UNIT-2

Linear Cryptanalysis, Differential Cryptanalysis, DES, Triple DES, Message Authentication and Digital Signatures, Attacks on Protocols, Elliptic Curve Architecture and Cryptography, Public Key Cryptography and RSA, , Evaluation criteria for AES, Key Management, Authentication requirements Digital forensics including digital evidence handling: Media forensics, Cyber forensics, Software forensics, Mobile forensics.

Linear Cryptanalysis:

linear cryptanalysis is a general form of [cryptanalysis](https://en.wikipedia.org/wiki/Cryptanalysis) based on finding [affine](https://en.wikipedia.org/wiki/Affine_transformation) approximations the action of a [cipher](https://en.wikipedia.org/wiki/Cipher). Attacks have been developed for [block ciphers](https://en.wikipedia.org/wiki/Block_cipher) and [stream ciphers](https://en.wikipedia.org/wiki/Stream_cipher). Linear cryptanalysis is one of the two most widely used attacks on block ciphers; the other being [differential cryptanalysis](https://en.wikipedia.org/wiki/Differential_cryptanalysis).

The attack on DES is not generally practical, requiring 247 [known plaintexts](https://en.wikipedia.org/wiki/Known-plaintext_attack).A variety of refinements to the attack have been suggested, including using multiple linear approximations or incorporating non-linear expressions, leading to a generalized [partitioning cryptanalysis](https://en.wikipedia.org/wiki/Partitioning_cryptanalysis). Evidence of security against linear cryptanalysis is usually expected of new cipher designs

## Overview

## There are two parts to linear cryptanalysis. The first is to construct linear equations relating plaintext, cipher text and key bits that have a high bias; that is, whose probabilities of holding (over the space of all possible values of their variables) are as close as possible to 0 or 1. The second is to use these linear equations in conjunction with known plaintext-cipher text pairs to derive key bits.

### Constructing linear equations

### For the purposes of linear cryptanalysis, a linear equation expresses the equality of two expressions which consist of binary variables combined with the exclusive-or (XOR) operation. For example, the following equation, from a hypothetical cipher, states the XOR sum of the first and third plaintext bits (as in a block cipher's block) and the first ciphertext bit is equal to the second bit of the key:

{\displaystyle P\_{1}\oplus P\_{3}\oplus C\_{1}=K\_{2}.}In an ideal cipher, any linear equation relating plaintext, ciphertext and key bits would hold with probability 1/2. Since the equations dealt with in linear cryptanalysis will vary in probability, they are more accurately referred to as linear *approximations*.

The procedure for constructing approximations is different for each cipher. In the most basic type of block cipher, a [substitution-permutation network](https://en.wikipedia.org/wiki/Substitution-permutation_network), analysis is concentrated primarily on the [S-boxes](https://en.wikipedia.org/wiki/S-box), the only nonlinear part of the cipher (i.e. the operation of an S-box cannot be encoded in a linear equation). For small enough S-boxes, it is possible to enumerate every possible linear equation relating the S-box's input and output bits, calculate their biases and choose the best ones. Linear approximations for S-boxes then must be combined with the cipher's other actions, such as permutation and key mixing, to arrive at linear approximations for the entire cipher. The [piling-up lemma](https://en.wikipedia.org/wiki/Piling-up_lemma) is a useful tool for this combination step. There are also techniques for iteratively improving linear approximations (Matsui 1994).

### Deriving key bits

Having obtained a linear approximation of the form:

{\displaystyle P\_{i\_{1}}\oplus P\_{i\_{2}}\oplus \cdots \oplus C\_{j\_{1}}\oplus C\_{j\_{2}}\oplus \cdots =K\_{k\_{1}}\oplus K\_{k\_{2}}\oplus \cdots }we can then apply a straightforward algorithm (Matsui's Algorithm 2), using known plaintext-cipher text pairs, to guess at the values of the key bits involved in the approximation.

For each set of values of the key bits on the right-hand side (referred to as a *partial key*), count how many times the approximation holds true over all the known plaintext-ciphertext pairs; call this count *T*. The partial key whose *T* has the greatest [absolute difference](https://en.wikipedia.org/wiki/Absolute_difference) from half the number of plaintext-cipher text pairs is designated as the most likely set of values for those key bits. This is because it is assumed that the correct partial key will cause the approximation to hold with a high bias. The magnitude of the bias is significant here, as opposed to the magnitude of the probability itself.

This procedure can be repeated with other linear approximations, obtaining guesses at values of key bits, until the number of unknown key bits is low enough that they can be attacked with [brute force](https://en.wikipedia.org/wiki/Brute-force_attack).



**The Data Encryption Standard**

The most widely used encryption scheme is based on the Data Encryption Standard (DES) adopted in 1977 by the National Bureau of Standards, now the National Institute of Standards and Technology (NIST), as Federal Information Processing Standard 46 (FIPS PUB 46). The algorithm itself is referred to as the Data Encryption Algorithm (DEA).For DES, data are encrypted in 64-bit blocks using a 56-bit key. The algorithm transforms 64-bit input in a series of steps into a 64-bit output. The same steps, with the same key, are used to reverse the encryption.

The DES enjoys widespread use. It has also been the subject of much controversy concerning how secure the DES is. To appreciate the nature of the controversy, let us quickly review the history of the DES.

In the late 1960s, IBM set up a research project in computer cryptography led by Horst Feistel. The project concluded in 1971 with the development of an algorithm with the designation.

LUCIFER is a Feistel block cipher that operates on blocks of 64 bits, using a key size of 128 bits. Because of the promising results produced by the LUCIFER project, IBM embarked on an effort to develop a marketable commercial encryption product that ideally could be implemented on a single chip. The effort was headed by Walter Tuchman and Carl Meyer, and it involved not only IBM researchers but also outside consultants and technical advice from NSA. The outcome of this effort was a refined version of LUCIFER that was more resistant to cryptanalysis but that had a reduced key size of 56 bits, to fit on a single chip.

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

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

DES, it remains important to understand the DES cipher.

