we just calculated, and then we must calculate $R_2 = L_1 + f(R_1, K_2)$, and so on for 16 rounds. At the end of the sixteenth round we have the blocks L_{16} and R_{16} . We then *reverse* the order of the two blocks into the 64-bit block

$$R_{16}L_{16}$$

and apply a final permutation \mathbf{IP}^{-1} as defined by the following table:

\mathbf{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

That is, the output of the algorithm has bit 40 of the pre-output block as its first bit, bit 8 as its second bit, and so on, until bit 25 of the pre-output block is the last bit of the output.

Example: If we process all 16 blocks using the method defined previously, we get, on the 16th round,

$$L_{16} = 0100\ 0011\ 0100\ 0010\ 0011\ 0010\ 0011\ 0100$$

$$R_{16} = 0000\ 1010\ 0100\ 1100\ 1101\ 1001\ 1001\ 0101$$

We reverse the order of these two blocks and apply the final permutation to

$$R_{16}L_{16} = 00001010\ 01001100\ 11011001\ 10010101\ 01000011\ 01000010\ 00110010\ 00110100$$

$$\mathbf{IP}^{-1} = 10000101\ 11101000\ 00010011\ 01010100\ 00001111\ 00001010\ 10110100\ 00000101$$

which in hexadecimal format is

$$85\text{E}813540\text{F}0\text{A}\text{B}405.$$

This is the encrypted form of $\mathbf{M} = 0123456789\text{A}\text{B}\text{C}\text{D}\text{E}\text{F}$: namely, $\mathbf{C} = 85\text{E}813540\text{F}0\text{A}\text{B}405$.

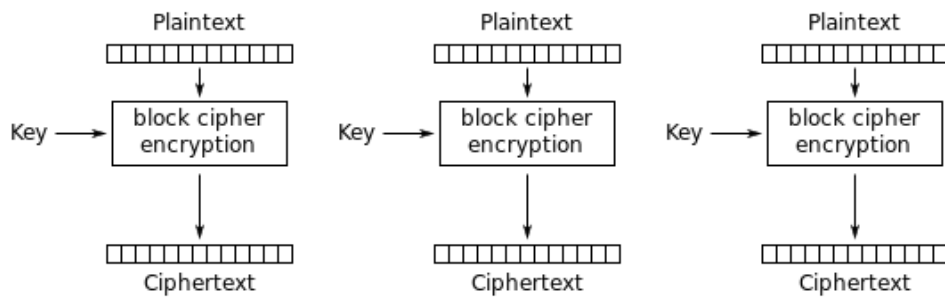
Decryption is simply the inverse of encryption, following the same steps as above, but reversing the order in which the subkeys are applied.

CHAPTER 5

DES Modes of Operation

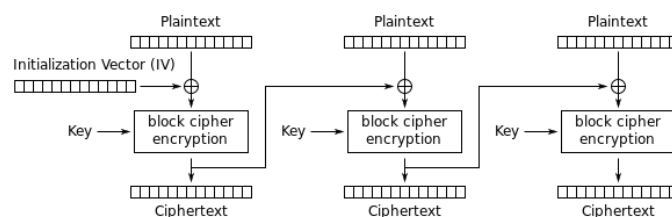
The DES algorithm turns a 64-bit message block **M** into a 64-bit cipher block **C**. If each 64-bit block is encrypted individually, then the mode of encryption is called **Electronic Code Book** (ECB) mode. There are two other modes of DES encryption, namely **Chain Block Coding** (CBC) and **Cipher Feedback** (CFB), which make each cipher block dependent on all the previous messages blocks through an initial XOR operation.

If each 64-bit is encrypted or decrypted independently, then this mode is ECB.

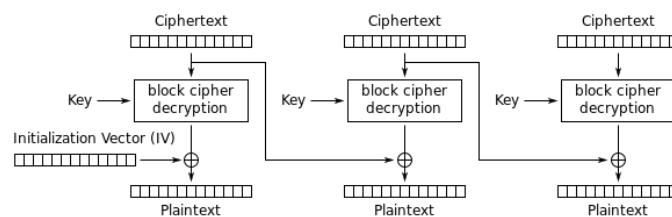


Electronic Codebook (ECB) mode encryption

If each 64-bit data is dependent on the previous one, then this mode is called CBC or CFB mode.

