Cryptanalysis of S-DES

Dr. K. S. Ooi
(ooi.kuansan@taylors.edu.my)

Brain Chin Vito
(v@chin.tc)

University of Sheffield Centre, Taylor's College

1st April 2002

Abstract

This paper describes an effort to attack S-DES using differential cryptanalysis and linear cryptanalysis. S-DES is a reduced version of the Data Encryption Standard (DES). It also includes a discussion on the subject of cryptology and a literature survey of useful papers regarding cryptography and cryptanalysis. This paper is meant as a tutorial on the fundamentals of differential cryptanalysis and linear cryptanalysis of a Feistel cipher.

Contents

1. Background

1.1 Security in the Information Age

The beginning of the Information Age heralded a new kind of threat to governments, corporations and individuals. Cyberspace had become the new arena of information exchange and commerce.Although highly beneficial, cyberspace can become a place for new forms of old crimes.

Along with the mainstream usage of cyberspace, governments and corporations are using cyberspace as a tool for economic espionage and infrastructure attacks. Electronic commerce crime on the other hand affects corporations, threatening the recent boom that has fueled the unprecedented economic recovery. Other forms of security threat do exist, for example: identity theft, cyber stalking and cyber terrorism [RP00]. These crimes expose individuals to financial, psychological, and even physical harm. **Figure 1.1** shows the sources of security threat.

Security is the main concerns of organizations participating in the information revolution. Cyber crime is very real and causes serious financial loss. **Figure 1.2** shows the total amount of financial losses caused by computer crime.

Even with billions of dollars spent on security systems, the problem of security breaches continues to rise throughout the years, encouraged by an even greater exposure of public civilians to security flaws and widespread of malicious tools (see **Figure 1.3**).

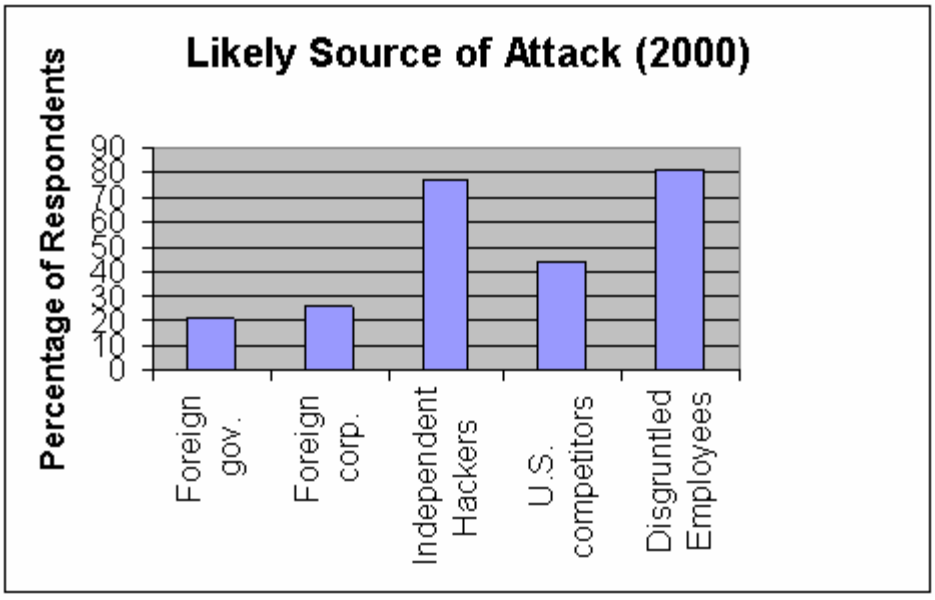


Figure 1.1
Source: 2000 CSI/FBI Computer Crime and Security Survey (from[RP00]*)*

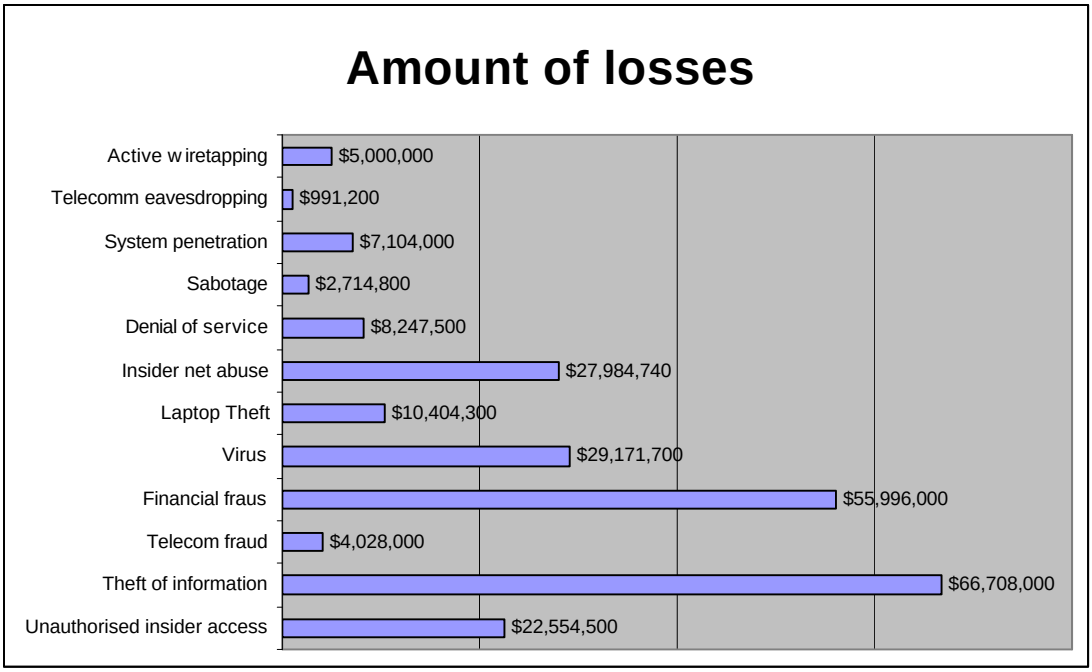


Figure 1.2
Source: 2000 CSI/FBI Computer Crime and Security Survey(from[RP00]*)*

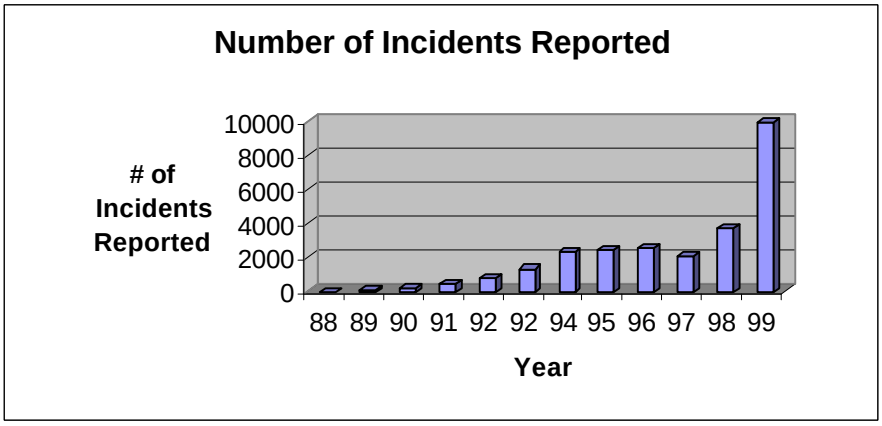


Figure 1.3
Source: CERT/CC Statistics 1998-1999(from*[RP00]*)

Historically, cryptography played an important role in many events, mostly wars, since the ancient Greek civilization [SS00]. The importance of cryptology is even more evidential in the information age, where commerce, military powers and governments depends so much on encryption technologies to create a secure environment for information exchange in unsecured channels.

1.2 Importance of Cipher Efficiency

Cryptographic systems are required in a wide range of applications. The internet explosion fuelled the constant need for good ciphers. Cryptography found its way into many applications such as e-mail verification, identity authentication, copyright protection, electronic watermark, biometrics and etc. At a lower level, cryptography are used to provide security for packet cables, embedded systems and many more [ER01]. Today, cryptographic systems are used in billions of devices worldwide, both wired and wireless.

Owing to its wide use, it is therefore desirable to design and implement efficient cryptographic systems that are robust in various platforms. Many factors are taken into consideration when gauging efficiency, these includes gate count for hardware design, memory requirements in software implementation and cross-platform performance.

The importance of cipher efficiency is evidential in the recent selection of Rijndael as the new AES [DR00]. It is chosen because it fulfills the security requirements set by the NIST with the least cost in hardware requirements compared to the other algorithms. Messages can be rapidly encrypted and decrypted with Rijndael and the performances are equally good across platforms ranging from smart cards to 64 bit processors. It also requires a modest amount of memory and performs the best (compared to other algorithms) when implemented in hardware.

1.3 Cryptology

Cryptology is a constantly evolving science; ciphers are invented and, given time, are almost certainly breakable. Cryptanalysis is the best way to understand the subject of cryptology. Cryptographers are constantly searching for the perfect security system, a system that is both fast and hard, a system that encrypts quickly but is hard or impossible to break. Cryptanalysts are always looking for ways to break the security provided by a cryptographic system, mostly though mathematical understanding of the cipher structure.

1.4 Why Cryptanalysis?

Recent advances in cryptanalytic techniques are remarkable. It is now considered essential for every paper of a new block cipher design to include a quantitative evaluation of security against state-of the-art cryptanalysis such as linear cryptanalysis and differential cryptanalysis.

Since the 1970s, cryptanalysis effort was centered on the cracking of the Data Encryption Standard (DES), a standard adopted by the National Institute of Standards and Technology (NIST). Recently, NIST adopted a new standard, termed the Advanced Encryption Standard (AES). NIST called for the public to submit ciphers that meet the standards set for the new AES. Most of the AES proposals submitted included results of linear cryptanalysis and differential cryptanalysis as an evaluation of the cipher strength. Most of the AES proposals stress the importance of cryptanalysis. This point is inline with the views of many cryptographers.

Bruce Schneier states that it is much more difficult to cryptanalyse a cipher than to design it [BS00]. He also mentions that cryptanalysis is the best way towards a concrete understanding of cryptographic technologies. Over 90 % of his efforts are spent on cryptanalysing when working on the proposal of the Twofish cipher. Most people have a conception that creating a good cipher system is difficult. It turns out that cryptanalysing a cipher is a much more difficult activity than creating one. To become a good cryptographer, the only route is through cryptanalysing. Only by cryptanalysing can a cryptographer really understand the inner-workings of a cryptographic system and to design systems which are stronger against attacks. This fact is clear if we observe cryptology books that are printed, while there are several good books on cryptography, there are no books, good or bad, on cryptanalysis [BS00]. This is due to the difficult nature of cryptanalysis and the fact that the only way to learn cryptanalysis is through practice. A cipher invented by someone who has shown that he can break algorithm is usually thought to be much secure. Design is easy and analysis is hard. Anyone can create an algorithm that he himself cannot break.

2.Literature Survey

2.1 Cryptology
2.2 Linear Cryptanalysis
2.3 Differential Cryptanalysis
2.4 Other Methods of Cryptanalysis
2.5 Programming and *C*

Literatures useful for research in block cipher cryptanalysis falls into one of the following gross categories:

2.1 Cryptology

Cryptology covers both cryptography and cryptanalysis. Although cryptology literatures are not as abundant compared to those of other fields, many of the literatures in existence are well-written. It is intriguing to see how the science of keeping secrets has evolved, especially in the information age.

- Bruce Scheneir
  - [BS96] is one of the most popular books on cryptography. Though not used extensively, it provides a good introduction to the various concepts involved in cryptography and some information on cryptanalysis.
  - [BS00] charts an organized pathway towards learning how to cryptanalyse.
  - [BS01] provides a good explanation on why cryptography is hard and the issues which cryptographers have to consider in designing ciphers.
- Bruce Scheneir et. al
  - [BS99] proposes a new cipher as a candidate for the new AES. By reading it, we get a grasp of the main concepts and issues involve in block cipher design. Also suggests cryptanalysis as a measure of cipher strength.
- Chan Yeob Yeun
  - [CY01] provides an encyclopedic look at the design, analysis and applications of cryptographic techniques. Good texts on number theory and abstract algebra.
- Edward Schaefer
  - [SC96] describes a simplified version of DES. It has the architecture of DES but has much lesser rounds and much lesser bits. The cipher is good for educational purposes.
- Fauzan Mirza
  - [FM00] is a very good paper on block cipher and cryptanalysis. Many of the foundational concepts required for this assignment is obtained here. A vital preliminary reading for anybody engaging in cryptanalysis.
- Howard Hayes
  - [HH00] has a somewhat similar notion as [FM00] but uses a more descriptive approach. Focuses on linear cryptanalysis and differential cryptanalysis of a given SPN cipher.
- Henning Schulzrinne
  - [SH00] is a series of slides on Cryptography. Particularly, it discusses DES-like ciphers.
- Joan Daemen,Vincent Rijmen
  - [DR00] describes Rijndael, the new AES. Provides good insight into many creative cryptographic techniques that increases cipher strength.
- Joe Kilian and Philip Rogaway
  - [KR01] investigates the security of Ron Rivest's DESX construction, a cheaper alternative to Triple DES.
- Lyndon G. Pierson
  - [PL00] is an interesting paper that compares modes of operations such as CBC, CFB, OFB and ECB.
- Kazumaro Aoki et. al.
  - [AM00] proposes a new cipher called Camelia that the creators claim be used for the next 10-20 years. An elegant paper which also includes cryptanalysis to demonstrate the strength of the cipher.
- Menezes,*P*. van Oorschot, S. Vanstone
  - [MP96] is the book that will be used for constant study and main reference throughout this project. A bible of cryptography.
- Paul Garrett
  - [PG01] is a textbook in cryptology which uses a highly mathematical path to explain cryptologic concepts. A good reference for mathematical concepts related to cryptology.
- Simon Singh

- [SS00] is a good book that provides an introduction to cryptology and the current interests in this field. An entertaining yet informative look at the history of cryptography, classical ciphers and the effect of cryptography on society.

- Susan Landau
  - [LS00] is an article on DES, the requirements of a good cryptosystem and cryptanalysis.

2.2 Linear Cryptanalysis

Linear cryptanalysis is about the approximation of an encryption algorithm with a linear equation. With the linear equation, we can then obtain key bits of the original secret key. It had been used extensively to attack DES.

- Buttyan, L. and I. Vajda.
  - [BV95] shows the correspondence between searching for the best characteristic in linear cryptanalysis and searching for the maximal weight path in a directed graph.
- Daemen et. al.
  - [DG94] propose the use of correlation matrix as a natural representation to understand and describe the mechanism of linear cryptanalysis.
- Eli Biham
  - [EB94] formalizes the method described in [MM94] and show that at the structural level, linear cryptanalysis is very similar to differential cryptanalysis. Used for further exploration into linear cryptanalysis.
- Harpes, C., G. Kramer and J. Massey.
  - [HG95] provides a generalization of linear cryptanalysis and suggests that IDEA and SAFER *K*-64 are secure against such generalization.
- Kaliski and Robshaw
  - [KR94] surveys the use of multiple linear approximations in cryptanalysis to improve efficiency and to reduce the amount of data required for cryptanalysis in certain circumstances.
- Mitsuru Matsui
  - [MM94] is the original paper on linear cryptanalysis. This paper is of substantial importance to the project. A must read for all cryptanalysts.
  - [MM941] is an improved version of [MM94]. It discusses an experimental attack of DES using 12 computers.
- Pascal Junod
  - [PJ98] describes an implementation of Matsui's linear cryptanalysis of DES with strong emphasis on efficiency. An experimental undertaking.

2.3 Differential Cryptanalysis

Differential cryptanalysis is a very general tool against iterated block ciphers. It involves the examination of the input and output difference, called differential characteristics. Used to cryptanalyse many ciphers including DES.

- Anne Canteaut
  - [AC97] describes the design of a Feistel cipher with at least 5 rounds that is resistance to differential cryptanalysis.
- C. Adams
  - [CA92] explores the possibility of defeating differential cryptanalysis by designing S-boxes with equiprobable output XORs using bent functions.
- Dawson and Tavares
  - [DT91] describes some design criterias for creating good S-boxes that are immune to differential cryptanalysis. These criterias are based on information theoretic concepts.
- Eli Biham and Adi Shamir

- [EA90] is the original paper on differential cryptanalysis. Introduces differential cryptanalysis on a reduced round variant of DES. Very useful for the project
- [EA91] breaks a variety of ciphers, the fastest break being of two-pass Snefru.
- [EA92] describes the cryptanalysis of the full 16-round DES using an improved version of [EA90].