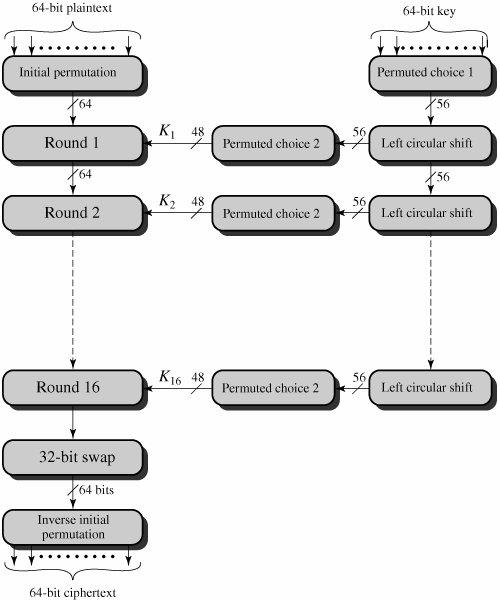
**DES Encryption**

The overall scheme for DES encryption is illustrated in Figure 2.1. As with any encryption scheme, there are two inputs to the encryption function: the plaintext to be encrypted and the key. In this case, the

Plain text must be 64 bits in length and the key is 56 bits in length.

Actually, the function expects a 64-bit key as input. However, only 56 of these bits are ever used; the other 8 bits can be used as parity bits or simply set arbitrarily.

**Figure2 .1. General Depiction of DES Encryption Algorithm**



Looking at the left-hand side of the figure, we can see that the processing of the plaintext proceeds in three phases. First, the 64-bit plaintext passes through an initial permutation (IP) that rearranges the bits to produce the permuted input. This is followed by a phase consisting of 16 rounds of the same function, which involves both permutation and substitution functions. The output of the last (sixteenth) round consists of 64 bits that are a function of the input plaintext and the key. The left and right halves of the output are swapped to produce the pre output. Finally, the pre output is passed through a permutation (IP-1) that is the inverse of the initial permutation function, to produce the 64-bit ciphertext. With the exception of the initial and final permutations, DES has the exact structure of a Feistel cipher, as shown in Figure 2.2.

The right-hand portion of Figure 3.4 shows the way in which the 56-bit key is used. Initially, the key is passed through a permutation function. Then, for each of the 16 rounds, a *subkey* (*Ki*) is produced by

the combination of a left circular shift and a permutation. The permutation function is the same for each round, but a different subkey is produced because of the repeated shifts of the key bits.

**Initial Permutation**

The initial permutation and its inverse are defined by tables, as shown in Tables 2.2a and 2.2b,

respectively. The tables are to be interpreted as follows. The input to a table consists of 64 bits numbered from 1 to 64. The 64 entries in the permutation table contain a permutation of the numbers from 1 to 64. Each entry in the permutation table indicates the position of a numbered input bit in the output, which also consists of 64 bits.

**Table 2.2. Permutation Tables for DES**

**(a) Initial Permutation (IP)**

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

**(b) Inverse Initial Permutation (IP1)**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 40 | 8 | 48 | 16 | 56 | 24 | 64 | 32 |
|  |  |  |  |  |  |  |  |
| 39 | 7 | 47 | 15 | 55 | 23 | 63 | 31 |
|  |  |  |  |  |  |  |  |
| 38 | 6 | 46 | 14 | 54 | 22 | 62 | 30 |
|  |  |  |  |  |  |  |  |
| 37 | 5 | 45 | 13 | 53 | 21 | 61 | 29 |
|  |  |  |  |  |  |  |  |
| 36 | 4 | 44 | 12 | 52 | 20 | 60 | 28 |
|  |  |  |  |  |  |  |  |
| 35 | 3 | 43 | 11 | 51 | 19 | 59 | 27 |
|  |  |  |  |  |  |  |  |

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 34 |  | 2 | 42 | 10 | 50 | 18 | 58 |  | 26 |
|  |  |  |  |  |  |  |  |  |  |
| 33 |  | 1 | 41 | 9 | 49 | 17 | 57 |  | 25 |
|  |  |  |  |  |  |  |  |  |  |
|  | **(c) Expansion Permutation (E)** | | | | | | | |  |
|  |  | |  |  |  |  |  |  |  |
|  |  | 32 | 1 | 2 | 3 | 4 | 5 |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  | 4 | 5 | 6 | 7 | 8 | 9 |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  | 8 | 9 | 10 | 11 | 12 | 13 |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  | 12 | 13 | 14 | 15 | 16 | 17 |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  | 16 | 17 | 18 | 19 | 20 | 21 |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  | 20 | 21 | 22 | 23 | 24 | 25 |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  | 24 | 25 | 26 | 27 | 28 | 29 |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  | 28 | 29 | 30 | 31 | 32 | 1 |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  | **(d) Permutation Function (P)** | | | | | | |  |
|  |  | |  |  |  |  |  |  |  |
| 16 |  | 7 | 20 | 21 | 29 | 12 | 28 |  | 17 |
|  |  |  |  |  |  |  |  |  |  |
| 1 |  | 15 | 23 | 26 | 5 | 18 | 31 |  | 10 |
|  |  |  |  |  |  |  |  |  |  |
| 2 |  | 8 | 24 | 14 | 32 | 27 | 3 |  | 9 |
|  |  |  |  |  |  |  |  |  |  |
| 19 |  | 13 | 30 | 6 | 22 | 11 | 4 |  | 25 |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |

To see that these two permutation functions are indeed the inverse of each other, consider

the following 64-bit input *M*:

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

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

where *Mi* is a binary digit. Then the permutation *X* = IP(*M*) is as follows:

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

**Details of Single Round**

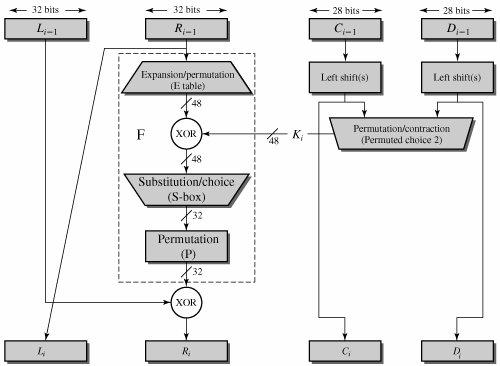
Figure shows the internal structure of a single round. Again, begin by focusing on the left-hand side