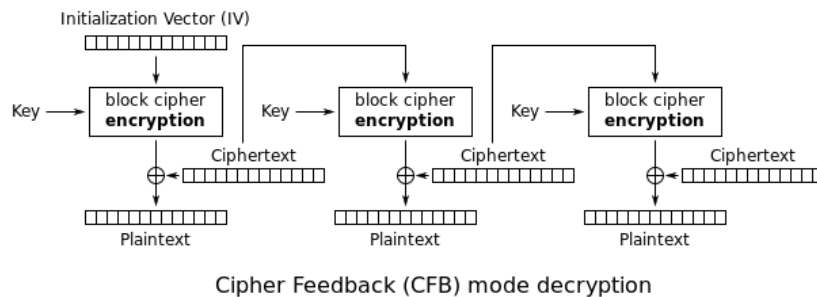
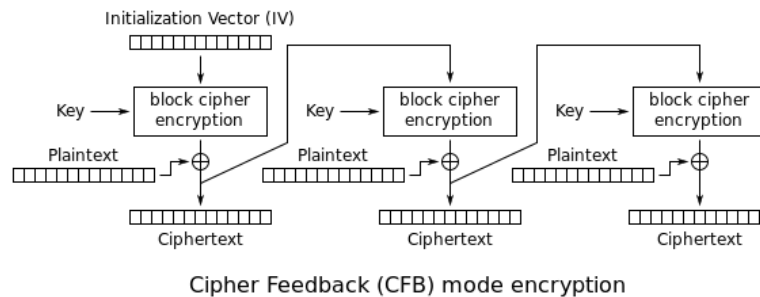


Cipher Block Chaining (CBC) mode encryption



Cipher Block Chaining (CBC) mode decryption

Here, in CBC mode, we can see that there is a new term called the Initial Vector (IV), which will be the same as the data block size. The data block will be XOR with IV and then encrypted with the key. Then, the output ciphertext uses the IV of the next block. The reverse process is used during decryption. So, we can say that the encryption of the current block is dependent on the encryption of the previous data block. Therefore, it's more secure than that of ECB.



In CFB mode, the initial vector is encrypted with a key, and then, the data block will XOR with encrypted output. Ciphertext, again, goes to it as an input for encryption function and again XOR with next plaintext block and so on.

CHAPTER 6

DES Analysis

Critics have used a strong magnifier to analyse DES. Tests have been done to measure the strength of some desired properties in a block cipher. The elements of DES have gone through scrutinise to see if they have met the established criteria. We discuss some of these in this section.

6.1 Properties

Two desired properties of a block cipher are the avalanche effect and the completeness.

Avalanche Effect *Avalanche effect* means a small change in the plaintext (or key) should create a significant change in the ciphertext. DES has been proved to be strong with regard to this property.

Example To check the avalanche effect in DES, let us encrypt two plaintext blocks (with the same key) that differ only in one bit and observe the differences in the number of bits in each round.

Plaintext: 0000000000000000 Key: 22234512987ABB23

Ciphertext: 4789FD476E82A5F1

Plaintext: 0000000000000000**1** Key: 22234512987ABB23

Ciphertext: 0A4ED5C15A63FEA3

Although the two plaintext blocks differ only in the rightmost bit, the ciphertext blocks differ in 29 bits.

This means that changing approximately 1.5 percent of the plaintext creates a change of approximately

45 percent in the ciphertext. Table 6.17 shows the change in each round. It shows that significant

changes occur as early as the third round.

Table *Number of bit differences*

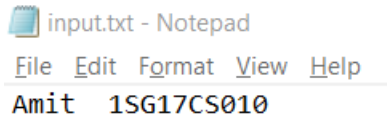
<i>Rounds</i>	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
<i>Bit differences</i>	1	6	20	29	30	33	32	29	32	39	33	28	30	31	30	29

CHAPTER 7

RESULT AND SCREENSHOTS

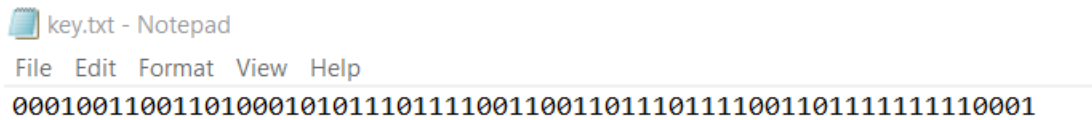
INPUT FILES TO THE PROGRAM –

input.txt- will contain our plain text (Max. Limit of plain text is 64kb).