- James Massey and Xuejia Lai
  - [LM91] introduces the concept of Markov ciphers and explain it's significance in differential cryptanalysis. Investigates the security of iterated block ciphers and shows how to when an r-round cipher is not vulnurable to attacks.
- K. Nyberg
  - [NK91] shows that there are DES-like iterated ciphers that does not yield to differential cryptanalysis.
- Sean Murphy and M.J.B. Robshaw
  - [SM00] proposes that eight round Twofish can be attacked. Investigates the role of key dependent S-boxes in differentaial cryptanalysis.
- Youssef, A., S. Tavares.
  - [YT95] is on the same line with [CA92] but proposes that the input variables be increased and that the S-box be balanced to increase resistance towards both differential and linear cryptanalysis.

2.4 Other Methods of Cryptanalysis

Many other cryptanalytic method exists but this project focuses on linear and differential cryptanalysis. Nevertheless, the following literatures mostly describes extensions of linear or differential cryptanalysis (except Square) which are useful in this project.

- Anne Gorksa et. al.
  - [AG00] describes analysis with multiple expressions and differential-linear cryptanalysis. Also provides experimental results of an implementation of differential-linear cryptanalysis with multiple expressions applied to DES variants.
- Martin Hellman and Susan Langford
  - MS94] is a paper describing the Differential-Linear cryptanalysis method, a very efficient method against DES in terms of required texts. This method may possibly be look into in later stages of the project
- Niels Ferguson et. al.
  - [NF01] is an attack on 7 and 8 round Rijndael using the Square method. Also describes a related-key attack that can break 9 round Rijndael with 256 bit keys.
- Serge Vaudenay and Shiho Moriai
  - [VM95] discusses the relations between linear and differntial cryptanalysis and presents classes of ciphers which are resistant towards these attacks.
- Vaudenay, S.
  - [VS95] is a paper describing a statistical cryptanalysis of DES, a combination and imrovement of both linear and differential cryptanalysis. Suggests that the linearity of S-boxes are not very important.

2.5 Programming and *C*

C is the common standard for the implementation of ciphers. C is an extremely flexible, expressive and terse language. This flexibility can either make or break the robustness of a cipher, depending on the correctness of implementation. This in turn lies greatly on the hands of the programmer. Thoughtful programming must be employed. Efficiency is also an important consideration.

- Matthew Kwan
  - [MK98] is an implementation of DES but with bit slicing. It shows how the bit-sliced implementation is more efficient.

- Mark Allen Weiss
  - [MW95] is a comprehensive book on C. Good reference text during the process of coding.
- Ooi Kuan San
  - [OS00] is a tutorial on using pointers in C++. Begins with a brief introduction to computer architecture and discusses many issues such as programming practices and even operator precedency.
  - [OS01] is an important tutorial for the purpose of this project as it teaches bit manipulation, which is prerequisite knowledge for implementing ciphers.

3.S-DES

3.1 Introduction
3.2 The Keys
3.3 The Cipher

3.1. Introduction

S-DES is a reduced version of the DES algorithm. It has similar properties to DES but deals with a much smaller block and key size (operates on 8-bit message blocks with a 10-bit key). It was designed as a test block cipher for learning about modern cryptanalytic techniques such as linear cryptanalysis, differential cryptanalysis and linear-differential cryptanalysis. It is a variant of Simplified DES [SC96].

The same key is used for encryption and decryption. Though, the schedule of addressing the key bits are altered so that the decryption is the reverse of encryption. An input block to be encrypted is subjected to an initial permutation ***IP.*** Then, it is applied to two rounds of key-dependent computation. Finally, it is applied to a permutation which is the inverse of the initial permutation. We shall now proceed to a detailed description of the components of S-DES.

3.1.2. The Keys

The 10 bit key is used to generate 2 different blocks of 8 bit subkeys where each block is used in a particular iteration. Let us denote the 10 bit key as *KEY*, the 8 bit subkeys as K_1 and K_2. The key-schedule used to generate the subkeys is denoted as *KS*.

Figure 3.1 illustrates the calculation of K_1 and K_2 given *KEY*. *KEY* is subject to an initial permutation, Permuted Choice 1 which is determined by the following table:

PC-1

9	7	3	8	0
2	6	5	1	4

The table has been divided into two parts. The upper part determines the bits of C_0 and the bottom part determines the bits of D_0 .The bits of *KEY* are numbered from 0 to 9. Thus, the bits of C_0 are bits 9, 7, 3… of *KEY* and the bits of D_0 are bits 2, 6, 5… of *KEY*.

A single left shift is then performed on both C_0 and D_0. The result of a single left shift of C_0 and D_0 is C_1 and D_1. To form K_1, D_1 is concatenated to C_1 (with the with the most significant bit of C_1 as the most significant bit of K_1, and the most significant bit of D_1 following the least significant bit of C_1) and then subjected to a permutation, Permuted Choice 2 which is determined by the following table:

PC-2

3 1 7 5 0 6 4 2

Thus, the first bit of K_1 is the third bit of C_1D_1. Notice that the bits of K_1 are only 8 bits.

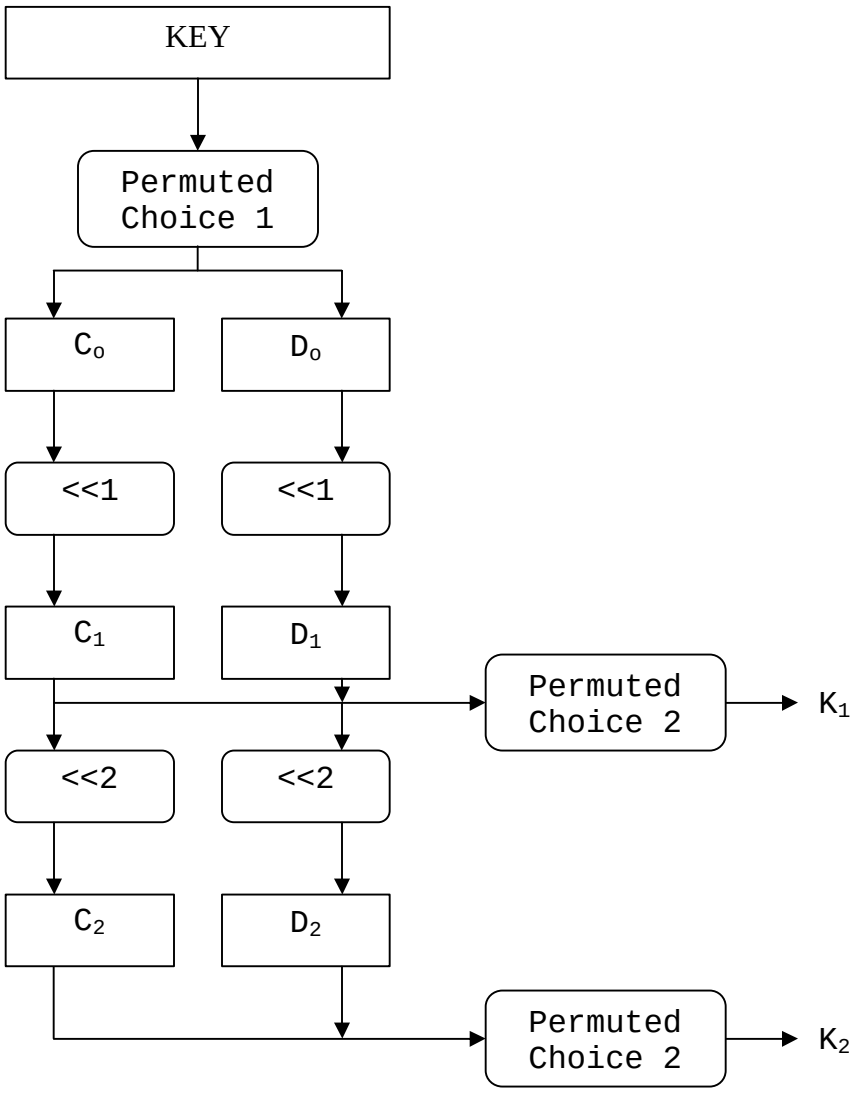


Figure 3.1

To form K_2, C_1 and D_1 is subjected to two left shifts yielding C_2 and D_2. Then, D_1 is concatenated to C_1 yielding C_1D_1. C_1D_1 is then subjected to a permutation, Permuted Choice 2 as described above, yielding K_2.

3.3 The Cipher

Enciphering

Figure 3.2 describes the computation required for enciphering. The encryption procedure can be summarized as:

$$C = E(P, K) = IP^{-1}(r_2(r_1(IP(P))))$$

We now begin to look at each stage of the algorithm in detail.

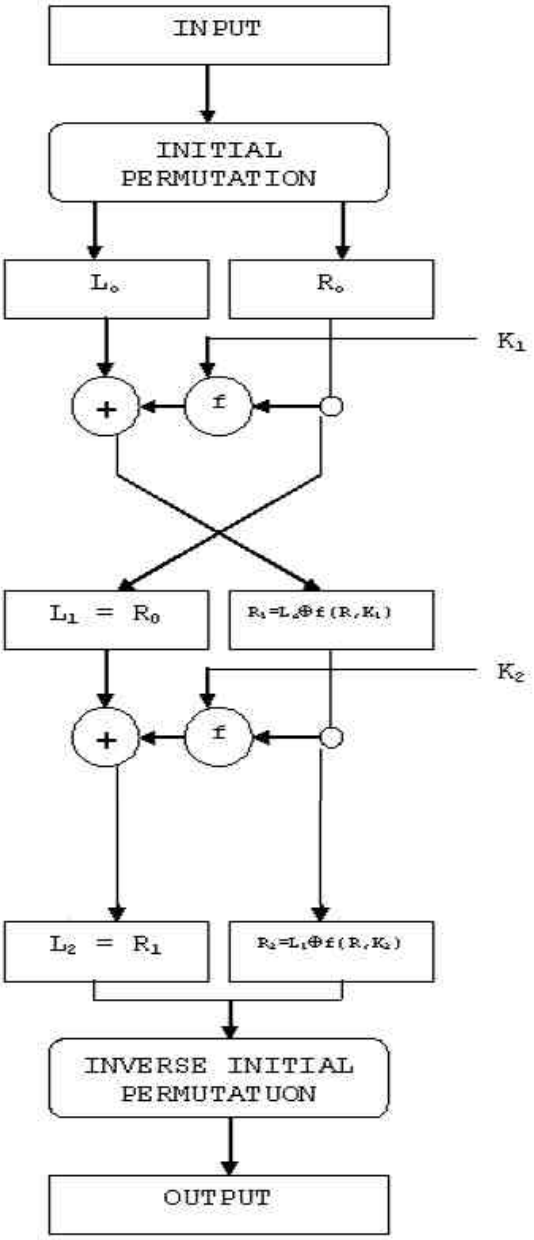


Figure 3.2

The 8-bit block is subjected to an initial permutation, *IP 1*, which is as follows:

IP 1

7	6	4	0
2	5	1	3

The table has been divided into two parts. The upper part determines the bits of L_0 and the bottom part determines the bits of R_0.The bits of INPUT are numbered from 0 to 7. Thus, the bits of L_0 are bits 7, 6, 4… *of* INPUT and the bits of R_0 are bits 2, 5, 1… of INPUT.

After the initial permutation, L_0 and R_0 is then subjected to round 1. The output of round 1 is L_1 and R_1. The calculation is as follows:

$L_1 = R_0$
$R_1 = L_0 \oplus f(R, K_1)$

The key-dependant cipher function *f*, will be described later.
L_1 and R_1 is then applied to round 2, yielding L_2 and and R_2 as calculated below:

$L_2 = R_1$
$R_2 = L_1 \oplus f(R, K_2)$

The last step requires the concatenation of R_2 to L_2 which yields L_2R_2. This is then subjected to a permutation which is the inverse of the initial permutation. After this permutation, OUTPUT is produced.

The Cipher Function f

A sketch of the calculation of ***f(R,K)*** is given in **Figure 3.3**

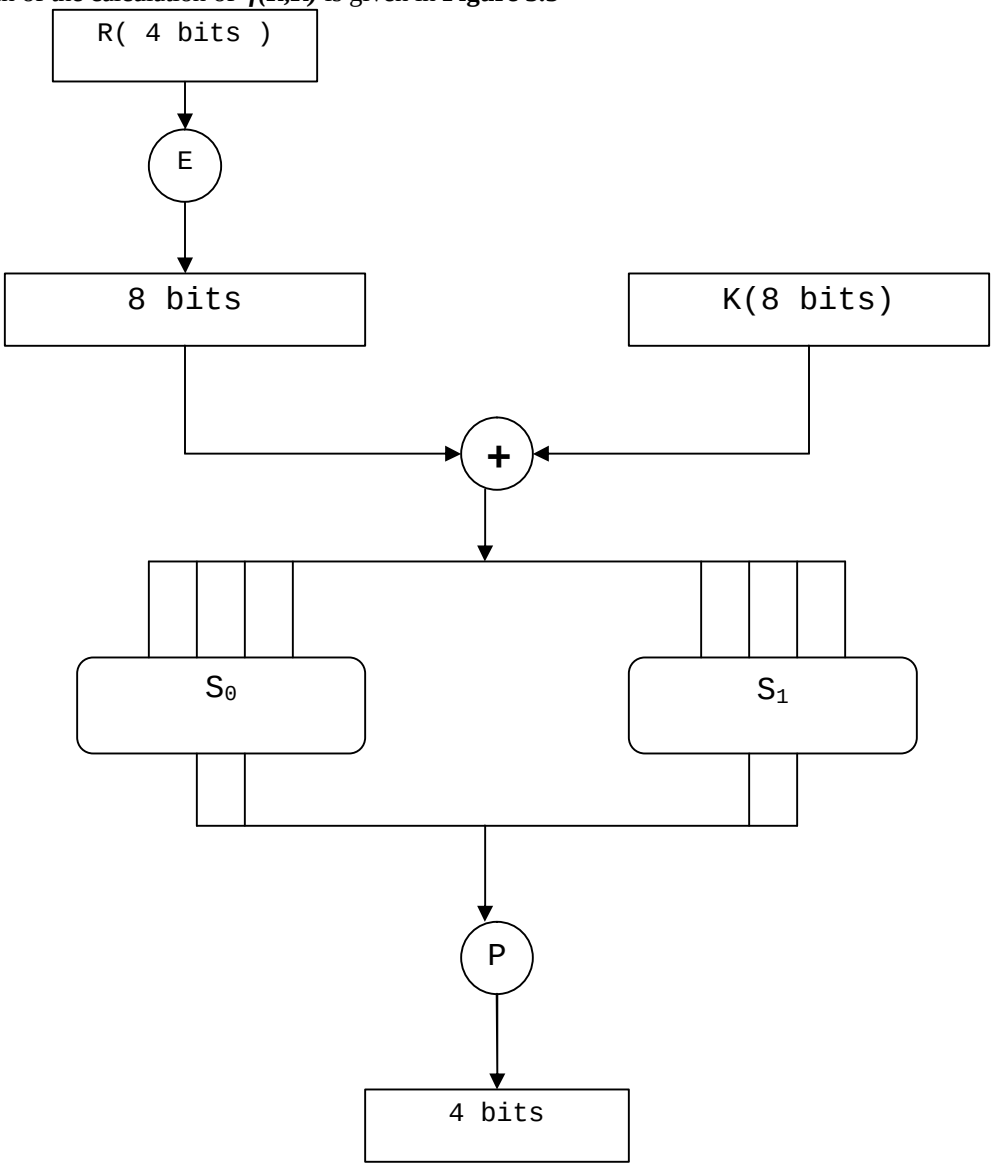


Figure 3.3

E denotes a function which takes in a 4 bit block input and yields a 8 bit block as output. The 8-bit output block of *E* is obtained according to the following table:

E-BIT SELECTION TABLE

3	0	1	2	1	2	3	0

Thus, the first three bits of $E(R)$ are bits 3, 0, 1 of *R*.
The 8 bit $E(R)$ is then XORed with the 8 bit subkey *K*. The subkey K_1 is used for round 1 and K_2 is used for round 2.

The result of the XORing operation is then split into two blocks, the first four bits from the most significant bit being B_1 and the remaining bits being B_2. B_1 and B_2 are then applied to S_0 and S_1 respectively.

S_0 and S_1 are S-Boxes which take in a 4 bit input and yield a 2 bit output.