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

*Li* = *Ri*-1

*Ri* = L*i*-1x F(*Ri*-1, *Ki*)

**Figure 2.3. Single Round of DES Algorithm**



The round key *Ki* is 48 bits. The *R* input is 32 bits. This *R* input is first expanded to 48 bits by using a

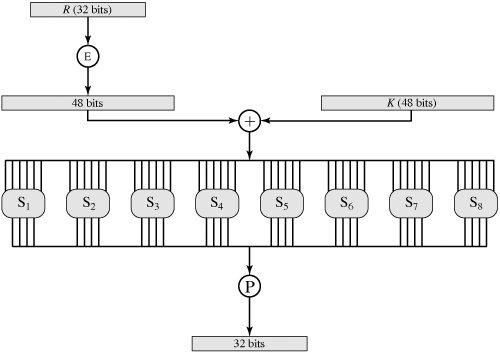
table that defines a permutation plus an expansion that involves duplication of 16 of the *R* bits (Table 2.2c). The resulting 48 bits are XORed with *Ki*. This 48-bit result passes through a substitution function

that produces a 32-bit output, which is permuted as defined by Table **3.2d.**

The role of the S-boxes in the function F is illustrated in Figure 3.6. The substitution consists of a set of

eight S-boxes, each of which accepts 6 bits as input and produces 4 bits as output. These transformations are defined in Table 3.3, which is interpreted as follows: The first and last bits of the input to box S*i* form a 2-bit binary number to select one of four substitutions defined by the four rows in the table for S*i*. The middle four bits select one of the sixteen columns. The decimal value in the cell selected by the row and column is then converted to its 4-bit representation to produce the output. For example, in S1 for input 011001, the row is 01 (row 1) and the column is 1100 (column 12). The value in row 1, column 12 is 9, so the output is 1001.

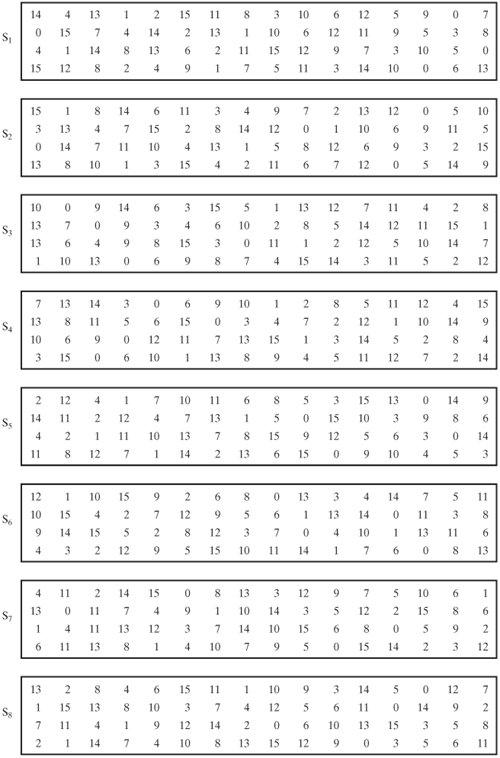
**Figure 2.4 Calculation of F(R, K)**



**Table 2.3. Definition of DES S-Boxes**

**(This item is displayed on page 79 in the print version)**

[[View full size image](file:///D:/1/0131873164/images/03tab03_alt.jpg)]



Each row of an S-box defines a general reversible substitution. Figure 2.3 may be useful in understanding the mapping. The figure shows the substitution for row 0 of box S1.

The operation of the S-boxes is worth further comment. Ignore for the moment the contribution of the key (*Ki*). If you examine the expansion table, you see that the 32 bits of input are split into groups of 4 bits, and then become groups of 6 bits by taking the outer bits from the two adjacent groups. For example, if part of the input word is

... efgh ijkl mnop ...

this becomes

... defghi hijklm lmnopq ...

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

**Key Generation**

Returning to Figures 3.4 and 3.5, we see that a 64-bit key is used as input to the algorithm. The bits of the key are numbered from 1 through 64; every eighth bit is ignored, as indicated by the lack of shading in Table 3.4a. The key is first subjected to a permutation governed by a table labeled Permuted Choice One (Table 3.4b). The resulting 56-bit key is then treated as two 28-bit quantities, labeled *C*0 and *D*0. At each round, *Ci*-1 and *Di*-1 are separately subjected to a circular left shift, or rotation, of 1 or 2 bits, as governed by Table 3.4d. These shifted values serve as input to the next round. They also serve as input to Permuted Choice Two (Table 3.4c), which produces a 48-bit output that serves as input to the function F(*Ri*-1, *Ki*).

**Table2.4. DES Key Schedule Calculation**

**(a) Input Key**

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | 1 |  | 2 |  | 3 |  | 4 |  | 5 |  | 6 |  | 7 |  | 8 |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  | 9 |  | 10 |  | 11 |  | 12 |  | 13 |  | 14 |  | 15 |  | 16 |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  | 17 |  | 18 |  | 19 |  | 20 |  | 21 |  | 22 |  | 23 |  | 24 |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | 25 |  | 26 |  | 27 |  | 28 |  | 29 |  | 30 |  | 31 |  | 32 |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  | 33 |  | 34 |  | 35 |  | 36 |  | 37 |  | 38 |  | 39 |  | 40 |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  | 41 |  | 42 |  | 43 |  | 44 |  | 45 |  | 46 |  | 47 |  | 48 |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  | 49 |  | 50 |  | 51 |  | 52 |  | 53 |  | 54 |  | 55 |  | 56 |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  | 57 |  | 58 |  | 59 |  | 60 |  | 61 |  | 62 |  | 63 |  | 64 |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |

**(b) Permuted Choice One (PC-1)**

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

**(c) Permuted Choice Two (PC-2)**

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | 14 |  | 17 |  | 11 |  | 24 |  | 1 |  |  | 5 |  | 3 |  | 28 |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  | 15 |  | 6 |  | 21 |  | 10 |  | 23 |  |  | 19 |  | 12 |  | 4 |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  | 26 |  | 8 |  | 16 |  | 7 |  | 27 |  |  | 20 |  | 13 |  | 2 |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  | 41 |  | 52 |  | 31 |  | 37 |  | 47 |  |  | 55 |  | 30 |  | 40 |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  | 51 |  | 45 |  | 33 |  | 48 |  | 44 |  |  | 49 |  | 39 |  | 56 |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  | 34 |  | 53 |  | 46 |  | 42 |  | 50 |  |  | 36 |  | 29 |  | 32 |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  | **(d) Schedule of Left Shifts** | | | | | | |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| Round number | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |  | 10 | 11 | 12 | 13 | 14 | 15 | 16 |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| Bits rotated | 1 | 1 | 2 | 2 | 2 | 2 | 2 | 2 | 1 |  | 2 | 2 | 2 | 2 | 2 | 2 | 1 |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |

**DES Decryption**

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

**The Avalanche Effect**

A desirable property of any encryption algorithm is that a small change in either the plaintext or the key should produce a significant change in the ciphertext. In particular, a change in one bit of the plaintext or one bit of the key should produce a change in many bits of the ciphertext. If the change were small, this might provide a way to reduce the size of the plaintext or key space to be searched.

DES exhibits a strong avalanche effect. Table 3.5 shows some results taken from [KONH81]. In Table 3.5a, two plaintexts that differ by one bit were used:

00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000

10000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000

with the key

0000001 1001011 0100100 1100010 0011100 0011000 0011100 0110010

**Table 2.5. Avalanche Effect in DES**

|  |  |  |  |
| --- | --- | --- | --- |
| **(a) Change in Plaintext** | |  | **(b) Change in Key** |
|  |  |  |  |
| **Round** | **Number of bits that differ** | **Round** | **Number of bits that differ** |
|  |  |  |  |
| 0 | 1 | 0 | 0 |
|  |  |  |  |
| 1 | 6 | 1 | 2 |
|  |  |  |  |
| 2 | 21 | 2 | 14 |
|  |  |  |  |
| 3 | 35 | 3 | 28 |
|  |  |  |  |
| 4 | 39 | 4 | 32 |
|  |  |  |  |
| 5 | 34 | 5 | 30 |
|  |  |  |  |
| 6 | 32 | 6 | 32 |
|  |  |  |  |
| 7 | 31 | 7 | 35 |
|  |  |  |  |
| 8 | 29 | 8 | 34 |
|  |  |  |  |
| 9 | 42 | 9 | 40 |
|  |  |  |  |
| 10 | 44 | 10 | 38 |
|  |  |  |  |
| 11 | 32 | 11 | 31 |
|  |  |  |  |
|  |  |  |  |

|  |  |  |  |
| --- | --- | --- | --- |
| 12 | 30 | 12 | 33 |
|  |  |  |  |
| 13 | 30 | 13 | 28 |
|  |  |  |  |
| 14 | 26 | 14 | 26 |
|  |  |  |  |
| 15 | 29 | 15 | 34 |
|  |  |  |  |
| 16 | 34 | 16 | 35 |
|  |  |  |  |
|  |  |  |  |

The Table 2.4a shows that after just three rounds, 21 bits differ between the two blocks. On completion, the two ciphertexts differ in 34 bit positions.

Table 2.5b shows a similar test in which a single plaintext is input:

01101000 10000101 00101111 01111010 00010011 01110110 11101011 10100100

with two keys that differ in only one bit position:

1110010 1111011 1101111 0011000 0011101 0000100 0110001 11011100

0110010 1111011 1101111 0011000 0011101 0000100 0110001 11011100

Again, the results show that about half of the bits in the ciphertext differ and that the avalanche effect is pronounced after just a few rounds.

**2.3. The Strength of Des**

Since its adoption as a federal standard, there have been lingering concerns about the level of security provided by DES. These concerns, by and large, fall into two areas: key size and the nature of the algorithm.

**The Use of 56-Bit Keys**

With a key length of 56 bits, there are 256 possible keys, which is approximately 7.2 x 1016. Thus, on the face of it, a brute-force attack appears impractical. Assuming that, on average, half the key space has to be searched, a single machine performing one DES encryption per microsecond would take more than a thousand years (see Table 2.2) to break the cipher.

However, the assumption of one encryption per microsecond is overly conservative. As far back as 1977, Diffie and Hellman postulated that the technology existed to build a parallel machine with 1 million encryption devices, each of which could perform one encryption per microsecond [DIFF77]. This would bring the average search time down to about 10 hours. The authors estimated that the cost would be about $20 million in 1977 dollars.

DES finally and definitively proved insecure in July 1998, when the Electronic Frontier Foundation (EFF) announced that it had broken a DES encryption using a special-purpose "DES cracker" machine that was built for less than $250,000. The attack took less than three days. The EFF has published a detailed description of the machine, enabling others to build their own cracker [EFF98]. And, of course, hardware prices will continue to drop as speeds increase, making DES virtually worthless.

It is important to note that there is more to a key-search attack than simply running through all possible keys. Unless known plaintext is provided, the analyst must be able to recognize plaintext as plaintext. If the message is just plain text in English, then the result pops out easily, although the task of recognizing English would have to be automated. If the text message has been compressed before encryption, then recognition is more difficult. And if the message is some more general type of data, such as a numerical file, and this has been compressed, the problem becomes even more difficult to automate. Thus, to supplement the brute-force approach, some degree of knowledge about the expected plaintext is needed, and some means of automatically distinguishing plaintext from garble is also needed. The EFF approach addresses this issue as well and introduces some automated techniques that would be effective in many contexts.

**The Nature of the DES Algorithm**

Another concern is the possibility that cryptanalysis is possible by exploiting the characteristics of the DES algorithm. The focus of concern has been on the eight substitution tables, or S-boxes, that are used in each iteration. Because the design criteria for these boxes, and indeed for the entire algorithm, were not made public, there is a suspicion that the boxes were constructed in such a way that cryptanalysis is possible for an opponent who knows the weaknesses in the S-boxes. This assertion is tantalizing, and over the years a number of regularities and unexpected behaviors of the S-boxes have been discovered. Despite this, no one has so far succeeded in discovering the supposed fatal weaknesses in the S-boxes.