A screenshot of a Notepad window titled "input.txt - Notepad". The menu bar shows "File", "Edit", "Format", "View", and "Help". The text content is "Amit 1SG17CS010".

```
input.txt - Notepad
File Edit Format View Help
Amit 1SG17CS010
```

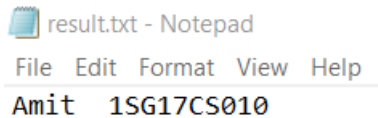
key.txt - will contain 64-bit key (take below key)

A screenshot of a Notepad window titled "key.txt - Notepad". The menu bar shows "File", "Edit", "Format", "View", and "Help". The text content is a 64-bit binary key: "0001001100110100010101110111100110011011101111001101111111110001".

```
key.txt - Notepad
File Edit Format View Help
0001001100110100010101110111100110011011101111001101111111110001
```

OUTPUT FILE –

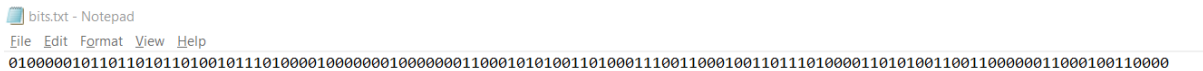
result.txt – it will contain our decrypted text.

A screenshot of a Notepad window titled "result.txt - Notepad". The menu bar shows "File", "Edit", "Format", "View", and "Help". The text content is "Amit 1SG17CS010".

```
result.txt - Notepad
File Edit Format View Help
Amit 1SG17CS010
```

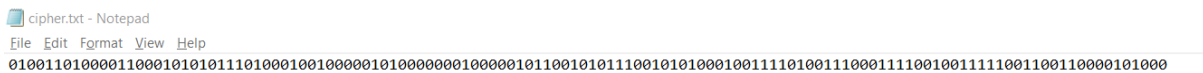
TEMP FILES –

bits.txt – it will contain our plain text converted in bits.

A screenshot of a Notepad window titled "bits.txt - Notepad". The menu bar shows "File", "Edit", "Format", "View", and "Help". The text content is the binary representation of "Amit 1SG17CS010": "010000010110110101101001111010000100000001000000110001010100110100011100110001001101110100001101010011001100000011000100110000".

```
bits.txt - Notepad
File Edit Format View Help
010000010110110101101001111010000100000001000000110001010100110100011100110001001101110100001101010011001100000011000100110000
```

cipher.txt – it will contain our encrypted text in bits.

A screenshot of a Notepad window titled "cipher.txt - Notepad". The menu bar shows "File", "Edit", "Format", "View", and "Help". The text content is the binary representation of the encrypted text: "0100110100001100010101011101000100100000010000010110010101110010101000100111101001110001111001001111001100110000101000".

```
cipher.txt - Notepad
File Edit Format View Help
0100110100001100010101011101000100100000010000010110010101110010101000100111101001110001111001001111001100110000101000
```

decrypted.txt – it will contain our decrypted text in bits (same as **bits.txt** in content)