	S_0 Column Number			
Row No.	0	1	2	3
0	1	0	2	3
1	3	1	0	2
2	2	0	3	1
3	1	3	2	0

	S_1 Column Number			
Row No.	0	1	2	3
0	0	3	1	2
1	3	2	0	1
2	1	0	3	2
3	2	1	3	0

We take S_0 as an example to illustrate how the output block is determined. S_0 takes the first bit and the last bit of the 4-bit block and used them to represent in base 2 a number in the range of 0 to 3. For example, for a block of bits 1101, 11 are obtained and are subsequently converted to 3. This is used to determine the row, in this case, row 3.

The middle 2 bits is used to represent in base 2 a number in the range of 0 to 3. This is used to determine the column. In the case of our example block, the two bits in the middle represents column 2.

From S_0 above, a number from row 3 column 2 is selected, thus yielding the number 2, which in binary is written as 10.

The result of S_0 and S_1 are concatenated to form a four bit block which is then applied to a permutation, *P*. This function is defined by the following table:

P

1	0	3	2

The result of *P* will be the 4 bits returned by the function *f*.

Deciphering

To decipher, the same algorithm is used, but the subkeys are applied in reverse order. That is, instead of applying K_1 for round 1,K_2 is used instead. For round 2, K_1 is used instead.

4. Brute Force Attack

Brute force cryptanalysis, as its name suggests is the most straightforward cryptanalysis attack. Although being one of the most primitive methods of attacking a cipher is gaining increasing applicability as a result of increasing computational power.

Though simple, brute force is only practical for cryptosystems with key size of maximum 56 bits (over 2^{56} = 72 quadrillion tries) [MC98] . Other cryptosystems with a key size larger than 56-bits may require a long time to crack with brute force and may not even be feasible. For example, the current AES with a 128-bit block size is almost impossible to crack by brute force with today's general purpose computers [DR00] . The number of combinations to try is more than the grains of sands on earth, many times more than a billion for every square meter on earth [DH01].

It is feasible to attack S-DES with brute force since it only has a key size of 10 bits. To perform brute force attack, we need to have a plaintext-ciphertext pair in which we search the key space until the appropriate plaintext, encrypted with the targeted key yields the ciphertext. **Appendix 8.2** shows the code and **Appendix 8.5** shows the result of the attack.

5. Differential Cryptanalysis

Differential cryptanalysis is a chosen plaintext/chosen ciphertext attack that was initially developed to attack DES-like ciphers [EA93] . A chosen plaintext attack is one where the attacker is able to select inputs to a cipher and examine the output. Being one of the earlier attacks on DES, differential cryptanalysis had been studied extensively [FM00]. Many of today's ciphers are designed with consideration to immunity against differential cryptanalysis. Nevertheless, differential cryptanalysis still provides a good understanding of the possible weakness of ciphers and techniques to overcome them.

Differential cryptanalysis involves the analysis of the effect of the plaintext pair difference on the resulting ciphertext difference. The most common difference utilized is the fixed XORed value of the plaintext pairs. By exploiting these differences, the partial subkey used in the cipher algorithm can be guessed. This guess is done statistically by using a counting procedure (Sect. 5.5) for each key in which the key with the highest count is assumed to be the most probable partial subkey.

5.1 Basic Concept

Consider the following basic linear cipher function:

$$C = P \oplus K$$

By taking the difference of a pair of ciphertext, we would have cancelled out the key involved, leaving us with no information about the key:

$$C \oplus C' = P \oplus K \oplus P' \oplus K$$
$$C \oplus C' = P \oplus P'$$

This is because of the linearity of the function. The above equation simply tells us that the difference between the plaintext is the same as the difference between the ciphertext [FM00].

S-DES is not a linear cipher. Thus, the difference between the ciphertext is not equal to the difference between the plaintexts. In S-DES, the difference in a ciphertext pair for a specific difference of

a plaintext pair is influenced by the key. Thus, by utilising this fact, and the knowledge that certain plaintext differences occurs with a higher probability than other differences, we can reveal information about the key [EA90] . Like linear cryptanalysis, we start by analysing the non-linear component of the cipher, the S-Box. Then, we extend the values obtained to form a complete differential characteristic sufficient to perform an attack.

5.2 Difference Pairs of an S-Box

Consider the S-Box S_0 and S_1 of S-DES. We denote the input to the S-Box as X and the output as Y [HH00]. The difference pairs of an S-Box is then denoted as (DX, DY), where $DX = X' \oplus X''$. It is more convenient if we consider all 16 values of X' with ΔX as a constraint to the value of X'', thus $X'' = X' \oplus \Delta X$. With X' and X'', the value of DY can then be obtained.

The table shown in **Appendix 8.8** shows all the difference pairs of S_0 and **Appendix 8.9** shows the difference pairs of S_1.

5.3 Difference Distribution Table

The tables shown in **Appendix 8.10** and **Appendix 8.11** are the difference distribution tables for S_0 and S_1 where the row represents DX value, the column represents DY values and the elements represents the number of occurences given the column and row values.

Interesting things can be performed with the availability of the difference distribution table . Two of the possibilities are:

1. We can obtain the possible input and output values given their differences [EA90]

   This is done by checking the corresponding value of the input and output differences in the difference distribution table. Consider the following input and output difference DX=8 and DY=1 of S_0. From the difference distribution table, we see that the number of occurences for this input and output difference is 2 so only two pairs can satisfy this difference. Further, we see that these pairs are duals. If the first pair is X', X'', then the other pair is X'', X'.

   Since DY is 1, then the output pairs must be 1 and 3. Subsequently, we find that the only input pairs that can yield 1 and 3 as output pairs and at the same time satisfy the input difference DX=8 is 9 and 1 respectively.

2. We can obtain the key bits involved in the S-Box using known input pairs and output differences of the S-Box [EA90]

   Given a particular input pair, we can obtain the possible key bits involved in the S-Box. Assuming X'= 2, X''= 8 and the S-Box considered is S_0. Then, Y'=0, Y''=2 and DY = 2. We denote the inputs to the S-Box after XOR-ing with the key as $I'=X' \oplus K$ and $I''=X'' \oplus K$. From [HH00], we know that the key has no influence on the input difference value. So, $DX = DI = 2 \oplus 8 = 10$.

   Now that we've obtained DX= 10 and DY = 2 we can proceed to obtain the key bits involved. From the distribution table, we see that DX=10 and DY=2 has two possibilities. This implies that there are 2 possibilities for the key. **Table 5.1** lists the keys and the corresponding X' and X''. Since DI=10, then the pairs of I that can satisfy this difference is 7 and 13.

   Given that $K = X \oplus I$, the first possible key, 5 is obtained from:

$$K = I' \oplus X' = 7 \oplus 2 = 5 \text{ and } K = I' \oplus X'' = 7 \oplus 8 = 5$$

   and the second key 15 is obtained from

$K = I'' \oplus X' = 13 \oplus 2 = 15$ and $K = I'' \oplus X'' = 13 \oplus 8 = 15$

S-Box Input	Possible Keys
7 13	5 15

Table 5.1 Possible Keys for X'=2 and X''=8

5.4 Differential Characteristics

The above example is only an introduction to the possibilities that are available to us when we analyse the difference between plaintext pairs and ciphertext pairs. We extend this knowledge to create a differential characteristic for 1 round of S-DES [HH00]. With this differential characteristic, we can obtain the subkey, K_2 used in the last round. First, we construct a differential characteristic that involves S_1 in both rounds of S-DES using the following difference pair of S_0 and S_1:

S_0: $DX_0 = 2 \rightarrow DY_0 = 2$ with probability 12/16
S_1: $\Delta X_1 = 4 \rightarrow DY_1 = 2$ with probability 10/16

Thus, by considering DX_0, DX_1 and the expansion, E which me modify to E = [0 2 1 3 0 1 2 3], the input difference to the first round is given by:

$$DU_1 = [0\ 0\ 0\ 0\ 0\ 1\ 0\ 0]$$

The expansion E is only a form of "diffusion sugaring" and does not add to the non-linearity of the cipher. The change is to make the derivation of the input difference clearer, expansion is not the main concern here.
Then, considering DY_0 and DY_1 and the permutation P that follows, we get the output difference for round 1:

$$DV_1 = [0\ 1\ 0\ 0\ 0\ 1\ 0\ 1]$$

This 1-round characteristic holds with probability $12/16 \times 10/16 = 15 / 32$, which means that for every 32 random and uniformly distributed pairs of chosen plaintexts with difference DU_1 we expect to find about 1 pair of corresponding ciphertexts which satisfies the difference DV_1. Pairs with plaintext that produces DU_1 and corresponding ciphertexts that produces DV_1 are called right pairs.

The differential characteristic can be best visualised using the figure used by Biham [EA90]. The visualisation of the differential characteristic for round 1 is shown in **Appendix 8.12.**

For a N-round cipher, we need to find the differential characteristic of N-1 round [HH00] [FM00] to conceive an attack. Since S-DES is a 2-round cipher, the 1-round characteristic which we obtained is sufficient.

5.5 Extracting the partial subkey

With the differential characteristics obtained above, we can now proceed to extract subkey K_2 of round 2. We shall call the subkey that we want to extract as the target subkey [HH00]. The process to extract the subkey is described algorithmically as follows:

1. Obtain a random plaintext P' and compute $P'' = P' \oplus DX$.
2. For P' and P''
    a. Encrypt both P' and P'' with both K_1 and K_2 to obtain C' and C''. Also, obtain CI' and CI'', the encrypted plaintext after one round, which is encrypted with K_1 only .
    b. If $CI' \oplus CI'' = DY$, then
        i. For all possible subkey values, encrypt $CI' \oplus CI''$ with only one round, which is round 2.

ii. If the result of the encryption of 2.b.i. is equivalent to those suggested by *C*' and *C*'', then we increase the count for the corresponding subkey value used.

3. Repeat 1 and 2 for another random plaintext *P*'. Perform this until one value of the subkey has been counted to be substantially more than the others. This subkey value is assumed to be the correct subkey value used in the second round.

5.6 Conclusion

Using the parameters discussed, the attack is performed on S_0 and S_1 and the results are listed in **Appendix 8.5.** The attack was very successful and was able to extract the entire subkey of round 2. These subkey bits are the actual 8 bits of the S-DES 10 key bits, 2 bits are still missing. Thus, we can now try all the 2^2 possibilities of the missing bits which are trivial. Also, with that subkey, the whole key can then be derived without trying the 2^2 possibilities, the method to perform this is explained in [EA90].

The source code for the entirely automated differential cryptanalysis is listed in **Appendix 8.3**

7 Linear Cryptanalysis of S-DES

6.1 Linear Cryptanalysis Principles

Linear cryptanalysis is a known-plaintext attack that is one of the most commonly used attack against block ciphers. It was invented by Mitsuru Matsui and is used initially to attack the Data Encryption Standard [MM94]. It is based on the fact that there are high probability of occurences of linear expressions consisting the plaintext bits, ciphertext bits and key bits.

The main idea behind linear cryptanalysis is to obtain an approximation to the block cipher as a whole using a linear expression. This linear expression has the form:

$$(1)\quad \left(\bigoplus_{a=1}^{u} X_{i_a}\right) \oplus \left(\bigoplus_{b=1}^{v} Y_{j_b}\right) = \left(\bigoplus_{g=1}^{w} K_{k_g}\right)$$

where *X* denotes plaintext bits, *Y* denotes ciphertext bits and *K* denotes the key bits. The indices u,v and w denote fixed bit locations.

The goal is to find the linear expression which holds with the highest/biggest linear probability bias. The linear probability bias, *e* is defined as

$$(2)\quad e = \left| p - 1/2 \right|$$

which is the magnitude of the bias from a probability of ½. The higher the magnitude of the bias, the higher the efficiency of the linear expression (1).

If equation (1) holds with a high probability bias, it means that the cipher used is not sufficiently random [HH00]. A cipher is considered to be random if the randomly selected value of the bits of its linear expression would cause the expression to hold with a probability of ½. Thus, the further away a linear expression is from holding with a probability of 0.5, the less random it is. Linear cryptanalysis takes advantage of this poor randomization.

1111

Once a linear approximation with the highest bias is obtained, the attack can be mounted by recovering a subset of the key bits. This is done using Algorithm 1 as described in [MM94].

6.2 Obtaining a linear equation with high probability bias

To obtain a linear equation with high probability bias, we begin by constructing a statistical linear path between the input and output bits of each S-box. We then extend this path to the entire cipher and will finally reach a linear expression without any intermediate value[MM94].

6.2.1 Linear Approximation of S-Boxes

Block ciphers commonly use non-linear operations in its S-boxes. Though, it is possible to construct a linear approximation of S-Boxes. Techniques for this purpose are described in [RR86].

The goal is to find the linear approximation with the highest bias magnitude. Following Matsui's notation,

Definition 6.1 *For a given S-box S_a (a = identifier of the S-box), $1 \le a \le p$ and $1 \le b \le q$, we define $NS_a(a,b)$ as the number of times out of p input patterns of S_a, such that a XORed value of the input bits masked by a matches with an XORed value of the output bits masked by b*

$$(3) \qquad NS_a(a,b) \underline{\underline{def}} \#\left\{x \mid 0 \le x \le p, \left(\bigoplus_{s=0}^{y-1}(x[s] \bullet a[s])\right) = \left(\bigoplus_{t=0}^{z-1}(S_a(x)[t] \bullet b[t])\right)\right\}$$

where y is the number of input bits and z is the number of output bits.

The results of this process can be enumerated in a linear approximation table, where the vertical and the horizontal axes represents a and b respectively. Each element of the table represents $NS_a(a, b) - (y + z)$. From the table, the linear approximation with the highest bias magnitude can be identified.

We shall now proceed to derive the linear approximation for S_0. The table in **Appendix 8.13** shows the input to the S_0 and the corresponding output. We shall call this table the S_0 I/O table.

The table helps us to obtain the probability for a particular value of a and b. For example:

$$NS_1(1,1) = \#\left\{x \mid 0 \le x < 16, \bigoplus_{s=0}^{3}(x[s] \bullet 1[s])) = \bigoplus_{t=0}^{1}(S_0(x)[t] \bullet 1[t]))\right\}$$

$$(4) \qquad = \#\{x \mid 0 \le x < 16, (X_0 = Y_0)\}$$

$$= 8$$

where X_i denotes the column to be considered as listed in the table in **Appendix 8.13** .

The above equation means that we compare the row X_0 and the row Y_0 to obtain the number of times where the element of a particular row of X_0 equals that of the same row of Y_0.

Table 6.1 shows a portion of the distribution table of S-box S_0, where the row represents a and the column represents b and the elements shows $NS_0(a,b)$ - 8. The table in **Appendix 8.15** shows the full distribution table. Column =b

a	b		
	1	**2**	**3**
1	0	0	0
2	0	0	-2
3	0	-2	-2

4	-2	2	0
5	6	2	0
6	-2	0	-2

Table 6.1

The most effective linear approximation is the one with the highest magnitude. We choose $NS_0(5,1)$ since $|NS_0(5,1)-8|$ is one of the highest in the table.

$$NS_0(5,1) = \#\left\{x \mid 0 \le x < 16, \bigoplus_{s=0}^{3}(x[s] \bullet 5[s]) = \bigoplus_{t=0}^{1}(S_0(x)[t] \bullet 1[t])\right\}$$

(5) $$= \#\{x \mid 0 \le x < 16, (X_2 \oplus X_0 = Y_0)\}$$

$$= 14$$

Thus, the linear approximation for S_0 is $X_2 \oplus X_0 = Y_0$ which holds with probability $\frac{14}{16}$.

Appendix 8.14 shows a visualisation of the above linear approximation of S_0

6.2.2 Linear Approximation of the F-Function

The linear approximation of the *f*-function can be obtained by taking into account the expansion *E* and the permutation *P*. We extend our equation from the previous section to obtain:

(6) $$R_0 \oplus R_2 \oplus f(R_1, K_1) = K_1 \oplus K_3$$

6.2.3 Linear Approximation of the entire algorithm

The entire algorithm consists of two rounds. We first apply equation (6) to the first round to get the following equation:

(7) $$R_0^1 \oplus L_0^0 \oplus R_0^0 \oplus R_2^0 = K_1^1 \oplus K_3^1$$

The equation for the second round is:

(8) $$L_0^1 \oplus L_0^2 \oplus R_0^1 \oplus R_2^1 = K_1^2 \oplus K_3^2$$

Having obtained equation (7) and (8), we can now derive a linear approximation of the entire algorithm by canceling out common terms, which is:

(9) $$L_0^0 \oplus L_0^1 \oplus L_0^2 \oplus R_0^0 \oplus R_2^0 \oplus R_2^1 = K_1^1 \oplus K_3^1 \oplus K_1^2 \oplus K_3^2$$

We use Piling-up lemma [MM94] to obtain the probability that this equation hold:

$$\Pr = \frac{1}{2} + 2^1 e_1 e_2$$

$$= \frac{1}{2} + 2 \times \frac{6}{16} \times \frac{6}{16}$$

$$= 0.78125$$

6.3 Extracting the partial subkey bits

Once a linear expression of the entire cipher had been obtained, we can deduce $K_1[1\ 3\ ] \oplus K_2[1\ 3]$ using Algorithm 1 [MM94], which is as follows:

Step 1 Let T be the number of plaintexts such that the left side of equation (9) is equal to zero.