At least, no one has publicly acknowledged such a discovery.

**Timing Attacks**

We discuss timing attacks in more detail in Part Two, as they relate to public-key algorithms. However, the issue may also be relevant for symmetric ciphers. In essence, a timing attack is one in which information about the key or the plaintext is obtained by observing how long it takes a given implementation to perform decryptions on various ciphertexts. A timing attack exploits the fact that an encryption or decryption algorithm often takes slightly different amounts of time on different inputs. [HEVI99] reports on an approach that yields the Hamming weight (number of bits equal to one) of the

secret key. This is a long way from knowing the actual key, but it is an intriguing first step. The authors conclude that DES appears to be fairly resistant to a successful timing attack but suggest some avenues to explore. Although this is an interesting line of attack, it so far appears unlikely that this technique will ever be successful against DES or more powerful symmetric ciphers such as triple DES and AES.



**2.4. Differential and Linear Cryptanalysis**

For most of its life, the prime concern with DES has been its vulnerability to brute-force attack because of its relatively short (56 bits) key length. However, there has also been interest in finding cryptanalytic attacks on DES. With the increasing popularity of block ciphers with longer key lengths, including triple DES, brute-force attacks have become increasingly impractical. Thus, there has been increased emphasis on cryptanalytic attacks on DES and other symmetric block ciphers. In this section, we provide a brief overview of the two most powerful and promising approaches: differential cryptanalysis and linear cryptanalysis.

**Differential Cryptanalysis**

One of the most significant advances in cryptanalysis in recent years is differential cryptanalysis. In this section, we discuss the technique and its applicability to DES.

**History**

Differential cryptanalysis was not reported in the open literature until 1990. The first published effort appears to have been the cryptanalysis of a block cipher called FEAL by Murphy . This was followed by a number of papers by Biham and Shamir, who demonstrated this form of attack on a variety of encryption algorithms and hash functions.The most publicized results for this approach have been those that have application to DES. Differential cryptanalysis is the first published attack that is capable of breaking DES in less than 255 complexity. The scheme, as reported in, can successfully cryptanalyze DES with an effort on the order of 247 encryptions, requiring 247 chosen plaintexts. Although 247 is certainly significantly less than 255 the need for the adversary to find 247 chosen plaintexts makes this attack of only theoretical interest.

Although differential cryptanalysis is a powerful tool, it does not do very well against DES. The reason, according to a member of the IBM team that designed DES , is that differential cryptanalysis was known to the team as early as 1974. The need to strengthen DES against attacks using differential cryptanalysis played a large part in the design of the S-boxes and the permutation P. As evidence of the impact of these changes, consider these comparable results reported in Differential.

Cryptanalysis of an eight-round LUCIFER algorithm requires only 256 chosen plaintexts, whereas an attack on an eight-round version of DES requires 214 chosen plaintexts.

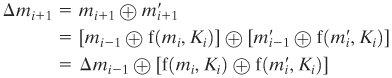
**Differential Cryptanalysis Attack**

The differential cryptanalysis attack is complex; [BIHA93] provides a complete description. The rationale behind differential cryptanalysis is to observe the behavior of pairs of text blocks evolving along each round of the cipher, instead of observing the evolution of a single text block. Here, we provide a brief overview so that you can get the flavor of the attack.

We begin with a change in notation for DES. Consider the original plaintext block *m* to consist of two halves *m*0,*m*1. Each round of DES maps the right-hand input into the left-hand output and sets the right-hand output to be a function of the left-hand input and the subkey for this round. So, at each round, only one new 32-bit block is created. If we label each new block *m*1(2  *i*  17), then the intermediate message halves are related as follows:

*mi*+1= *mi*-1  f(*mi*, *Ki*), *i* = 1, 2, ..., 16

In differential cryptanalysis, we start with two messages, *m* and *m*', with a known XOR difference *m* = *m*  *m*', and consider the difference between the intermediate message halves: *mi* = *mi*  *mi*' Thenwe have:



Now, suppose that many pairs of inputs to f with the same difference yield the same output difference if the same subkey is used. To put this more precisely, let us say that *X may cause Y with probability p*, if for a fraction *p* of the pairs in which the input XOR is *X*, the output XOR equals *Y*. We want to suppose that there are a number of values of *X* that have high probability of causing a particular output difference. Therefore, if we know *mi*-1 and *mi* with high probability, then we know *mi*+1 with high probability. Furthermore, if a number of such differences are determined, it is feasible to determine the subkey used in the function f.

The overall strategy of differential cryptanalysis is based on these considerations for a single round. The procedure is to begin with two plaintext messages *m* and *m*' with a given difference and trace through a probable pattern of differences after each round to yield a probable difference for the ciphertext. Actually, there are two probable patterns of differences for the two 32-bit halves: (*m*17||*m*16). Next, we submit *m* and *m*' for encryption to determine the actual difference under the unknown key and compare the result to the probable difference. If there is a match,

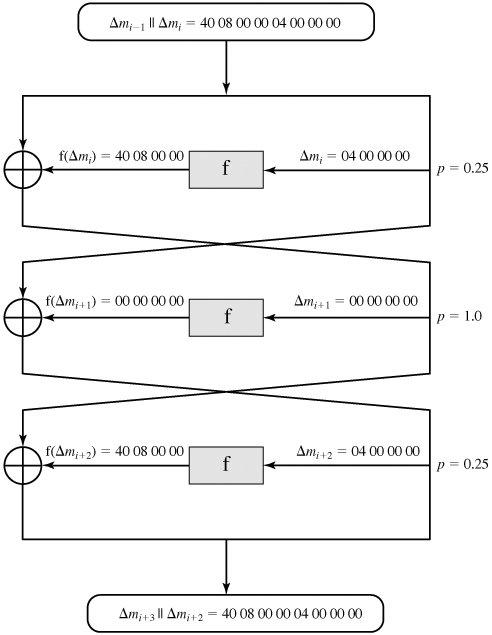
E(*K*, *m*)  E(*K*, *m*') = (*m*17||*m*16)

then we suspect that all the probable patterns at all the intermediate rounds are correct. With that assumption, we can make some deductions about the key bits. This procedure must be repeated many times to determine all the key bits.