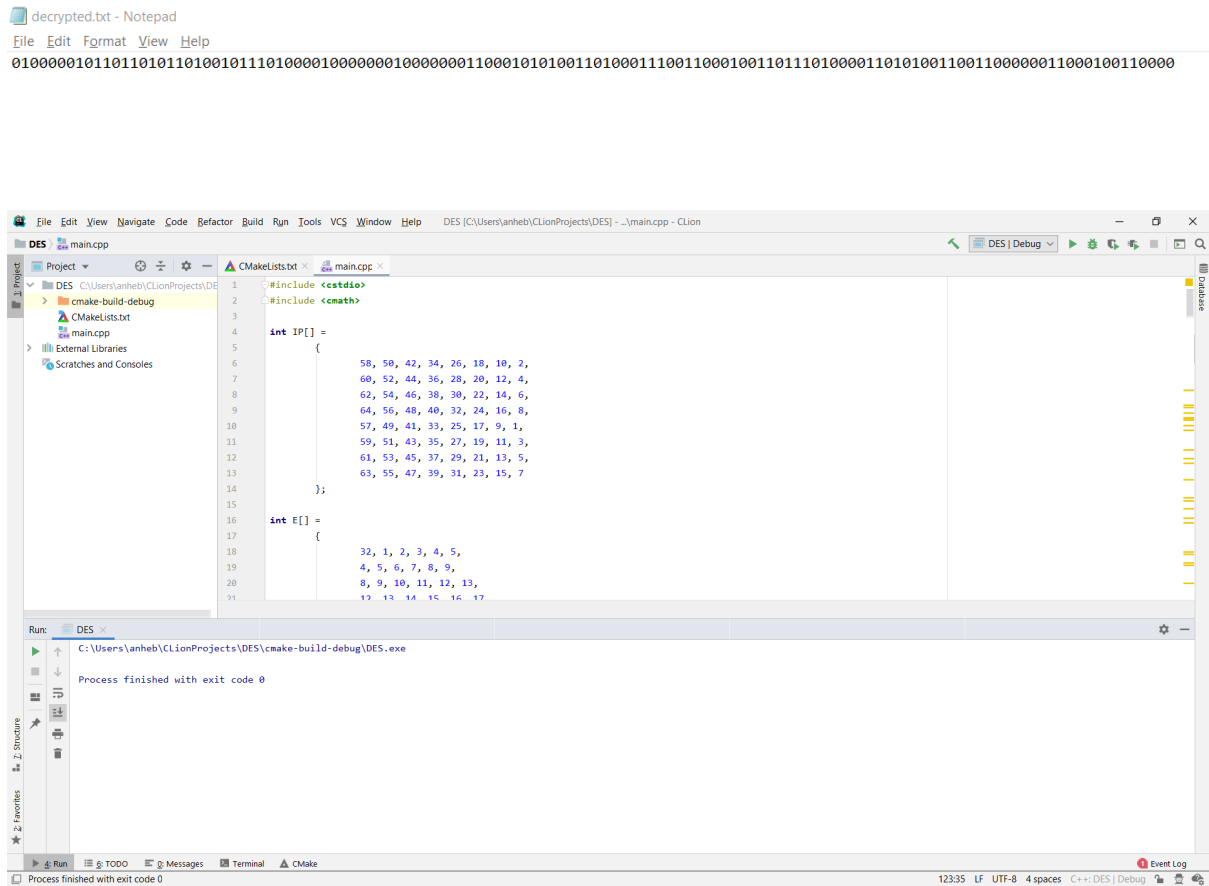


Figure 5.1 shows the output of the DES program run in CLion workspace platform

CHAPTER 8

DES Application

Data encryption (cryptography) is utilized in various applications and environments. The specific utilization of encryption and the implementation of the DES and TDEA1 will be based on many factors particular to the computer system and its associated components. In general, cryptography is used to protect data while it is being communicated between two points or while it is stored in a medium vulnerable to physical theft. Communication security provides protection to data by enciphering it at the transmitting point and deciphering it at the receiving point.

File security provides protection to data by enciphering it when it is recorded on a storage medium and deciphering it when it is read back from the storage medium. In the first case, the key must be available at the transmitter and receiver simultaneously during communication. In the second case, the key must be maintained and accessible for the duration of the storage period. FIPS 171 provides approved methods for managing the keys used by the algorithms specified in this standard. Public-key based protocols may also be used (e.g., ANSI X9.42).

- a. DES algorithm was made mandatory for all financial transactions by the U.S government which involves electronic fund transfer.
- b. High speed in ATM
- c. It is used for secure video teleconferencing
- d. Used in Routers and Remote Access Servers

8.1 Advantages and Disadvantages

Advantages:

- a. The use of 56-bit keys: 56-bit key is used in encryption, there are 256 possible keys. A brute force attack on such number of keys is impractical.
- b. The key needed for ciphering and deciphering purposes remains the same thus eradicating the burden of new key creation.
- c. Use of different ciphering and deciphering algorithm at two ends, makes it difficult for hackers to pass through using single strategy and requires them to have two different strategies for the two different ends.

- d. Its high-speed encryption and decryption capability, was one of the most important features for its use for over three decades
- e. It is unique because of being supported by most system, libraries and protocols
- f. Permits usage of same hardware or software for decryption as well, with same structure that is used for encryption.
- g. The keys are used in reverse order which means that second key is utilized first and first key is utilized last for decryption purpose.

Disadvantages:

- a. The algorithm is not suitable for enterprise level implementation due to its vulnerability to brute-force attack.
- b. It is slow in speed when compared to many other symmetric key encryption algorithms such as AES, Blowfish, RC4, RC5, and RC6.
- c. Weak keys: the key that is selected on the rounds are a problem. During splitting of keys to two half and swapping them might throw up the same result if they have continuous 1's and 0's. This ends up in using the same key throughout the 16-cycles.
- d. There can be same output from the S-Boxes on different inputs on permutation. These are called Semi weak keys.

8.2 Cracking DES

Before DES was adopted as a national standard, during the period NBS was soliciting comments on the proposed algorithm, the creators of public key cryptography, Martin Hellman and Whitfield Diffie, registered some objections to the use of DES as an encryption algorithm. Hellman wrote: "Whit Diffie and I have become concerned that the proposed data encryption standard, while probably secure against commercial assault, may be extremely vulnerable to attack by an intelligence organization" (letter to NBS, October 22, 1975).