Step 2 If T > N/ 2 (N denotes the number of plaintexts),
then guess $K_1[1\ 3\ ] \oplus K_2[1\ 3] = 0$ (when p > 1 / 2) or 1 (when p < 1/2)
else guess $K_1[1\ 3\ ] \oplus K_2[1\ 3] = 1$ (when p > 1 / 2) or 0 (when p < 1/2)

6.4 Conclusion

An automated linear cryptanalyser had been created and its source code is listed in **Appendix 8.4.** The program is then used to attack S-DES with the above parameter instance. The result of the successful attack is listed in **Appendix 8.7** . It is clear that the increase of plaintexts also increases the success rate our method [MM94] [HH00]. S-DES, as expected, yields to linear cryptanalysis. Since the entire process is automated, we can try several S-Box designs and use the automated cryptanalyser to obtain the best S-Box. This is done by selecting the S-Box that produces the linear approximation with the lowest probability.

7. Reference

[CA92] Adams, C. (1992), *On immunity against Biham and Shamir's differential cryptanalysis.* Information Processing Letters. 41: 77-80.

[AC97] Anne Canteaut (1997), *Differential cryptanalysis of Fesitel ciphers and differentially d-uniform mappings,*Domaine de Voluceau, France.

[MP96] A. Menezes,P. van Oorschot, S. Vanstone (1996), *Handbook of Applied Cryptography,* CRC Press.

[AG00] Anna Gorska et. al. (2000), *New Experimental Results in Differential-Linear Cryptanalysis of Reduced Variant of DES*, Polish Academy of Sciences

[BV95] Buttyan, L. and I Vajda (1995), *Searching for the best linear approximation of DES-like cryptosystems*. Electronics Letters. 31(11): 873-874.

[BS96] Bruce Schneier (1996), *Applied Crytography,Second Edition,* John Wiley and Sons,

[BE99] Bruce Schneier et. al. (1999), *The Twofish Encryption Algorithm,* John Wiley and Sons.

[BS00] Bruce Schneier (2000), *A Self-Study Course in Block-Cipher Cryptanalysis,* Counterpane Internet Security

[BS01] Bruce Schneier (2001), *Why Cryptography Is Harder Than It LooKS,* Counterpane Internet Security

[HM95] C. Harpes, G. Kramer, and J. L. Massey (1995), *A generalization of linear cryptanalysis and the applicability of Matsui's piling-up lemma,* Lecture Notes in Computer Science, 921.

[CY01]Chan Yeob Yeun Design (2000), *Analysis and applications of cryptographic techniques,* Department of Mathematics, Royal Holloway University of London.

[DH01]Daithi Hanluain.(2001) *Got Your Number* Ericsson ON: The New World of Communication, Issue 3. 2001

[DT91]Dawson, M. and S. Tavares (1991), *An Expanded Set of S-box Design Criteria Based on Information Theory and its Relation to Differential-Like AttacKS.* Advances in Cryptology -- EUROCRYPT '91. 353-367.

[DG94]Daemen, J., R. Govaerts and J. Vandewalle (1995*), Correlation Matrices. Fast Software Encryption.* Lecture Notes in Computer Science (LNCS) 1008. Springer-Verlag. 275-285

[EA93] Eli Biham and Adi Shamir (1993), *Differential Cryptanalysis of the Data Encryption Standard,* Springer-Verlag,Berlin,Germany.

[EA90] Eli Biham and Adi Shamir (1990), *Differential Cryptanalysis of DES-like Cryptosystems*. Advances in Cryptology -- CRYPTO '90. Springer-Verlag. 2-21.

[EA91]Eli Biham and Adi Shamir (1991). *Differential Cryptanalysis of Snefru, Khafre, REDOC-II, LOKI and Lucifer.* Advances in Cryptology -- CRYPTO '91. Springer-Verlag. 156-171.

[EA92]Eli Biham and Adi Shamir (1992), *Differential Cryptanalysis of the Full 16-round DES.* Advances in Cryptology -- CRYPTO '92. Springer-Verlag. 487-496.

[EB94]Eli Biham (1994), *On Matsui's Linear Cryptanalysis.* Technion, Israel Institute of Technology, Israel

[EF98] Electronic Fountier Foundation (1998), *Cracking DES:Secrets of Encryption Research, Wiretap Politics Chip Design,* OReilly and Associates,Inc.

[ER01] Eric Rescorla**,** *SSL and TLS: Design and Building Secure System,*Addison Wesley Professional, U.S.A.

[SC96] Edward Schaefer (1996),*A Simplified Data Encryption Standard Algorithm,*Cryptologia 96

[FC94] Florent Chabaud, Serge Vaudenay (1994), *LinKS between Differential and Linear Cryptanalysis,* Ecole Normale Superieure

[FM00]Fauza Mirzan (2000),*Block Ciphers and Cryptanalysis,* Department of Mathematics, Royal Holloway University of London

[HG95]Harpes, *C*., G. Kramer and J. Massey (1995), *A Generalization of Linear Cryptanalysis and the Applicability of Matsui's Piling-up Lemma,* Advances in Cryptology -- EUROCRYPT '95. 24-38.

[SH00]Henning Schulzrinne (2000),*Network Security:Secret Key Cryptograph,* Columbia University, New York

[HH00] Howard M. Heys (2000), *A Tutorial on Linear and Differential Cryptanalysis,*Memorial University of Newfoundland, Canada

[BJ98] Jason Bock (1998), *Visual Basic 6 WIN32 API Tutorial,* Wrox Publishing, 1998.

[DR00] Joan Daemen,Vincent Rijmen (2000), *AES Proposal:Rijndael,*http://csrc.nist.gov/encryption/aes/ Last Visited: 7th February 2001

[KR01]Joe Kilian and Philip Rogaway (2000), *How to Protect DES Against Exhaustive Key Search,* NEC Research Institute U.S.A.

[JT95]Joe Kilian and Philip Rogaway (1995),*A Fast Method for the Cryptanalysis of Substitution Ciphers,* Dragoer, Denmark

[AM00]Kazumaro Aoki, et. al. (2000),*Camellia: A 128-Bit Block Cipher Suitable for Multiple Platforms*, NTT Coporation and Mitsubishi Electric Corporation

[KM01] Keith T. Manley (2001) *Time For Java*, JAVAPro, Vol. 5, No. 8, Pg. 67, August,2001

[LH01] Liam Keliher, Henk Meijer, and Stafford Tavares (2001) *New Method for Upper Bounding the Maximum Average Linear Hull Probablility for SPNs*,Queen's University at Kingston, Canada

[KR94]Kaliski, B. and M. Robshaw (1994) Linear Cryptanalysis Using Multiple Approximations. *Advances in Cryptology -- CRYPTO '94.* 26-39.

[KH00] Liam Keliher, Henk Meijer, and Stafford Tavares (1999) *Modelling Linear Characteristics of Substitution-Permutation NetworKS*, pg. 78-91,SAC 1999.

[PL00]Lyndon G. Pierson (2000),*Comparing Cryptographic Modes of Operation using Flow Diagrams*,Sandia National Labarotaries, U.S.A.

[MC98] Matt Curtin,Justin Dolske. (1998).*A Brute Force Search of DES Keyspace* http://www.interhack.net/pubs/des-key-crack/ Last Visited: 27th November 2001

[MW01] Michael Welschenbach (2001), *Cryptography in C and C++*, Apress Publications.

[MS01] Microsoft Developer Network (2001), *Microsoft Developer Network Library*, Microsoft Corporation

[MK98] Matthew Kwan(1998) *Bitslice DES, S-Boxes Implementations*, http://www.darkside.com.au/bitslice/sboxes.c. Last Visited: 28 November 2001

[MM94] Mitsuru Matsui (1994), *Linear cryptanalysis method for DES cipher*, EUROCRYPT 1994, no.765, pg. 386-397,

[MF99] Matthew Fischer (1999), *How to implement the Data Encryption Standard*, University of Iowa

[MS94] M. Hellman and S. Langford (1994), *Differential-Linear Cryptanalysis*, CRYPTO '94, no. 839, pg. 26-39.

[MW95]Mark Allen Weiss (1995) , *Efficient C Programming, A practical approach*, Prentice Hall, U.S.A.

[MM941]Matsui, M. (1994), *The First Experimental Cryptanalysis of the Data Encryption Standard*. Advances in Cryptology -- CRYPTO '94. 1-11

[NS99] National Institute of Standards and Technology (1999), *Data Encryption Standard (DES)*, U.S. Department Of Commerce, U.S.A.

[NF01] Niels Ferguson et.al.(2001), *Improved Cryptanalysis of Rijndael*, Counterpane Internet Security,U.S.A.

[NK91] Nyberg, *K.* 1991. *Perfect nonlinear S-boxes.* Advances in Cryptology -- EUROCRYPT '91. 378-386.

[OS00] Ooi Kuan San. (2000), *Experiment with Pointers*,University of Sheffield Centre, Taylor's College, Malaysia.

[OS01] Ooi Kuan San (2001),*Bit Manipulation*,University of Sheffield Centre, Taylor's College, Malaysia.

[PJ98] Pascal A. Junod (1998), *Linear Cryptanalysis of DES*, Eidgenssische Tenhcische Hochschule, Zurich

[PG01] Paul Garrett (2001), *Making, Breaking Codes*, Prentice Hall, U.S.A.

[RP00]Richard Power, (2000).*Tangled Web*. Que Publications.U.S.A.

[RR86] Rainier A. Rueppel (1986), "Analysis and Design of Stream Ciphers," Spreinger Verlag.

[SS00] Simon Singh (2000), *The Science of Secrecy*, Fourth Estate Limited,

[LS00]Susan Landau (2000) Stan*ding the Test of Time: The Data Encryption Standard*, Sun Microsystems

[VM95] Serge Vaudenay and Shiho Moriai (1994), *Comparison of the Randomness Provided by Some AES Candidates*, EUROCRYPT 1994, Springer Verlag no. 950, pg. 386-397,

[SM00] Sean Murphy,M.J.B. Robshaw (2000),*Differential Cryptanalysis,Key-dependant,S-boxes, and Twofish*

[VS95]Vaudenay, S. (1995). *An Experiment on DES Statistical Cryptanalysis.* Ecole Normale Supérieure, France.

[LM91]Xuejia Lai, Massey J.L (1991). *Markov Ciphers and Differential Cryptanalysis.* Swiss Federal Institude of Technology, Royal Holloway University of London

[YT95]Youssef, A., S. Tavares. (1995). *Resistance of Balanced S-boxes to Linear and Differential Cryptanalysis.* Information Processing Letters. 56: 249-252

8.Appendix

8.1 S-DES Source Code

```
/**
 *S-DES Cipher
 *Developer 1: Dr. K.S. Ooi (ksooi@mailexcite.com)
 *Developer 2: Brain Chin Vito (v@chin.tc)
 *University of Sheffield Centre, Taylor's College
 *
 */
#include <stdio.h>
#include <stdlib.h>
#include <limits.h>
#include <string.h>

#define UINT unsigned int
#define BYTE unsigned char
#define BYTESIZE CHAR_BIT
#define BLOCKSIZE BYTESIZE
#define KEYSIZE      10
#define SUBKEYSIZE   8
#define SPLITKEYSIZE 5


/* ======================== general functions ================================= */


/**
 * E.g. of usage:
 *   //printBin(" 23 is: ", 23, BYTESIZE);
 *   //printBin(" ",cbin2UINT("10110010",BYTESIZE), BYTESIZE);
```

```
 *   //printBin(" 217 is: ", 217, BYTESIZE);
 *   //printf("String 11011001 is %u\n", cbin2UINT("11011001",BYTESIZE));
 *
 *   //printBin(" 729 is: ", 729, 10);
 *   //printf("String 1011011001 is %u\n", cbin2UINT("1011011001",KEYSIZE));
 *
 */
void printBin(const char *str,unsigned int bInteger,unsigned int nSize){
  char s[BYTESIZE*sizeof(UINT)];
  UINT i;
  UINT n=bInteger;

  for(i=0; i<nSize; i++)
    *(s + i)='0';  *(s+i)='\0';

  i= nSize - 1;

  while(n > 0){
   s[i--]=(n % 2)? '1': '0';
   n = n/2;
  }

  printf("%s%s [%+3u in decimal]\n",str,s,bInteger);

}



/**
 * E.g. of usage:
 *   //printf("String 11011001 is %u\n", cbin2UINT("11011001",BYTESIZE));
 *   //printf("String 1011011001 is %u\n", cbin2UINT("1011011001",KEYSIZE));
 */
UINT cbin2UINT(char *s, unsigned int nSize){
  int nLen = strlen(s);
  UINT uResult = 0;

  while (--nLen >= 0)
    if(s[nLen] == '1')
       uResult = 1 << (nSize - nLen - 1)  | uResult;

  return uResult;
}



/* ======================== Key Scheduling ================================== */

/**
 *
 * Only valid for max size 10
 * Shift size is two, max
 *   printBin(" ",cbin2UINT("10110010",BYTESIZE), BYTESIZE);
 *   printBin(" ",leftShift(bin2UINT("10110010",BYTESIZE),2,BYTESIZE), BYTESIZE);
 *   printBin(" ",leftShift(bin2UINT("10110010",BYTESIZE),1,BYTESIZE), BYTESIZE);
 *
 */
UINT leftShift(UINT nKey, UINT nShift, UINT nSize){
  UINT n = nKey >> (nSize - nShift), i, nMask=0;
  nKey <<= nShift;

  for(i=0; i< nSize; i++)
     nMask |= 1 << i;

  return (nKey | n) & nMask;
}



/**
 * Permutation box p10
```

```
 *   printBin(" ",box_p10(cbin2UINT("1011011001",KEYSIZE)),KEYSIZE);
 */
UINT p10[] ={9,7,3,8,0,2,6,5,1,4};

UINT box_p10(UINT key10){
  UINT uResult=0, i=0;

  for(; i< KEYSIZE; i++)
    if (1 << (KEYSIZE - p10[i] - 1) & key10)
      uResult |= 1 << (KEYSIZE - i - 1);

  return uResult;
}

/**
 * Split Key
 *   printBin("",splitKey(cbin2UINT("1011011001",KEYSIZE), keyArray),5);
 * where keyArray is an array of 2 '5 bit' keys
 */
void splitKey(UINT p10Key10, UINT uResult[]){
  /**
   *  31 == 0000011111
   * 992 == 1111100000
   */
  UINT H_SPLIT5BIT_MASK = 31, L_SPLIT5BIT_MASK = 992;

  uResult[0] =  (p10Key10 & L_SPLIT5BIT_MASK) >> SPLITKEYSIZE;
  uResult[1] =  p10Key10 & H_SPLIT5BIT_MASK;

}

/**
 * Permutation box p8
 */
UINT p8[] = {3,1,7,5,0,6,4,2};

UINT box_p8(UINT key5[]){
  UINT uResult=0, uTemp, i=0;
  /*
   * 255 = 11111111
   */
  UINT uMask = 255;

  uTemp =  key5[0] << SPLITKEYSIZE | key5[1];
  uTemp &= uMask;

  for(; i< SUBKEYSIZE; i++)
    if (1 << (KEYSIZE - p8[i] - 1) & uTemp)
      uResult |= 1 << (KEYSIZE - i - 3);

  return uResult;
}


/**
 * Key Schedule
 */
void keySchedule(UINT key10,UINT key8[]){
  UINT key5[2]={0,0}, keyTemp, i;

  keyTemp = box_p10(key10);

  splitKey(keyTemp, key5);

  for(i=0; i<2 ; i++){
    key5[0] = leftShift(key5[0], i+1, SPLITKEYSIZE);
    key5[1] = leftShift(key5[1], i+1, SPLITKEYSIZE);

    key8[i]=box_p8(key5);
  }
}
```

```
/* ======================== IP and IP_1 ================================== */

UINT IP[]  = {7,6,4,0,2,5,1,3};

UINT IP_1[]= {3,6,4,7,2,5,1,0};


BYTE per(UINT P[], BYTE input){
  BYTE bRes = 00;
  int i = 8;

  while(--i >= 0)
    if( 01 << (BLOCKSIZE - P[BLOCKSIZE - i - 1] - 1) & input )
      bRes |= (01 << i);

  return bRes;
}

/* ======================== Round ============================== */
void split824(BYTE bInput8, BYTE bLR[]){
  BYTE L_mask = 240, H_mask = 15;

  /** left  */
  bLR[0] = (bInput8 & L_mask) >> 4;

  /** right */
  bLR[1] = bInput8 & H_mask;

}

BYTE E[] = {0,1,0,0,2,3,3,2};
BYTE P4[] = {1,0,3,2};
BYTE S0[]={1,0,2,3,3,1,0,2,2,0,3,1,1,3,2,0};
BYTE S1[]={0,3,1,2,3,2,0,1,1,0,3,2,2,1,3,0};

/**
 * f-function
 */
BYTE f(BYTE bRight, BYTE key){
  BYTE bRes = 00, bTemp;
  BYTE sLR4[]={0,0}, r, c;
  int i = SUBKEYSIZE;

  while(--i >= 0)
    if( 01 << (4 - E[SUBKEYSIZE - i - 1] - 1) & bRight )
      bRes |= (01 << i);

  bRes ^= key;

  split824(bRes,sLR4);

  c = (sLR4[0] & 6) >> 1;
  r = (sLR4[0] & 8) >> 2 | (sLR4[0] & 01);
  sLR4[0] = S0[4*r + c] << 2;

  c = (sLR4[1] & 6) >> 1;
  r = (sLR4[1] & 8) >> 2 | (sLR4[1] & 01);
  sLR4[1] = S1[4*r + c];

  bTemp = sLR4[0] | sLR4[1];

  bRes = 00;

  // permute using P4
  i=4;
  while(--i >= 0)
    if( 01 << (4 - P4[4 - i - 1] - 1) & bTemp )
      bRes |= (01 << i);
```

```
  return bRes;
}

int main(void){

  UINT key8[2]={0,0};
  UINT key10  = cbin2UINT("0000100000",KEYSIZE);
  BYTE input8 = (BYTE) cbin2UINT("00100000",BLOCKSIZE), exInput8, i;
  /** left and Right */
  BYTE LR[] ={00,00};
  char useKey;
  char userKey[]="0000000000",userText[]="0000000000";

  printf("================================S-DES================================\n");
  putchar('\n');
  printf(" Do you want to use the predetermined key (y/n) ?");
  scanf("%c",&useKey);
  if(useKey!='y'){
        printf(" Please specify key to use (10 BITS) ?");
    scanf("%10s",&userKey);
        key10  = cbin2UINT(userKey,KEYSIZE);
  }

  printf(" Please specify plaintext (10 BITS) ?");
  scanf("%8s",&userText);
  /** display the input */
  input8 = (BYTE) cbin2UINT(userText,BLOCKSIZE);
  printBin(" Plaintext  = ", input8, BLOCKSIZE);
  printBin(" Key        = ", key10, KEYSIZE);
  keySchedule(key10,key8);
  putchar('\n');
  printf(" ============================ Encrypt ============================ \n");


  input8 = per(IP,input8);

// ===> Start of the round
  exInput8 = input8;

 for(i=0; i< 2; i++){
  /** =====> begin round */

  split824(exInput8,LR);

  input8 = (f(LR[1],(BYTE)key8[i])^LR[0]) << 4; //WAPIANG!! Sai mu sai ah..
  input8 |= LR[1];
  exInput8 = ((input8 & 240) >> 4) | ( (input8 & 15) << 4 );
  /** =====> end of round */
 }
  input8 = per(IP_1,input8);

  printBin(" Ciphertext = ", input8, BLOCKSIZE);
  putchar('\n');
  printf(" ========================== Decrypt ========================== \n");
  input8 = per(IP,input8);

  exInput8 = input8;

  for(i=0; i< 2; i++){
  /** =====> begin round */
  split824(exInput8,LR);

  input8 = (f(LR[1],(BYTE)key8[(i+1)%2])^LR[0]) << 4;
  input8 |= LR[1];
  exInput8 = ((input8 & 240) >> 4) | ( (input8 & 15) << 4 );
  /** =====> end of round */
 }
  input8 = per(IP_1,input8);
  printBin(" Decrypted Ciphertext   = ", input8, BLOCKSIZE);
  putchar('\n');
```

```
  return EXIT_SUCCESS;
}
```