Figure 3.7, based on a figure in [BIHA93], illustrates the propagation of differences through three

rounds of DES. The probabilities shown on the right refer to the probability that a given set of intermediate differences will appear as a function of the input differences. Overall, after three rounds the probability that the output difference is as shown is equal to 0.25 x 1 x 0.25 = 0.0625.

**Figure 2.7. Differential Propagation through Three Round of DES (numbers in hexadecimal)**



**Linear Cryptanalysis**

A more recent development is linear cryptanalysis. This attack is based on finding linear approximations to describe the transformations performed in DES. This method can find a DES key given 243 known plaintexts, as compared to 247 chosen plaintexts for differential cryptanalysis. Although this is a minor improvement, because it may be easier to acquire known plaintext rather than chosen plaintext, it still leaves linear cryptanalysis infeasible as an attack on DES. So far, little work has been done by other groups to validate the linear cryptanalytic approach.

We now give a brief summary of the principle on which linear cryptanalysis is based. For a cipher with n-bit plaintext and ciphertext blocks and an *m*-bit key, let the plaintext block be labeled P[1], ... P[*n*], the cipher text block C[1], ... C[*n*], and the key K[1], ... K[*m*]. Then define

*A*[*i, j, ..., k*] = *A*[*i*]  *A*[*j*]  ...  *A*[*k*]

The objective of linear cryptanalysis is to find an effective *linear* equation of the form:

P[1, 2, ..., a]  C[1, 2, ..., b] = K[1, 2, ..., c]

(where *x* = 0 or 1; 1  *a, b*  *n*, 1  *c*  *m*, and where the,andterms represent fixed, unique bit locations) that holds with probability *p*  0.5. The further *p* is from 0.5, the more effective the equation. Once a proposed relation is determined, the procedure is to compute the results of the left-hand side of the preceding equation for a large number of plaintext-ciphertext pairs. If the result is 0 more than half the time, assume K[1, 2, ..., c] = 0. If it is 1 most of the time, assume K[1, 2, ..., c] = 1. This gives us a linear equation on the key bits. Try to get more such relations so that we can solve for the key bits. Because we are dealing with linear equations, the problem can be approached one round of the cipher at a time, with the results combined.



**2.5. Block Cipher Design Principles**

Although much progress has been made in designing block ciphers that are cryptographically strong, the basic principles have not changed all that much since the work of Feistel and the DES design team in the early 1970s. It is useful to begin this discussion by looking at the published design criteria used in the DES effort. Then we look at three critical aspects of block cipher design: the number of rounds, design of the function F, and key scheduling.

**DES Design Criteria**

The criteria used in the design of DES, as reported in [COPP94], focused on the design of the S-boxes and on the P function that takes the output of the S boxes (Figure 3.6). The criteria for the S-boxes are as follows:

**1.**No output bit of any S-box should be too close a linear function of the input bits. Specifically, if we select any output bit and any subset of the six input bits, the fraction of inputs for which this output bit equals the XOR of these input bits should not be close to 0 or 1, but rather should be near 1/2.

**2.**Each row of an S-box (determined by a fixed value of the leftmost and rightmost input bits) should include all 16 possible output bit combinations.

**3.**If two inputs to an S-box differ in exactly one bit, the outputs must differ in at least two bits.

**4.**If two inputs to an S-box differ in the two middle bits exactly, the outputs must differ in at least two bits.

**5.**If two inputs to an S-box differ in their first two bits and are identical in their last two bits, the two outputs must not be the same.

**6.**For any nonzero 6-bit difference between inputs, no more than 8 of the 32 pairs of inputs exhibiting that difference may result in the same output difference.

**7.**This is a criterion similar to the previous one, but for the case of three S-boxes.

Coppersmith pointed out that the first criterion in the preceding list was needed because the S-boxes are the only nonlinear part of DES. If the S-boxes were linear (i.e., each output bit is a linear combination of the input bits), the entire algorithm would be linear and easily broken. We have seen this phenomenon with the Hill cipher, which is linear. The remaining criteria were primarily aimed at thwarting differential cryptanalysis and at providing good confusion properties.

The criteria for the permutation P are as follows:

**1.**The four output bits from each S-box at round *i* are distributed so that two of them affect (provide input for) "middle bits" of round (*i* + 1) and the other two affect end bits. The two middle bits of input to an S-box are not shared with adjacent S-boxes. The end bits are the two left-hand bits and the two right-hand bits, which are shared with adjacent S-boxes.

**2.**The four output bits from each S-box affect six different S-boxes on the next round, and no two affect the same S-box.

**3.**For two S-boxes *j*, *k*, if an output bit from S*j* affects a middle bit of S*k* on the next round, then an output bit from S*k* cannot affect a middle bit of S*j*. This implies that for *j* = *k*, an output bit from S*j* must not affect a middle bit of S*j*.

These criteria are intended to increase the diffusion of the algorithm.

**Number of Rounds**

The cryptographic strength of a Feistel cipher derives from three aspects of the design: the number of rounds, the function F, and the key schedule algorithm. Let us look first at the choice of the number of rounds.

The greater the number of rounds, the more difficult it is to perform cryptanalysis, even for a relatively weak F. In general, the criterion should be that the number of rounds is chosen so that known cryptanalytic efforts require greater effort than a simple brute-force key search attack. This criterion was certainly used in the design of DES. Schneier [SCHN96] observes that for 16-round DES, a differential cryptanalysis attack is slightly less efficient than brute force: the differential cryptanalysis attack requires 255.1 operations,whereas brute force requires 255. If DES had 15 or fewer rounds, differential cryptanalysis would require less effort than brute-force key search.