Diffie and Hellman then outlined a "brute force" attack on DES. (By "brute force" is meant that you try as many of the 2^{56} possible keys as you have to before decrypting the ciphertext into a sensible plaintext message.) They proposed a special purpose "parallel computer using one million chips to try one million keys each" per second, and estimated the cost of such a machine at \$20 million.

Fast forward to 1998. Under the direction of John Gilmore of the EFF, a team spent \$220,000 and built a machine that can go through the entire 56-bit DES key space in an average of 4.5 days. On July 17, 1998, they announced they had cracked a 56-bit key in 56 hours. The computer, called Deep Crack, uses 27 boards each containing 64 chips, and is capable of testing 90 billion keys a second.

Despite this, as recently as June 8, 1998, Robert Litt, principal associate deputy attorney general at the Department of Justice, denied it was possible for the FBI to crack DES: "Let me put the technical problem in context: It took 14,000 Pentium computers working for four months to decrypt a single message . . . We are not just talking FBI and NSA [needing massive computing power], we are talking about every police department."

Responded cryptography expert Bruce Schneier: ". . . the FBI is either incompetent or lying, or both." Schneier went on to say: "The only solution here is to pick an algorithm with a longer key; there isn't enough silicon in the galaxy or enough time before the sun burns out to brute-force triple-DES" (*Crypto-Gram*, Counterpane Systems, August 15, 1998).

8.3 Triple-DES

Triple-DES is just DES with two 56-bit keys applied. Given a plaintext message, the first key is used to DES- encrypt the message. The second key is used to DES-decrypt the encrypted message. (Since the second key is not the right key, this decryption just scrambles the data further.) The twice-scrambled message is then encrypted again with the first key to yield the final ciphertext. This three-step procedure is called triple-DES.

Triple-DES is just DES done three times with two keys used in a particular order. (Triple-DES can also be done with three separate keys instead of only two. In either case the resultant key space is about 2^{112} .)

CHAPTER 9

CONCLUSION

As we toward a society where automated information resources are increased and cryptography will continue to increase in importance as a security mechanism. Electronic networks for banking, shopping, inventory control, benefit and service delivery, information storage and retrieval, distributed processing, and government applications will need improved methods for access control and data security. The information security can be easily achieved by using Cryptography technique. DES is now considered to be insecure for some applications like banking system. there are also some analytical results which demonstrate theoretical weaknesses in the cipher. So it becomes very important to augment this algorithm by adding new levels of security to make it applicable. By adding additional key, modified S-Box design, modifies function implementation and replacing the old XOR by a new operation as proposed by this thesis to give more robustness to DES algorithm and make it stronger against any kind of intruding. DES Encryption with two keys instead of one key already will increase the efficiency of cryptography.

GENERAL REFERENCES

"Cryptographic Algorithms for Protection of Computer Data During Transmission and Dormant Storage," *Federal Register* 38, No. 93 (May 15, 1973).

Data Encryption Standard, Federal Information Processing Standard (FIPS) Publication 46, National Bureau of Standards, U.S. Department of Commerce, Washington D.C. (January 1977).

Carl H. Meyer and Stephen M. Matyas, *Cryptography: A New Dimension in Computer Data Security*, John Wiley & Sons, New York, 1982.

Dorothy Elizabeth Robling Denning, *Cryptography and Data Security*, Addison-Wesley Publishing Company, Reading, Massachusetts, 1982.

D.W. Davies and W.L. Price, *Security for Computer Networks: An Introduction to Data Security in Teleprocessing and Electronic Funds Transfer*, Second Edition, John Wiley & Sons, New York, 1984, 1989.

Douglas R. Stinson, *Cryptography: Theory and Practice*, CRC Press, Boca Raton, 1995.

Bruce Schneier, *Applied Cryptography, Second Edition*, John Wiley & Sons, New York, 1996.

Alfred J. Menezes, Paul C. van Oorschot, and Scott A. Vanstone, *Handbook of Applied Cryptography*, CRC Press, Boca Raton, 1997.

Data Encryption Standard

https://www.tutorialspoint.com/cryptography/data_encryption_standard.htm. Accessed 19 Sep, 2019.

Data encryption standard (DES) | Set 1, <https://www.geeksforgeeks.org/data-encryption-standard-des-set-1>. Accessed 15 Sep, 2019.

Data Encryption Standard, https://en.wikipedia.org/wiki/Data_Encryption_Standard. Accessed 19 Sep, 2019.