8.2 Automated Brute Force Attack of S-DES Source Code

```
/**
 *Automated Brute Force Attack
 *Designed for: S-DES
 *Developer 1: Dr. K.S. Ooi (ksooi@mailexcite.com)
 *Developer 2: Brain Chin Vito (v@chin.tc)
 *University of Sheffield Centre, Taylor's College
 */
#include <stdio.h>
#include <stdlib.h>
#include <limits.h>
#include <string.h>

#define UINT unsigned int
#define BYTE unsigned char
#define BYTESIZE CHAR_BIT
#define BLOCKSIZE BYTESIZE
#define KEYSIZE      10
#define SUBKEYSIZE   8
#define SPLITKEYSIZE 5

BYTE ctext = 0;
BYTE E[] = {0,1,0,0,2,3,3,2};
BYTE P4[] = {1,0,3,2};
BYTE S0[]={1,0,2,3,3,1,0,2,2,0,3,1,1,3,2,0};
BYTE S1[]={0,3,1,2,3,2,0,1,1,0,3,2,2,1,3,0};

/* ======================== general functions ================================ */

/**
 * E.g. of usage:
 *    printBin(" 23 is: ", 23, BYTESIZE);
 *    printBin(" ",cbin2UINT("10110010",BYTESIZE), BYTESIZE);
 *
 *   printBin(" 217 is: ", 217, BYTESIZE);
 *   printf("String 11011001 is %u\n", cbin2UINT("11011001",BYTESIZE));
 *
 *   printBin(" 729 is: ", 729, 10);
 *   printf("String 1011011001 is %u\n", cbin2UINT("1011011001",KEYSIZE));
 *
 */

void printBin(const char *str,unsigned int bInteger,unsigned int nSize){
  char s[BYTESIZE*sizeof(UINT)];
  UINT i;
  UINT n=bInteger;

  for(i=0; i<nSize; i++)
    *(s + i)='0';  *(s+i)='\0';

  i= nSize - 1;

  while(n > 0){
   s[i--]=(n % 2)? '1': '0';
   n = n/2;
  }

  printf("%s%s [%+3u in decimal]\n",str,s,bInteger);

}

/**
 * E.g. of usage:
 *   //printf("String 11011001 is %u\n", cbin2UINT("11011001",BYTESIZE));
 *   //printf("String 1011011001 is %u\n", cbin2UINT("1011011001",KEYSIZE));
 */
```

```
UINT cbin2UINT(char *s, unsigned int nSize){
  int nLen = strlen(s);
  UINT uResult = 0;

  while (--nLen >= 0)
    if(s[nLen] == '1')
       uResult = 1 << (nSize - nLen - 1)  | uResult;

  return uResult;
}



/* ======================== Key Scheduling ================================= */

/**
 *
 * Only valid for max size 10
 * Shift size is two, max
 *   printBin(" ",cbin2UINT("10110010",BYTESIZE), BYTESIZE);
 *   printBin(" ",leftShift(bin2UINT("10110010",BYTESIZE),2,BYTESIZE), BYTESIZE);
 *   printBin(" ",leftShift(bin2UINT("10110010",BYTESIZE),1,BYTESIZE), BYTESIZE);
 *
 */
UINT leftShift(UINT nKey, UINT nShift, UINT nSize){
  UINT n = nKey >> (nSize - nShift), i, nMask=0;
  nKey <<= nShift;

  for(i=0; i< nSize; i++)
     nMask |= 1 << i;

  return (nKey | n) & nMask;
}



/**
 * Permutation box p10
 *   printBin(" ",box_p10(cbin2UINT("1011011001",KEYSIZE)),KEYSIZE);
 */
UINT p10[] ={9,7,3,8,0,2,6,5,1,4};

UINT box_p10(UINT key10){
  UINT uResult=0, i=0;

  for(; i< KEYSIZE; i++)
    if (1 << (KEYSIZE - p10[i] - 1) & key10)
      uResult |= 1 << (KEYSIZE - i - 1);

  return uResult;
}

/**
 * Split Key
 *   printBin("",splitKey(cbin2UINT("1011011001",KEYSIZE), keyArray),5);
 * where keyArray is an array of 2 '5 bit' keys
 */
void splitKey(UINT p10Key10, UINT uResult[]){
  /**
   *  31 == 0000011111
   * 992 == 1111100000
   */
  UINT H_SPLIT5BIT_MASK = 31, L_SPLIT5BIT_MASK = 992;

  uResult[0] =  (p10Key10 & L_SPLIT5BIT_MASK) >> SPLITKEYSIZE;
  uResult[1] =  p10Key10 & H_SPLIT5BIT_MASK;

}

/**
 * Permutation box p8
```

```
 */
UINT p8[] = {3,1,7,5,0,6,4,2};

UINT box_p8(UINT key5[]){
  UINT uResult=0, uTemp, i=0;
  /*
   * 255 = 11111111
   */
  UINT uMask = 255;

  uTemp =  key5[0] << SPLITKEYSIZE | key5[1];
  uTemp &= uMask;

  for(; i< SUBKEYSIZE; i++)
    if (1 << (KEYSIZE - p8[i] - 1) & uTemp)
      uResult |= 1 << (KEYSIZE - i - 3);

  return uResult;
}

/**
 * Key Schedule
 */
void keySchedule(UINT key10,UINT key8[]){
  UINT key5[2]={0,0}, keyTemp, i;

  keyTemp = box_p10(key10);

  splitKey(keyTemp, key5);

  for(i=0; i<2 ; i++){
    key5[0] = leftShift(key5[0], i+1, SPLITKEYSIZE);
    key5[1] = leftShift(key5[1], i+1, SPLITKEYSIZE);

    key8[i]=box_p8(key5);
  }
}


/* ======================== IP and IP_1 ================================== */

UINT IP[]  = {7,6,4,0,2,5,1,3};

UINT IP_1[]= {3,6,4,7,2,5,1,0};


BYTE per(UINT P[], BYTE input){
  BYTE bRes = 00;
  int i = 8;

  while(--i >= 0)
    if( 01 << (BLOCKSIZE - P[BLOCKSIZE - i - 1] - 1) & input )
      bRes |= (01 << i);

  return bRes;
}

/* ======================== Round ============================== */
void split824(BYTE bInput8, BYTE bLR[]){
  BYTE L_mask = 240, H_mask = 15;

  /** left  */
  bLR[0] = (bInput8 & L_mask) >> 4;

  /** right */
  bLR[1] = bInput8 & H_mask;

}

/**
 * f-function
```

```
 */
BYTE f(BYTE bRight, BYTE key){
  BYTE bRes = 00, bTemp;
  BYTE sLR4[]={0,0}, r, c;
  int i = SUBKEYSIZE;

  while(--i >= 0)
    if( 01 << (4 - E[SUBKEYSIZE - i - 1] - 1) & bRight )
      bRes |= (01 << i);

  bRes ^= key;

  split824(bRes,sLR4);

  c = (sLR4[0] & 6) >> 1;
  r = (sLR4[0] & 8) >> 2 | (sLR4[0] & 01);
  sLR4[0] = S0[4*r + c] << 2;

  c = (sLR4[1] & 6) >> 1;
  r = (sLR4[1] & 8) >> 2 | (sLR4[1] & 01);
  sLR4[1] = S1[4*r + c];


  bTemp = sLR4[0] | sLR4[1];

  bRes = 00;

  // permute using P4
  i=4;
  while(--i >= 0)
    if( 01 << (4 - P4[4 - i - 1] - 1) & bTemp )
      bRes |= (01 << i);

   return bRes;
}

/**
 *Takes in a key and encrypt a fixed plaintext using that key to get the corresponding
 *ciphertext. The ciphertext is then compared to the ciphertext ctext(global variable),
 *and if it is the same, this function returns 1, otherwise this function returns a zero.
 */
int crypt(UINT currentkey,int flag){

  UINT key8[2]={0,0};
  UINT key10  = currentkey;
  BYTE input8 = (BYTE) cbin2UINT("00100000",BLOCKSIZE), exInput8, i;


  /** left and Right */
  BYTE LR[] ={00,00};

  /** display the input */

  keySchedule(key10,key8);

  input8 = per(IP,input8);

// ===> Start of the round
  exInput8 = input8;

 for(i=0; i< 2; i++){
  /** =====> begin round */

  split824(exInput8,LR);

  input8 = (f(LR[1],(BYTE)key8[i])^LR[0]) << 4;
  input8 |= LR[1];
  exInput8 = ((input8 & 240) >> 4) | ( (input8 & 15) << 4 );
  /** =====> end of round */
 }
  input8 = per(IP_1,input8);
```

```
  if(flag==0){
        ctext=input8;
        return  -1;
  }
  else{
  if(input8==ctext)
         return 1;
  else return 0;
  }
}

int main(void){

  UINT i=0;
  UINT targetKey=0;
  UINT actualKey  = cbin2UINT("0000100000",KEYSIZE);
  char cont,useKey;
  char userKey[]="0000000000";

  printf("============Automated Brute Force Attack============");
  putchar('\n');
  printf(" Do you want to use the predetermined key (y/n) ?");
  scanf("%c",&useKey);
  if(useKey!='y'){
        printf(" Please specify key to use (10 BITS) ?");
    scanf("%s",&userKey);
        actualKey = cbin2UINT(userKey,KEYSIZE);
  }

  crypt(actualKey,0);

  while(crypt(i,1)==0)
  {
     i++;
         targetKey++;
  }

  printBin(" Cracked Key = ",targetKey,KEYSIZE);
  printBin(" Actual  Key = ",actualKey,KEYSIZE);
  printf(" Note: Key Suggested May Differ From Actual Key Because Suggested Key Can Also
\n");
  printf("       Be Used To Encrypt And Decrypt With Equivalent Results \n");


  return EXIT_SUCCESS;
}
```