Recall that differential cryptanalysis of DES requires 247 *chosen* plaintext. If all you have to work with is known plaintext, then you must sort through a large quantity of known plaintext-ciphertext pairs looking for the useful ones. This brings the level of effort up to 255.1.

This criterion is attractive because it makes it easy to judge the strength of an algorithm and to compare different algorithms. In the absence of a cryptanalytic breakthrough, the strength of any algorithm that satisfies the criterion can be judged solely on key length.

**Design of Function F**

The heart of a Feistel block cipher is the function F. As we have seen, in DES, this function relies on the use of S-boxes. However, we can make some general comments about the criteria for designing F. After that, we look specifically at S-box design.

**Design Criteria for F**

The function F provides the element of confusion in a Feistel cipher. Thus, it must be difficult to "unscramble" the substitution performed by F. One obvious criterion is that F be nonlinear, as we discussed previously. The more nonlinear F, the more difficult any type of cryptanalysis will be. There are several measures of nonlinearity, which are beyond the scope of this book. In rough terms, the more difficult it is to approximate F by a set of linear equations, the more nonlinear F is.

Several other criteria should be considered in designing F. We would like the algorithm to have good avalanche properties. Recall that, in general, this means that a change in one bit of the input should produce a change in many bits of the output. A more stringent version of this is the **strict avalanche** **criterion (SAC)** [WEBS86], which states that any output bit*j*of an S-box should change with probability 1/2 when any single input bit *i* is inverted for all *i*, *j*. Although SAC is expressed in terms of S-boxes, a similar criterion could be applied to F as a whole. This is important when considering designs that do not include S-boxes.

Another criterion proposed in [WEBS86] is the **bit independence criterion (BIC)**, which states that

output bits *j* and *k* should change independently when any single input bit *i* is inverted, for all *i*, *j*, and *k*. The SAC and BIC criteria appear to strengthen the effectiveness of the confusion function.

**S-Box Design**

One of the most intense areas of research in the field of symmetric block ciphers is that of S-box design.

The papers are almost too numerous to count.[10] Here we mention some general principles. In essence, we would like any change to the input vector to an S-box to result in random-looking changes to the output. The relationship should be nonlinear and difficult to approximate with linear functions.

One obvious characteristic of the S-box is its size. An *n* x *m* S-box has *n* input bits and *m* output bits. DES has 6 x 4 S-boxes. Blowfish, described in Chapter 6, has 8 x 32 S-boxes. Larger S-boxes, by and large, are more resistant to differential and linear cryptanalysis [SCHN96]. On the other hand, the larger the dimension *n*, the (exponentially) larger the lookup table. Thus, for practical reasons, a limit of *n* equal to about 8 to 10 is usually imposed. Another practical consideration is that the larger the S-box, the more difficult it is to design it properly.

S-boxes are typically organized in a different manner than used in DES. An *n* x *m* S-box typically.consists of 2*n* rows of *m* bits each. The *n* bits of input select one of the rows of the S-box, and the *m* bits in that row are the output. For example, in an 8 x 32 S-box, if the input is 00001001, the output consists of the 32 bits in row 9 (the first row is labeled row 0).

Mister and Adams [MIST96] propose a number of criteria for S-box design. Among these are

that the S-box should satisfy both SAC and BIC. They also suggest that all linear combinations of S-box columns should be *bent*. Bent functions are a special class of Boolean functions that are highly nonlinear according to certain mathematical criteria [ADAM90]. There has been increasing interest in designing and analyzing S-boxes using bent functions.

A related criterion for S-boxes is proposed and analyzed in [HEYS95]. The authors define the

**guaranteed avalanche (GA)** criterion as follows: An S-box satisfies GA of orderif, for a 1-bit inputchange, at least output bits change. The authors conclude that a GA in the range of order 2 to order 5 provides strong diffusion characteristics for the overall encryption algorithm.

For larger S-boxes, such as 8 x 32, the question arises as to the best method of selecting the S-box entries in order to meet the type of criteria we have been discussing. Nyberg, who has written a lot about the theory and practice of S-box design, suggests the following approaches (quoted in [ROBS95b]):

* **Random:** Use some pseudorandom number generation or some table of random digits togenerate the entries in the S-boxes. This may lead to boxes with undesirable characteristics for small sizes (e.g., 6 x 4) but should be acceptable for large S-boxes (e.g., 8 x 32).
* **Random with testing:** Choose S-box entries randomly, then test the results against variouscriteria, and throw away those that do not pass.
* **Human-made:** This is a more or less manual approach with only simple mathematics to supportit. It is apparently the technique used in the DES design. This approach is difficult to carry through for large S-boxes.
* **Math-made:** Generate S-boxes according to mathematical principles. By using mathematicalconstruction, S-boxes can be constructed that offer proven security against linear and differential cryptanalysis, together with good diffusion.

A variation on the first technique is to use S-boxes that are both random and key dependent. An example of this approach is Blowfish, described in Chapter 6, which starts with S-boxes filled with pseudorandom digits and then alters the contents using the key. A tremendous advantage of key-dependent S-boxes is that, because they are not fixed, it is impossible to analyze the S-boxes ahead of time to look for weaknesses.

**Key Schedule Algorithm**

A final area of block cipher design, and one that has received less attention than S-box design, is the key schedule algorithm. With any Feistel block cipher, the key is used to generate one subkey for each round. In general, we would like to select subkeys to maximize the difficulty of deducing individual subkeys and the difficulty of working back to the main key. No general principles for this have yet been promulgated.

**Digital forensics**

**Definition - What does *Digital Forensics* mean?**

Digital forensics is the process of uncovering and interpreting electronic data. The goal of the process is to preserve any evidence in its most original form while performing a structured investigation by collecting, identifying and validating the digital information for the purpose of reconstructing past events.

**Challenges facing digital forensic investigators:**

* How does one duplicate or preserve evidence without knowing the duplication itself inherently changed the data?
* Time lines are critical for showing who did what, and when. But digital time stamps are notoriously absent, or can easily be spoofed, in digital data.
* In order to be able to state conclusively that Action A caused Result B, the concept of repeatability must be introduced. This is very difficult with digital forensics.