8.3 Automated Differential Cryptanalyser Source Code

```
/**
 *========== Automated Differential Cryptanalyser ==========
 *Designed for: S-DES
 *Developer 1: Dr. K.S. Ooi (ksooi@mailexcite.com)
 *Developer 2: Brain Chin Vito (v@chin.tc)
 *University of Sheffield Centre, Taylor's College
 */

#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <limits.h>
#include <string.h>

#define COLUMN unsigned int
#define ROW unsigned int
#define ELEMENT unsigned int
#define INDEX unsigned int
#define BYTE unsigned char
```

```
#define UINT unsigned int
#define BYTESIZE CHAR_BIT
#define BLOCKSIZE BYTESIZE
#define KEYSIZE      10
#define SUBKEYSIZE   8
#define SPLITKEYSIZE 5

UINT cbin2UINT(char*, UINT);

/*================================global variables==========================*/
BYTE R1X=0;
BYTE R1Y=0;
BYTE C=0;
BYTE C2=0;
int r=0;
BYTE S0[]={1,0,2,3,3,1,0,2,2,0,3,1,1,3,2,0};
BYTE S1[]={0,1,2,3,2,0,1,3,3,0,1,0,2,1,0,3};
BYTE E[] = {0,2,1,3,0,1,2,3};
BYTE P4[] = {1,0,3,2};
ELEMENT DPS0[16][16];
ELEMENT DTS0[16][4];
ELEMENT DP2S0[16][16];
ELEMENT DT2S0[16][4];
BYTE R1XCHAR=0;
BYTE R1YCHAR=0;
int dex=0,dey=0,dex2=0,dey2=0;
double prob=0.0,prob2=0.0;
UINT key10  = 255;


/* ======================== general functions ================================= */

/**
 * E.g. of usage:
 *   //printBin(" 23 is: ", 23, BYTESIZE);
 *   //printBin(" ",cbin2UINT("10110010",BYTESIZE), BYTESIZE);
 *   //printBin(" 217 is: ", 217, BYTESIZE);
 *   //printf("String 11011001 is %u\n", cbin2UINT("11011001",BYTESIZE));
 *
 *   //printBin(" 729 is: ", 729, 10);
 *   //printf("String 1011011001 is %u\n", cbin2UINT("1011011001",KEYSIZE));
 *
 */

void printBin(const char *str,unsigned int bInteger,unsigned int nSize){
  char s[BYTESIZE*sizeof(UINT)];
  UINT i;
  UINT n=bInteger;

  for(i=0; i<nSize; i++)
    *(s + i)='0';  *(s+i)='\0';

  i= nSize - 1;

  while(n > 0){
   s[i--]=(n % 2)? '1': '0';
   n = n/2;
  }

  printf("%s%s [%+3u in decimal]\n",str,s,bInteger);

}

/**
 * E.g. of usage:
 *   //printf("String 11011001 is %u\n", cbin2UINT("11011001",BYTESIZE));
 *   //printf("String 1011011001 is %u\n", cbin2UINT("1011011001",KEYSIZE));
 */

UINT cbin2UINT(char *s, unsigned int nSize){
  int nLen = strlen(s);
```

```
  UINT uResult = 0;

  while (--nLen >= 0)
    if(s[nLen] == '1')
       uResult = 1 << (nSize - nLen - 1)  | uResult;

  return uResult;
}



/* ======================== Key Scheduling ================================== */

/**
 *
 * Only valid for max size 10
 * Shift size is two, max
 *   //printBin(" ",cbin2UINT("10110010",BYTESIZE), BYTESIZE);
 *   //printBin(" ",leftShift(bin2UINT("10110010",BYTESIZE),2,BYTESIZE), BYTESIZE);
 *   //printBin(" ",leftShift(bin2UINT("10110010",BYTESIZE),1,BYTESIZE), BYTESIZE);
 *
 */
UINT leftShift(UINT nKey, UINT nShift, UINT nSize){
  UINT n = nKey >> (nSize - nShift), i, nMask=0;
  nKey <<= nShift;

  for(i=0; i< nSize; i++)
     nMask |= 1 << i;

  return (nKey | n) & nMask;
}



/**
 * Permutation box p10
 *   //printBin(" ",box_p10(cbin2UINT("1011011001",KEYSIZE)),KEYSIZE);
 */
UINT p10[] ={9,7,3,8,0,2,6,5,1,4};

UINT box_p10(UINT key10){
  UINT uResult=0, i=0;

  for(; i< KEYSIZE; i++)
    if (1 << (KEYSIZE - p10[i] - 1) & key10)
      uResult |= 1 << (KEYSIZE - i - 1);

  return uResult;
}

/**
 * Split Key
 *   printBin("",splitKey(cbin2UINT("1011011001",KEYSIZE), keyArray),5);
 * where keyArray is an array of 2 '5 bit' keys
 */
void splitKey(UINT p10Key10, UINT uResult[]){
  /**
   *  31 == 0000011111
   * 992 == 1111100000
   */
  UINT H_SPLIT5BIT_MASK = 31, L_SPLIT5BIT_MASK = 992;

  uResult[0] =  (p10Key10 & L_SPLIT5BIT_MASK) >> SPLITKEYSIZE;
  uResult[1] =  p10Key10 & H_SPLIT5BIT_MASK;

}

/**
 * Permutation box p8
 */
UINT p8[] = {3,1,7,5,0,6,4,2};
```

```
UINT box_p8(UINT key5[]){
  UINT uResult=0, uTemp, i=0;
  /*
   * 255 = 11111111
   */
  UINT uMask = 255;

  uTemp =  key5[0] << SPLITKEYSIZE | key5[1];
  uTemp &= uMask;

  for(; i< SUBKEYSIZE; i++)
    if (1 << (KEYSIZE - p8[i] - 1) & uTemp)
      uResult |= 1 << (KEYSIZE - i - 3);

  return uResult;
}

/**
 * Key Schedule
 */
void keySchedule(UINT key10,UINT key8[]){
  UINT key5[2]={0,0}, keyTemp, i;

  keyTemp = box_p10(key10);

  splitKey(keyTemp, key5);

  for(i=0; i<2 ; i++){
    key5[0] = leftShift(key5[0], i+1, SPLITKEYSIZE);
    key5[1] = leftShift(key5[1], i+1, SPLITKEYSIZE);
    key8[i]=box_p8(key5);
  }
}


/* ======================== IP and IP_1 ================================== */

UINT IP[]  = {7,6,4,0,2,5,1,3};

UINT IP_1[]= {3,6,4,7,2,5,1,0};


BYTE per(UINT P[], BYTE input){
  BYTE bRes = 00;
  int i = 8;

  while(--i >= 0)
    if( 01 << (BLOCKSIZE - P[BLOCKSIZE - i - 1] - 1) & input )
      bRes |= (01 << i);

  return bRes;
}

/* ======================== Round ================================ */
void split824(BYTE bInput8, BYTE bLR[]){
  BYTE L_mask = 240, H_mask = 15;

  /** left  */
  bLR[0] = (bInput8 & L_mask) >> 4;

  /** right */
  bLR[1] = bInput8 & H_mask;

}


/**
 * f-function
 */
BYTE f(BYTE bRight, BYTE key){
```

```
  BYTE bRes = 00, bTemp;
  BYTE sLR4[]={0,0}, r, c;
  int i = SUBKEYSIZE;

  while(--i >= 0)
    if( 01 << (4 - E[SUBKEYSIZE - i - 1] - 1) & bRight )
      bRes |= (01 << i);

  bRes ^= key;

  split824(bRes,sLR4);

  c = (sLR4[0] & 6) >> 1;
  r = (sLR4[0] & 8) >> 2 | (sLR4[0] & 01);
  sLR4[0] = S0[4*r + c] << 2;

  c = (sLR4[1] & 6) >> 1;
  r = (sLR4[1] & 8) >> 2 | (sLR4[1] & 01);
  sLR4[1] = S1[4*r + c];


  bTemp = sLR4[0] | sLR4[1];


  bRes = 00;

  // permute using P4
  i=4;
  while(--i >= 0)
    if( 01 << (4 - P4[4 - i - 1] - 1) & bTemp )
      bRes |= (01 << i);

  return bRes;
}


/**
 * Encrypt the given inputbits and returns the encryped input.
 * Besides it stores the following data in global variables:
 * RY1 : Round 1 output
 * C: Ciphertext for first plaintext to be encrypted
 * C2: Ciphertext for second plaintext to be encrypted
 */
int crypt(BYTE inputbits){

  UINT key8[2]={0,0};
  BYTE input8 = inputbits, exInput8, i;

  /** left and Right */
  BYTE LR[] ={00,00};

  keySchedule(key10,key8);

// ===> Start of the round
  exInput8 = input8;

  R1X=exInput8;
 for(i=0; i< 2; i++){
  /** =====> begin round */

  split824(exInput8,LR);

  input8 = (f(LR[1],(BYTE)key8[i])^LR[0]) << 4;
  input8 |= LR[1];
  exInput8 = ((input8 & 240) >> 4) | ( (input8 & 15) << 4 );
  /** =====> end of round */
  if(i==0){
         R1Y=exInput8;
  }
 }
```

```
  if(r==0){
        C=input8;
        r++;
  }else{
        C2=input8;
        r--;
  }

  return exInput8;
}


/**
 * The last round of the cipher. Takes in an inputbit an a key and encrypt it,
 * with that key for one round.
 */
BYTE lastRound(BYTE inputbits,UINT key){

          /** left and Right */
  BYTE LR[] ={00,00};

  BYTE input8 = inputbits;
  BYTE exInput8;
  exInput8 = input8;
  split824(exInput8,LR);
  input8 = (f(LR[1],(BYTE)key)^LR[0]) << 4;
  input8 |= LR[1];

  return input8;
}

/*==========================cryptanalytical functions======================*/

void init(){
        INDEX i,j;

        for(i=0;i<16;i++)
                for(j=0;j<16;j++){
                        DPS0[i][j]=0;
                        DP2S0[i][j]=0;
                }
        for(i=0;i<16;i++)
                for(j=0;j<4;j++){
                        DTS0[i][j]=0;
                        DT2S0[i][j]=0;
                }
}

/**
 * Returns the output value of the stated S-Box given the input to the S-Box.
 */
ELEMENT SIO(COLUMN col,BYTE S[16]){

  ELEMENT value;
  BYTE r,c;

  c = (col & 6) >> 1;
  r = (col & 8) >> 2 | (col & 01);
  value = S[4*r + c];

  return value;
}

/**
 *Construct a difference pair table for the two S-Boxes of S-DES
 *
 */
void difPair(){

        COLUMN x=0;
        COLUMN dx=0;
```

```
	for(x=0;x<16;x++)
		for(dx=0;dx<16;dx++){
			DPS0[x][dx]=((SIO(x,S0))^(SIO(x^dx,S0)));
			DP2S0[x][dx]=((SIO(x,S1))^(SIO(x^dx,S1)));
		}
}

/**
 *Counts the number of dy in the column dx in DS0
 *dy: The output difference
 *dx: The input difference
 *DS0: The difference pair table
 */
ELEMENT count(COLUMN dx,ROW dy,ELEMENT DS0[16][16]){

	INDEX i;
	int cnt=0;

	for(i=0;i<16;i++)
		if((DS0[i][dx])==dy)
			cnt++;

	return cnt;
}

/**
 * The difference distribution table
 */
void difTab(){

	COLUMN dx=0;
	ROW dy=0;

	for(dx=0;dx<16;dx++){
		for(dy=0;dy<4;dy++){
			DTS0[dx][dy]=count(dx,dy,DPS0);
			DT2S0[dx][dy]=count(dx,dy,DP2S0);
		}
	}
}

/**
 * Print out the difference pair table on the screen.
 */
void printDPT(ELEMENT DS0[16][16]){

	ROW x=0;
	COLUMN dx=0;
	printf("------------------------------------------------- \n");
	printf("              ");
	putchar((char)383);
	printf("Y given ");
	putchar((char)383);
	printf("X                \n");
	printf("------------------------------------------------- \n");
	printf("x   0  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 \n");
	printf("------------------------------------------------- \n");
	for(x=0;x<16;x++){
	printf("%.2d",x);
		for(dx=0;dx<16;dx++){
			printf("  %d",DS0[x][dx]);
		}
		putchar('\n');
	}
}

/**
 * Print out the difference distribution table on the screen.
 */
void printDT(ELEMENT DS0[16][4]){
```

```
        ROW x=0;
        COLUMN dx=0;
    printf("-------------------\n");
        printf("              ");
        putchar((char)383);
        printf("Y \n");
        putchar((char)383);
        printf("X     0     1     2     3   \n");
        printf("--------------------\n");

        for(x=0;x<16;x++){
                printf("%.2d",x);
                for(dx=0;dx<4;dx++){
                        printf("    %.2d",DS0[x][dx]);
                }
                putchar('\n');
        }
}

/**
 * Find the largest value in the given difference distribution table
 * and store it's column and row in a global variable dex and dey.
 */
void findDC(int ident,ELEMENT DTS[16][4]){

        ELEMENT curV=0,curL=0;
    INDEX i,j;

        if(ident==0){
        for(i=0;i<16;i++){
                for(j=0;j<4;j++){
                        curV=DTS[i][j];
                        if((curV>curL)&(curV!=16)){
                                curL=curV;
                                dex=i;
                                dey=j;
                        }
                }
        }prob=((double)curL)/16;}else{
        for(i=0;i<16;i++){
                for(j=0;j<4;j++){
                        curV=DTS[i][j];
                        if((curV>curL)&(curV!=16)){
                                curL=curV;
                                dex2=i;
                                dey2=j;
                        }
                }
        }       prob2=((double)curL)/16;
        }


}

/**
 *Print the expected subkey on the screen
 */
void printES(){
  UINT key8[2]={0,0};
  keySchedule(key10,key8);
  printBin("Expected subkey = ",(key8[1]),SUBKEYSIZE);
}

/**
 * Given an array of key counts, with the index representing the key,
 * this function prints the key with the largest count as the Guessed Subkey
 * on the Screen.
 */
void printGS(int K[256]){
```

```
  ELEMENT curV=0,curL=0;
  int i,k;

        for(i=0;i<255;i++){
                curV=K[i];
                if(curV>curL){
                                curL=curV;
                                k=i;
                }
        }

  printBin("Guessed Subkey = ",k,SUBKEYSIZE);
}

/**
 *Extend the Differential Characteristic of the S-Boxes to the round with
 *consideration to the expansion and permutation in f.
 */
void extendDC(){

        if((dex&8)==8){
                        R1XCHAR=(R1XCHAR|8);
                        R1YCHAR=(R1YCHAR|128);
        }
        if((dex&4)==4){
                        R1XCHAR=(R1XCHAR|2);
                        R1YCHAR=(R1YCHAR|32);
        }
        if((dex&2)==2){
                        R1XCHAR=(R1XCHAR|4);
                        R1YCHAR=(R1YCHAR|64);
        }
        if((dex&1)==1){
                        R1XCHAR=(R1XCHAR|1);
                        R1YCHAR=(R1YCHAR|16);
        }
        if((dex2&8)==8){
                        R1XCHAR=(R1XCHAR|8);
                        R1YCHAR=(R1YCHAR|128);
        }
        if((dex2&4)==4){
                        R1XCHAR=(R1XCHAR|4);
                        R1YCHAR=(R1YCHAR|64);
        }
        if((dex2&2)==2){
                        R1XCHAR=(R1XCHAR|2);
                        R1YCHAR=(R1YCHAR|32);
        }
        if((dex2&1)==1){
                        R1XCHAR=(R1XCHAR|1);
                        R1YCHAR=(R1YCHAR|16);
        }

                if((dey&2)==2)
                        R1YCHAR=(R1YCHAR|4);
                if((dey&2)==1)
                        R1YCHAR=(R1YCHAR|8);
                if((dey2&2)==2)
                        R1YCHAR=(R1YCHAR|1);
                if((dey2&1)==1)
                        R1YCHAR=(R1YCHAR|2);

}

int main(void){

        BYTE input=0;
        BYTE dx=16;
    BYTE curR1Y=0;
        BYTE i;
        BYTE k;
```

```
        BYTE candidate=0;
        BYTE lstest=0;
        int count=0;
        int PK2[256];
        char cont,useKey;
    char userKey[]="0000000000";

  printf("========== Automated Differential Cryptanalyser ==========\n");
  putchar('\n');
  printf(" Do you want to use the predetermined key (y/n) ?");
  scanf("%c",&useKey);
  if(useKey!='y'){
        printf(" Please specify key to use (10 BITS) ?");
    scanf("%s",&userKey);
        key10  = cbin2UINT(userKey,KEYSIZE);
  }
  scanf("%c",&cont);

  init();
  difPair();
  difTab();
  printf("Difference Pairs for S0\n");
  printDPT(DPS0);
  printf("Press enter to continue\n");
  scanf("%c",&cont);
  printf("Difference Distribution Table for S0\n");
  printDT(DTS0);
  printf("Press enter to continue\n");
  scanf("%c",&cont);
  printf("Difference Pairs for S1\n");
  printDPT(DP2S0);
  printf("Press enter to continue\n");
  scanf("%c",&cont);
  printf("Difference Distribution Table for S1\n");
  printDT(DT2S0);
  findDC(0,DTS0);
  printf("Press enter to continue\n");
  scanf("%c",&cont);
  printf("For S-Box 0\n ----------\n");
  printf("Best Difference Pair: dex = %i , dey = %i \n",dex,dey);
  printf("Probability = %f \n",prob);
  findDC(1,DT2S0);
  putchar('\n');
  printf("For S-Box 1\n ----------\n");
  printf("Best Difference Pair: dex2 = %i , dey2 = %i \n",dex2,dey2);
  printf("Probability = %f \n",prob2);
  putchar('\n');
  printf("Probability of Characteristic\n ----------------------\n");
  printf("Best Difference Pair: dex2 = %i , dey2 = %i \n",dex2,dey2);
  printf("Probability = %f \n",prob*prob2);
  putchar('\n');

  extendDC();

        for(i=0;i<255;i++)
                PK2[i]=0;

  printf("Press enter to continue\n");
  scanf("%c",&cont);

for(input=0;input<255;input++){

        crypt(input);

        curR1Y=R1Y;
        crypt((BYTE)(input^R1XCHAR));

        if((R1Y^curR1Y)==(R1YCHAR)){
                count++;
                for(k=0;k<255;k++){
                        if((lastRound(curR1Y,k)==C)&(lastRound(R1Y,k)==C2))
```

```
                                PK2[k]++;
                        }
                }

                }

                for(i=0;i<255;i++)
                        printf("Key %d = %d \n",i,PK2[i]);

                printf("COUNT=%d\n",count);
                printES();
                printGS(PK2);

                putchar('(');
                putchar((char)383);
                printf("X)");
                printBin("Round 1 Input Characteristic = ",R1XCHAR,BLOCKSIZE);
                putchar('(');
                putchar((char)383);
                printf("Y)");
                printBin("Round 1 Output Characteristic = ",R1YCHAR,BLOCKSIZE);

  return EXIT_SUCCESS;
}
```

8.4 Automated Linear Cryptanalyser Source Code

```
/**
 *========== Automated Linear Cryptanalyser ==========
 *Designed for: S-DES
 *Developer 1: Dr. K.S. Ooi (ksooi@mailexcite.com)
 *Developer 2: Brain Chin Vito (v@chin.tc)
 *University of Sheffield Centre, Taylor's College
 *
 *
 *   x0  x1  x2  x3
 *    |   |   |   |
 *  ---------------
 *  |              |
 *  |     s0       |
 *  |              |
 *  ---------------
 *      |     |
 *      y0    y1
 *
 *    r= 0    1    2    3
 *  c  -------------------
 *  0  | 1    0    2    3
 *  1  | 3    1    0    2
 *  2  | 2    0    3    1
 *  3  | 1    3    2    0
 *
 *
 *  BYTE S0[]={1,0,2,3,3,1,0,2,2,0,3,1,1,3,2,0};
 *  For row r (0 to 3, inclusive) and column c (0 to 3, inclusive)
 *  the output is S0[4*r + c]
 *
 */

#include <stdio.h>
#include <stdlib.h>
#include <limits.h>
#include <string.h>
#include <math.h>

#define UINT unsigned int
#define BYTE unsigned char
#define BYTESIZE CHAR_BIT
#define BLOCKSIZE BYTESIZE
```

```
#define KEYSIZE      10
#define SUBKEYSIZE   8
#define SPLITKEYSIZE 5
#define BIT unsigned int
#define INDEX unsigned int
#define COLUMN unsigned int
#define ELEMENT int

UINT cbin2UINT(char*, UINT);

/* ======================== global variables ================================= */
BYTE S0[]={1,0,2,3,3,1,0,2,2,0,3,1,1,3,2,0};
BYTE S1[]={0,3,1,2,3,2,0,1,1,0,3,2,2,1,3,0};
UINT key10  = 0;
BYTE E[] = {3,0,1,2,1,2,3,0};
BYTE P4[] = {1,0,3,2};
BIT S0I[64];
BIT S0O[32];
COLUMN IOTABLE[6];
COLUMN A[16];
COLUMN B[4];
ELEMENT DISTABLE[15][3];
BIT SUBK[4]={0,0,0,0};
BIT R0MASK[4]={0,0,0,0};
BIT R1MASK[4]={0,0,0,0};
BIT L0MASK[4]={0,0,0,0};
BIT L1MASK[4]={0,0,0,0};
BIT L2MASK[4]={0,0,0,0};
INDEX S0IPt=0;
INDEX S0OPt=0;
INDEX a=0,b=0;
double prob=0;
double ovProb=0;


/* ======================== general functions ================================= */

/**
 * E.g. of usage:
 *    printBin(" 23 is: ", 23, BYTESIZE);
 *    printBin(" ",cbin2UINT("10110010",BYTESIZE), BYTESIZE);
 *
 *   printBin(" 217 is: ", 217, BYTESIZE);
 *   printf("String 11011001 is %u\n", cbin2UINT("11011001",BYTESIZE));
 *
 *   printBin(" 729 is: ", 729, 10);
 *   printf("String 1011011001 is %u\n", cbin2UINT("1011011001",KEYSIZE));
 *
 */

void printBin(const char *str,unsigned int bInteger,unsigned int nSize){
  char s[BYTESIZE*sizeof(UINT)];
  UINT i;
  UINT n=bInteger;

  for(i=0; i<nSize; i++)
    *(s + i)='0';  *(s+i)='\0';

  i= nSize - 1;

  while(n > 0){
   s[i--]=(n % 2)? '1': '0';
   n = n/2;
  }

  printf("%s%s [%+3u in decimal]\n",str,s,bInteger);

}

/**
 * E.g. of usage:
```

```
 *   //printf("String 11011001 is %u\n", cbin2UINT("11011001",BYTESIZE));
 *   //printf("String 1011011001 is %u\n", cbin2UINT("1011011001",KEYSIZE));
 */
UINT cbin2UINT(char *s, unsigned int nSize){
  int nLen = strlen(s);
  UINT uResult = 0;

  while (--nLen >= 0)
    if(s[nLen] == '1')
       uResult = 1 << (nSize - nLen - 1)  | uResult;

  return uResult;
}

void init(){
       int i=0,j=0;
       for(i=0;i<64;i++)
              S0I[i]=00;

       for(i=0;i<6;i++)
              IOTABLE[i]=00;

       for(i=0;i<15;i++){
              for(j=0;j<3;j++)
              DISTABLE[i][j]=00;
       }

       for(i=0;i<16;i++)
              A[i]=00;

       for(i=0;i<4;i++)
              B[i]=00;
}
/**
 * Prints the given number in Binary on the Screen
 */
void printBinary(UINT theNum, int nSize){
  while(--nSize > -1)
   (1 << nSize & theNum )? printf("  %d",1):printf("  %d",0);

}

/* ======================== Key Scheduling ================================= */

/**
 *
 * Only valid for max size 10
 * Shift size is two, max
 *   printBin(" ",cbin2UINT("10110010",BYTESIZE), BYTESIZE);
 *   printBin(" ",leftShift(bin2UINT("10110010",BYTESIZE),2,BYTESIZE), BYTESIZE);
 *   printBin(" ",leftShift(bin2UINT("10110010",BYTESIZE),1,BYTESIZE), BYTESIZE);
 *
 */
UINT leftShift(UINT nKey, UINT nShift, UINT nSize){
  UINT n = nKey >> (nSize - nShift), i, nMask=0;
  nKey <<= nShift;

  for(i=0; i< nSize; i++)
     nMask |= 1 << i;

  return (nKey | n) & nMask;
}



/**
 * Permutation box p10
 *   printBin(" ",box_p10(cbin2UINT("1011011001",KEYSIZE)),KEYSIZE);
 */
UINT p10[] ={9,7,3,8,0,2,6,5,1,4};
```

```
UINT box_p10(UINT key10){
  UINT uResult=0, i=0;

  for(; i< KEYSIZE; i++)
    if (1 << (KEYSIZE - p10[i] - 1) & key10)
      uResult |= 1 << (KEYSIZE - i - 1);

  return uResult;
}

/**
 * Split Key
 *   printBin("",splitKey(cbin2UINT("1011011001",KEYSIZE), keyArray),5);
 * where keyArray is an array of 2 '5 bit' keys
 */
void splitKey(UINT p10Key10, UINT uResult[]){
  /**
   *  31 == 0000011111
   * 992 == 1111100000
   */
  UINT H_SPLIT5BIT_MASK = 31, L_SPLIT5BIT_MASK = 992;

  uResult[0] =  (p10Key10 & L_SPLIT5BIT_MASK) >> SPLITKEYSIZE;
  uResult[1] =  p10Key10 & H_SPLIT5BIT_MASK;

}

/**
 * Permutation box p8
 */
UINT p8[] = {3,1,7,5,0,6,4,2};

UINT box_p8(UINT key5[]){
  UINT uResult=0, uTemp, i=0;
  /*
   * 255 = 11111111
   */
  UINT uMask = 255;

  uTemp =  key5[0] << SPLITKEYSIZE | key5[1];
  uTemp &= uMask;

  for(; i< SUBKEYSIZE; i++)
    if (1 << (KEYSIZE - p8[i] - 1) & uTemp)
      uResult |= 1 << (KEYSIZE - i - 3);

  return uResult;
}

/**
 * Key Schedule
 */
void keySchedule(UINT key10,UINT key8[]){
  UINT key5[2]={0,0}, keyTemp, i;

  keyTemp = box_p10(key10);

  splitKey(keyTemp, key5);

  for(i=0; i<2 ; i++){
    key5[0] = leftShift(key5[0], i+1, SPLITKEYSIZE);
    key5[1] = leftShift(key5[1], i+1, SPLITKEYSIZE);

    key8[i]=box_p8(key5);
  }
}


/* ======================= IP and IP_1 ================================= */

UINT IP[]  = {7,6,4,0,2,5,1,3};
```

```
UINT IP_1[]= {3,6,4,7,2,5,1,0};


BYTE per(UINT P[], BYTE input){
  BYTE bRes = 00;
  int i = 8;

  while(--i >= 0)
    if( 01 << (BLOCKSIZE - P[BLOCKSIZE - i - 1] - 1) & input )
      bRes |= (01 << i);

  printBin(" IP ", bRes, BLOCKSIZE);
  return bRes;
}

/* ======================== Round ================================ */
void split824(BYTE bInput8, BYTE bLR[]){
  BYTE L_mask = 240, H_mask = 15;

  /** left  */
  bLR[0] = (bInput8 & L_mask) >> 4;

  /** right */
  bLR[1] = bInput8 & H_mask;

}

/**
 * f-function
 */
BYTE f(BYTE bRight, BYTE key){
  BYTE bRes = 00, bTemp;
  BYTE sLR4[]={0,0}, r, c;
  int i = SUBKEYSIZE;

  while(--i >= 0)
    if( 01 << (4 - E[SUBKEYSIZE - i - 1] - 1) & bRight )
      bRes |= (01 << i);

  bRes ^= key;


  split824(bRes,sLR4);

  c = (sLR4[0] & 6) >> 1;
  r = (sLR4[0] & 8) >> 2 | (sLR4[0] & 01);
  sLR4[0] = S0[4*r + c] << 2;

  c = (sLR4[1] & 6) >> 1;
  r = (sLR4[1] & 8) >> 2 | (sLR4[1] & 01);
  sLR4[1] = S1[4*r + c];


  bTemp = sLR4[0] | sLR4[1];

  bRes = 00;

  // permute using P4
  i=4;
  while(--i >= 0)
    if( 01 << (4 - P4[4 - i - 1] - 1) & bTemp )
      bRes |= (01 << i);

  return bRes;
}

/* ====================Encrypt/Decrypt and Compare======================== */
/**
 * Encrypt the given inputbits and returns the the XOR of all
 *  the bits as specified in the linear function involved.
```

```
 */
int crypt(BYTE inputbits){

  UINT key8[2]={0,0};
  BYTE input8 = inputbits, exInput8,i;
  /** left and Right */
  BYTE LR[] ={00,00};
  BIT compared;
  BIT left;
  BIT right;
  /** Cryptanalysis tracker */
  BYTE L0 =0;
  BYTE R0 =0;
  BYTE L1 =0;
  BYTE R1 =0;
  BYTE L2 =0;
  BYTE R1K1=0;
  BYTE R2K1=0;
  int round;

  /** display the input */
  keySchedule(key10,key8);

 // ===> Start of the round
  exInput8 = input8;

 round=0;
 for(i=0; i< 2; i++){

  /** =====> begin round */

  split824(exInput8,LR);
  if(i==0)
  {
       L0=LR[0];
    R0=LR[1];
  }
  else if(i==1)
  {
       L1=LR[0];
       R1=LR[1];
  }

  input8 = (f(LR[1],(BYTE)key8[i])^LR[0]) << 4;
  L2=LR[0];
  input8 |= LR[1];
  exInput8 = ((input8 & 240) >> 4) | ( (input8 & 15) << 4 );
  /** =====> end of round */
 }

 printf(" ========================== Cryptanalysis ========================== \n");
 printBin("K1: ", key8[0],SUBKEYSIZE);
 printBin("K2: ", key8[1],SUBKEYSIZE);
 R1K1 = (SUBK[0]&((key8[0] & 128)==128))^(SUBK[1]&((key8[0] &
64)==64))^(SUBK[2]&((key8[0] & 32)==32))^(SUBK[3]&((key8[0] & 16)==16));
 R2K1 = (SUBK[0]&((key8[1] & 128)==128))^(SUBK[1]&((key8[1] &
64)==64))^(SUBK[2]&((key8[1] & 32)==32))^(SUBK[3]&((key8[1] & 16)==16));
 printf("R1K1=%i \n",R1K1);
 printf("R2K1=%i \n",R2K1);
 printf("Expected Result = %i \n",R1K1^R2K1);
 printBin("L0: ", L0,SUBKEYSIZE);
 printBin("R0: ", R0,SUBKEYSIZE);
 printBin("L1: ", L1,SUBKEYSIZE);
 printBin("R1: ", R1,SUBKEYSIZE);
 printBin("L2: ", L2,SUBKEYSIZE);
 left=(L0MASK[0]&((L0 & 8)==8))^(L0MASK[1]&((L0 & 4)==4))^(L1MASK[0]&((L1 &
8)==8))^(L1MASK[1]&((L1 & 4)==4))^(L2MASK[0]&((L2 & 8)==8))^(L2MASK[1]&((L2 & 4)==4));
 right=(R0MASK[0]&((R0 & 8)==8))^(R0MASK[1]&((R0 & 4)==4))^(R0MASK[2]&((R0 &
2)==2))^(R0MASK[3]&((R0 & 1)==1))^(R1MASK[0]&((R1 & 8)==8))^(R1MASK[1]&((R1 &
4)==4))^(R1MASK[2]&((R1 & 2)==2))^(R1MASK[3]&((R1 & 1)==1));
 compared=left^right;
```

```
 printf("Compare: %i \n",compared);

  return compared;
}


/* ====================cryptanalytical functions======================== */

/**
 *Puts the BIT value in the current array slot and then increase the index
 *to point to the next slot. For the S-Box input array.
 */
void putS0I(BIT i){
       S0I[S0IPt]=i;
    S0IPt++;
}

void putS0O(BIT i){
       S0O[S0OPt]=i;
    S0OPt++;
}

/**
 *Puts the BIT value in the current array slot and then increase the index
 *to point to the next slot.For the S-Box output array.
 */
void putIO(UINT theNum, int nSize){
  int type=nSize;
  while(--nSize > -1){
         if(type==4)
               (1 << nSize & theNum )? putS0I(1):putS0I(0);
         else if(type==2)
               (1 << nSize & theNum )? putS0O(1):putS0O(0);
  }
}


/**
 *Constructs the S-Box I/O table
 */
void sIO(BYTE s[],int nMax){
  int i,r,c;
  UINT mask0 = 8, mask1=4, mask2=2, mask3=1;

  for(i=0; i< nMax; i++){
    r=(i & mask0)>>2 | i & mask3;
    c=(i & mask1 | i & mask2) >> 01;
    printBinary((UINT)i, 4); //input
    printBinary((UINT)s[4*r + c], 2); //output
    putIO((UINT)i, 4); //input
    putIO((UINT)s[4*r + c], 2); //output
    putchar('\n');
  }

}

/**
 *Count the number of occurences of zeros in a given column, col which is
 *represented as an integer.
 */
ELEMENT count(COLUMN col){

       ELEMENT occ=0;
    ELEMENT i=1;

       while(i != 32768){

              if((col&i)==0)
                     occ++;
              i*=2;
```

```
        }

        return occ-8;

}

/**
 * Converts the I/O table in to decimal form and then create
 * the Distribution Table.
 */
void decimalize(){

    int i,j,k,l;

    i=32768;

        for(k=0;k<64;k=k+4){
                for(j=k;j<k+4;j++){
                        l=j%4;
                        IOTABLE[l]+=i*S0I[j];
                }
                i/=2;
        }

    i=32768;

        for(k=0;k<32;k=k+2){
                for(j=k;j<k+2;j++){
                        l=j%2;
                        IOTABLE[l+4]+=i*S0O[j];
                }
                i/=2;
        }

        A[0]=0;
    A[1]=IOTABLE[3];
    A[2]=IOTABLE[2];
        A[3]=IOTABLE[2]^IOTABLE[2];
        A[4]=IOTABLE[1];
        A[5]=IOTABLE[1]^IOTABLE[3];
    A[6]=IOTABLE[1]^IOTABLE[2];
    A[7]=IOTABLE[1]^IOTABLE[2]^IOTABLE[3];
        A[8]=IOTABLE[0];
        A[9]=IOTABLE[0]^IOTABLE[3];
        A[10]=IOTABLE[0]^IOTABLE[2];
    A[11]=IOTABLE[0]^IOTABLE[2]^IOTABLE[3];
    A[12]=IOTABLE[0]^IOTABLE[1];
        A[13]=IOTABLE[0]^IOTABLE[1]^IOTABLE[3];
        A[14]=IOTABLE[0]^IOTABLE[1]^IOTABLE[2];
        A[15]=IOTABLE[0]^IOTABLE[1]^IOTABLE[2]^IOTABLE[3];
        B[0]=0;
        B[1]=IOTABLE[5];
        B[2]=IOTABLE[4];
        B[3]=IOTABLE[4]^IOTABLE[5];

        for(i=1;i<16;i++){
                for(j=1;j<4;j++){
                        DISTABLE[i][j]=count(A[i]^B[j]);
                }
        }

}


/**
 *Obtain the linear appoximation of S-Box, S0. This si done by
 *searching for the value in the distribution table with the highest
 *magnitude and then storing the column and row of that value in global
 *variables. Then the probability of the linear approximation is calculated.
 *to point to the next slot.For the S-Box output array.
 */
```

```
void linApp(){

        ELEMENT curV=0,curL=0;
    INDEX i,j;

        for(i=1;i<16;i++){
                for(j=1;j<4;j++){
                        curV=(ELEMENT)fabs((double)DISTABLE[i][j]);
                        if(curV>curL){
                                curL=curV;
                                a=i;
                                b=j;
                        }
                }
        }
        prob=((double)curL+8)/16;
        ovProb=pow(prob,2)+pow(1-prob,2);

}

/**
 *Extend the linear approximation to cover the entire cipher
 *and to create masKS for the bits involved in the linear
 *approximation. These masKS will be used by the encryption
 *function.
 *
 */
void alApp(){

INDEX k=0,i;

for(i=8;i>0;i/=2){
        if((a&i)==i){
                SUBK[k]=1;
                if(i==8){
                        R0MASK[3]=R0MASK[3]^1;
                        R1MASK[3]=R1MASK[3]^1;
                }if(i==4){
                        R0MASK[0]=R0MASK[0]^1;
                        R1MASK[0]=R1MASK[0]^1;
                }if(i==2){
                        R0MASK[1]=R0MASK[1]^1;
                        R1MASK[1]=R1MASK[1]^1;
                }if(i==1){
                        R0MASK[2]=R0MASK[2]^1;
                        R1MASK[2]=R1MASK[2]^1;
                }
        }
        k++;
}


for(i=4;i>0;i/=2){
        if((b&i)==i){
                if(i==2){
                        R1MASK[1]=R1MASK[1]^1;
                        L0MASK[1]=L0MASK[1]^1;
                        L1MASK[1]=L1MASK[1]^1;
                        L2MASK[1]=L2MASK[1]^1;
                }if(i==1){
                        R1MASK[0]=R1MASK[0]^1;
                        L0MASK[0]=L0MASK[0]^1;
                        L1MASK[0]=L1MASK[0]^1;
                        L2MASK[0]=L2MASK[0]^1;
                }
        }
}

}
```

```
/* ====================main function======================== */

int main(void){


  char cont,useKey;

  char userKey[]="0000000000";

  INDEX i;
  BYTE j;
  INDEX T=0;
  INDEX N=255;

  printf("========== Automated Linear Cryptanalyser ==========\n");
  putchar('\n');
  printf(" Do you want to use the predetermined key (y/n) ?");
  scanf("%c",&useKey);
  if(useKey!='y'){
        printf(" Please specify key to use (10 BITS) ?");
    scanf("%s",&userKey);
        key10  = cbin2UINT(userKey,KEYSIZE);
  }


  init();
  printf("========== The S-Box 0 I/O Table ==========\n");
  printf(" X3 X2 X1 X0 Y1 Y0\n");
  sIO(S0,16);
  //scanf("%c",&cont);

  decimalize();
  scanf("%c",&cont);
  printf("Press enter to continue\n");
  scanf("%c",&cont);

  printf("========= Distribution Table ============\n");
  printf("%c                  %c \n",(char)224,(char)225);
  printf("           %-11d %-11d %-11d \n",1,2,3);
  printf("----------------------------------------\n");
  for(i=1;i<16;i++){
         printf("%-11d",i);
         for(j=1;j<4;j++){
               printf("%-11d   ",DISTABLE[i][j]);
         }
         putchar('\n');
  }

  printf("Press enter to continue\n");
  scanf("%c",&cont);

  linApp();
  printf("========== Properties of Linear Approximation ==========\n");
  printf("Most Effective Linear Approximation: %c = %i , %c = %i
\n",(char)224,a,(char)225,b);
  printf("One Round Probability = %f \n",prob);
  printf("Overall Probability = %f \n",ovProb);
  alApp();
  putchar('\n');
  printf("Press enter to continue\n");
  scanf("%c",&cont);

    for(j=0;j<N;j++){
          if(crypt(j)==1)
                T++;
        }

        if(ovProb>0.5){
                if(T>(N/2))
                        printf("Result: 0 \n" );
                else
```

```
                        printf("Result: 1 \n" );
    }else{
            if(T<(N/2))
                        printf("Result: 0 \n" );
                else
                        printf("Result: 1 \n" );
        }

  return EXIT_SUCCESS;
}
```

8.5 Brute Force Attack on S-DES

Summary

The attack was successful. The entire process is automated, the user need only supply a key which he wants to assign for encryption and decryption. In some cases, the guessed key is different from the expected key, this means that the guessed key can also be used for encryption with the same results.

Test Cases

Test Key	Guessed Key
0000000000	0000000000
0000001001	0000001001
0000001111	0000001111
0010000000	0000000100
0011111111	0010011000

Conclusion

Brute force attack although can yield good results are less practical for commercial ciphers using block size of 128 bits. Though, for S-DES, brute force attack wor*KS* just fine.

8.6 Differential Cryptanalysis on S-DES: Experiment Results

Summary

The cryptanalysis was successful. The results obtained are correct. Based on the pre-determined test cases, the attack produces 100% correct guesses of subkeys. The entire process is automated, from generation of S-Box difference table to the counting procedure.

This experiment uses 95 encryptions. Lower amount of encryptions had also been tested with correct results.

Parameter Information

Round 1 Input Differential Characteristic, $DU_1 = 4$
Round 1 Output Differential Characteristic, $DV_1 = 69$
Modified expansion E = [0 2 1 3 0 1 2 3]

Test Cases

Test Key	Actual Subkey	Guessed Subkey	Count
0000000000	00000000	00000000	95/95
0000001001	00000001	00000001	95/95
0000001111	10000001	10000001	95/95

0010000000	00100000	00100000	95/95
0011111111	10100111	10100111	95/95

Conclusion

The cryptanalysis has found 8 bits of the subkey of the last round. These subkey bits are the actual 8 bits of the S-DES 10 key bits, 2 bits are still missing. Thus, we can now try all the 2^2 possibilities of the missing bits which is trivial.

8.7 Linear Cryptanalysis on S-DES: Experiment Results

Summary

The attack was successful. Based on the pre-determined test cases, the attack produces correct guesses of the target partial subkey. The entire process is automated, from generation of I/O Table, distribution table to the actual count of encryptions that satisfies the linear equation .

255 encryptions were used in this test. Lower amount of encryptions had also been tested to yield correct results. Though, decreasing the amount of encryptions will lead to a lower success rate[MM94].

Parameter Information

$\alpha = 5$
$\beta = 1$

Linear Approximation Used: $L_0^0 \oplus L_0^1 \oplus L_0^2 \oplus R_0^0 \oplus R_2^0 \oplus R_2^1 = K_1^1 \oplus K_3^1 \oplus K_1^2 \oplus K_3^2$

Probability: 0.78125

Test Cases

Test Key	Actual $K_1^1 \oplus K_3^1 \oplus K_1^2 \oplus K_3^2$	Cryptanalysis Result $K_1^1 \oplus K_3^1 \oplus K_1^2 \oplus K_3^2$
0000000000	0	0
0000001001	1	1
0000001111	1	1
0010000000	0	0
0011111111	1	1

Conclusion

The linear cryptanalysis performed was successful. The results helps in lowering the amount of computational steps required for an exhaustive key search. Although the success rate is high, the probability of success depends greatly on the amount of plaintexts used [MK94].

8.8 Difference Pair of S_0

X	***Y***	*DY* given *DX*															
		0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**
0000	01	00	10	01	00	11	01	10	11	11	00	01	10	10	11	00	01
0001	11	00	10	10	11	11	01	01	00	10	01	00	11	01	00	11	10
0010	00	00	01	01	11	11	10	10	00	00	11	10	01	01	00	11	10
0011	01	00	01	10	00	11	10	01	11	10	01	00	11	01	00	11	10
0100	10	00	10	01	00	11	01	10	11	01	00	11	10	00	11	10	01
0101	00	00	10	10	11	11	01	01	00	10	11	00	01	01	10	11	00
0110	11	00	01	01	11	11	10	10	00	10	11	00	01	11	00	01	10
0111	10	00	01	10	00	11	10	01	11	10	11	00	01	01	10	11	00
1000	10	00	11	10	01	01	00	11	10	11	01	10	11	00	10	01	00
1001	01	00	11	10	01	11	10	01	00	10	00	00	01	01	11	11	10
1010	00	00	11	10	01	01	00	11	10	00	01	01	11	11	10	10	00
1011	11	00	11	10	01	11	10	01	00	10	11	00	10	01	00	11	01
1100	11	00	01	10	11	01	10	11	00	01	11	00	01	10	00	11	10
1101	10	00	01	10	11	11	00	01	10	10	00	00	01	01	11	11	10
1110	01	00	01	10	11	01	10	11	00	10	11	11	01	01	00	00	10
1111	00	00	01	10	11	11	00	01	10	10	11	00	10	01	00	11	01

where *DX* is in hexadecimal,

8.9 Difference Pair of S_1

X	***Y***	*DY* given *DX*															
		0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**
0000	01	00	10	01	00	10	01	11	11	11	10	00	01	00	00	00	11
0001	11	00	10	10	11	11	00	01	01	00	01	11	10	11	11	01	10
0010	00	00	01	01	11	10	10	11	00	01	00	10	11	10	10	00	01
0011	01	00	01	10	00	11	11	01	10	01	00	10	11	00	00	00	01
0100	10	00	11	01	01	10	00	11	10	11	10	10	01	00	00	10	11
0101	00	00	11	10	10	11	01	01	00	01	00	10	01	10	10	00	01
0110	11	00	00	01	10	10	11	11	01	11	00	10	11	10	10	00	01
0111	10	00	00	10	01	11	10	01	11	00	11	11	10	11	11	01	00
1000	10	00	01	11	10	10	11	11	00	11	01	10	11	10	10	00	00
1001	01	00	01	11	10	10	11	01	10	00	10	10	11	00	00	00	01
1010	00	00	01	11	10	00	11	01	00	01	00	00	10	11	11	10	01
1011	11	00	01	11	10	10	01	01	00	01	00	11	01	10	10	00	11
1100	11	00	01	01	10	10	11	01	00	11	00	10	11	11	11	00	01
1101	10	00	01	11	00	10	11	01	00	01	10	11	11	00	00	00	01
1110	01	00	11	01	00	00	01	11	10	11	11	10	01	00	00	00	10
1111	00	00	11	11	10	10	11	01	00	00	00	10	01	10	01	01	11

where *DX* is in hexadecimal,

8.10 Difference Distribution Table for S_0

Input Difference *DX*	**Output Difference *DY***			
	0	**1**	**2**	**3**
0	16	0	0	0
1	0	8	4	4
2	0	4	12	0
3	4	4	0	8
4	0	4	0	12
5	4	4	8	0
6	0	8	4	4
7	8	0	4	4
8	2	2	10	2
9	4	4	0	8
10	10	2	2	2
11	0	8	4	4
12	2	10	2	2
13	8	0	4	4
14	2	2	2	10
15	4	4	8	0

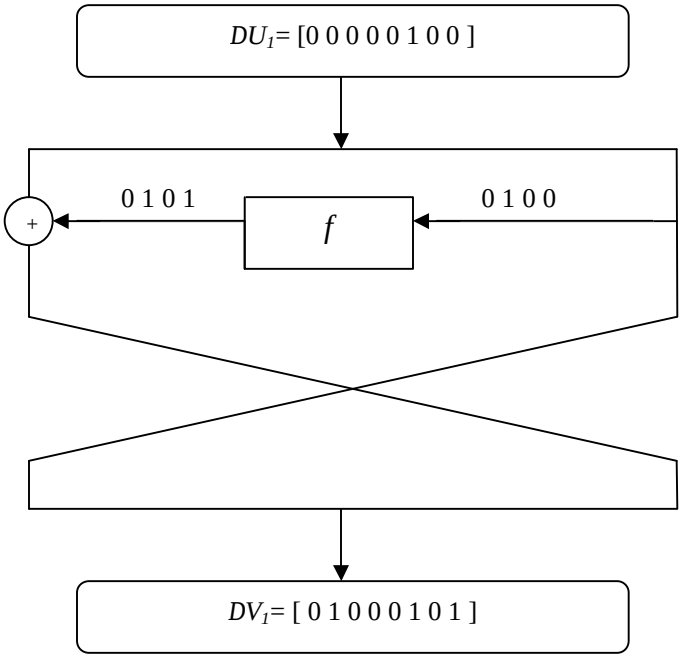
where ΔX, ΔY shown in hexadecimal.

8.11 Difference Distribution Table for S_0

Input Difference *DX*	**Output Difference *DY***			
	0	**1**	**2**	**3**
0	16	0	0	0
1	2	8	2	4
2	0	6	4	6
3	4	2	8	2
4	2	0	10	4
5	2	4	2	8
6	0	1	0	6
7	8	2	4	2
8	4	6	0	6
9	8	2	4	2
10	2	0	10	4
11	0	6	4	6
12	0	6	4	6
13	6	0	6	4
14	1	4	2	0
15	2	8	2	4

where ΔX, ΔY shown in hexadecimal.

8.12 Differential Characteristic for S-DES

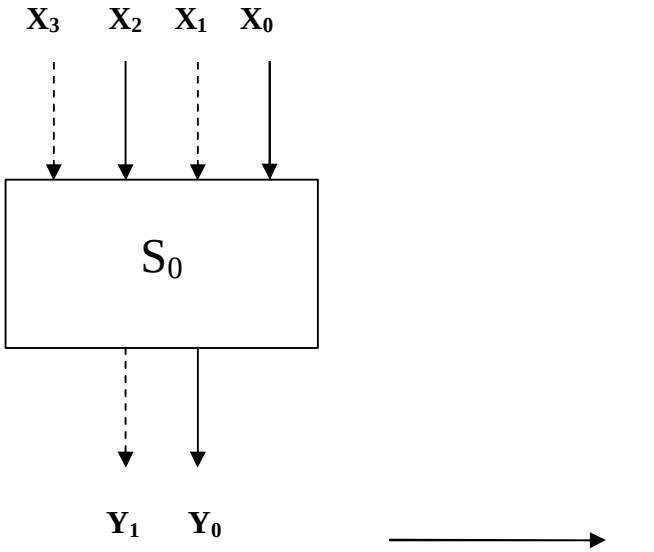


Probability = 15/32

8.13 I/O Table for S_0

X_3	X_2	X_1	X_0	Y_1	Y_0
0	0	0	0	0	1
0	0	0	1	1	1
0	0	1	0	0	0
0	0	1	1	0	1
0	1	0	0	1	0
0	1	0	1	0	0
0	1	1	0	1	1
0	1	1	1	1	0
1	0	0	0	1	0
1	0	0	1	0	1
1	0	1	0	0	0
1	0	1	1	1	1
1	1	0	0	1	1
1	1	0	1	1	0
1	1	1	0	0	1
1	1	1	1	0	0

8.14 Visualisation of Linear Approximation for S_0



Linear Approximation of S_0

8.15 Distribution table for S_0

a	b		
	1	**2**	**3**
1	0	0	0
2	0	0	-2
3	0	-2	-2
4	-2	2	0
5	6	2	0
6	-2	0	-2
7	-2	0	-2
8	0	0	0
9	0	0	0
10	0	2	2
11	0	2	-6
12	-2	2	0
13	-2	2	0
14	-2	-4	2
15	-2	4	2

8.16 Notation Table

Symbol	Denotation
$\cdot$	Bitwise AND operation
p	Permutation
s	Substitution
r	Round
C	Ciphertext
P	Plaintext
E	Encryption Function
D	Decryption Function
K	Key
(X',X'')	Input Pairs
DU	Input Difference
DV	Output Difference
R_b^r	Low part of plaintext/ciphertext block. r denotes the round involved. b is the index of the bit where 0 is the leftmost bit.
L_b^r	High part of plaintext/ciphertext block. . r denotes the round involved. b is the index of the bit where 0 is the leftmost bit.