# Digital Evidence

Computers are used for committing crime, and, thanks to the burgeoning science of digital evidence forensics, law enforcement now uses computers to fight crime.

Digital evidence is information stored or transmitted in binary form that may be relied on in court. It can be found on a computer hard drive, a mobile phone, a personal digital assistant (PDA), a CD, and a flash card in a digital camera, among other place s. Digital evidence is commonly associated with electronic crime, or e-crime, such as child pornography or credit card fraud. However, digital evidence is now used to prosecute all types of crimes, not just e-crime..

In an effort to fight e-crime and to collect relevant digital evidence for all crimes, law enforcement agencies are incorporating the collection and analysis of digital evidence, also known as computer forensics, into their infrastructure. Law enforcement agencies are challenged by the need to train officers to collect digital evidence and keep up with rapidly evolving technologies such as computer operating systems.

## Handling Digital Evidence

## To safe the hardware, software, and data can be an important aspect not only of our investigation, but also in the prosecution of a suspected computer crime.

## • Proper procedures for search and seizure at the scene

## • Computerized log of all evidence seized at the scene .

## • Detailed return given to owner of the computer equipment

## • Report describing each action taken .

## • The only assurance of integrity is the trustworthiness of the examiners, who have the skills and knowledge to properly process electronic evidence

## • Detailed report on processing and structure of the suspect's hard drive

## • Record times and places

## • Identify materials, names and serial numbers of all equipment and computer programs used

## • In Case report contains a lot of this information, and more… (Review Encase report and process from lesson plan) • Maintain regular chain of custody, as for other types of evidence • (within limits for heat, humidity, dust, magnetic fields, etc.)

## • Evidence processing procedures (Encase)

## • Labeling-identifying storage media other than the hard drive

## • Data storage mediums (tape, cartridge, zip or floppy disk, CD-ROM), if removable, will be positively identified as follows: • Contents • Date certified/tested/examined

## • Examiner's name

## • Disk write-protected prior to review, disk copied, work on copy, not original; virus checked.

## • Label each diskette a-1, a-2, etc. • Print a directory for each diskette

## • If incriminating information found, print the file contents and label the printout with same alphanumeric • Procedures for providing discovery copies to public defender, defense attorney, etc

## . • Make a working copy of the original evidence

## • Print the report from the copy made, unless too large or too much volume. Brief in a report, submit copied electronic evidence

## • Data contained in computer storage devices and computer-readable media (magnetic tape, hard drives, removable disks such as floppies and zip disks or attachable tape backup drives) are sometimes needed as evidence in a human-readable form, such as printing.

**Media forensics**

It is scientific study into the collection, analysis, interpretation, and presentation of audio, video, and image evidence obtained during the course of investigations and litigious proceedings.

**Cyber forensics**  sometimes known as **computer forensic science** is a branch of [digital forensic science](https://en.wikipedia.org/wiki/Digital_forensics) pertaining to evidence found in computers and digital storage media. The goal of computer forensics is to examine digital media in a forensically sound manner with the aim of identifying, preserving, recovering, analyzing and presenting facts and opinions about the digital information.

Evidence from computer forensics investigations is usually subjected to the same guidelines and practices of other digital evidence. It has been used in a number of high-profile cases and is becoming widely accepted as reliable within U.S. and European court systems.

In the early 1980s personal computers became more accessible to consumers, leading to their increased use in criminal activity (for example, to help commit [fraud](https://en.wikipedia.org/wiki/Fraud)). At the same time, several new "computer crimes" were recognized (such as [hacking](https://en.wikipedia.org/wiki/Hacker_(computer_security))). The discipline of computer forensics emerged during this time as a method to recover and investigate [digital evidence](https://en.wikipedia.org/wiki/Digital_evidence) for use in court. Since then computer crime and computer related crime has grown, and has jumped 67% between 2002 and 2003.[[2]](https://en.wikipedia.org/wiki/Computer_forensics#cite_note-leigland-2) Today it is used to investigate Crimes. The discipline also features in civil proceedings as a form of information gathering (for example, [Electronic discovery](https://en.wikipedia.org/wiki/Electronic_discovery)) The scope of a forensic analysis can vary from simple information retrieval to reconstructing a series of events.

**Forensic process**

Computer forensic investigations usually follow the standard digital forensic process or phases: acquisition, examination, analysis and reporting. Investigations are performed on static data (i.e. [acquired images](https://en.wikipedia.org/wiki/Disk_imaging#Hard_drive_imaging)) rather than "live" systems. This is a change from early forensic practices where a lack of specialist tools led to investigators commonly working on live data.

**Techniques**

A number of techniques are used during computer forensics investigations and much has been written on the many techniques used by law enforcement in particular.

* **Cross-drive analysis**

A forensic technique that correlates information found on multiple [hard drives](https://en.wikipedia.org/wiki/Hard_drive). The process, still being researched, can be used to identify social networks and to perform [anomaly detection](https://en.wikipedia.org/wiki/Anomaly_detection).[[9]](https://en.wikipedia.org/wiki/Computer_forensics#cite_note-garfinkel-9)[[10]](https://en.wikipedia.org/wiki/Computer_forensics#cite_note-nsf-10)

* **Live analysis**

The examination of computers from within the operating system using custom forensics or existing [sysadmin tools](https://en.wikipedia.org/wiki/Sysadmin_tools) to extract evidence. The practice is useful when dealing with [Encrypting File Systems](https://en.wikipedia.org/wiki/Encrypting_File_System), for example, where the encryption keys may be collected and, in some instances, the logical hard drive volume may be imaged (known as a live acquisition) before the computer is shut down.

* **Deleted files**

A common technique used in computer forensics is the recovery of deleted files. Modern forensic software have their own tools for recovering or carving out deleted data. Most [operating systems](https://en.wikipedia.org/wiki/Operating_system) and [file systems](https://en.wikipedia.org/wiki/File_system) do not always erase physical file data, allowing investigators to reconstruct it from the physical [disk sectors](https://en.wikipedia.org/wiki/Disk_sector). [File carving](https://en.wikipedia.org/wiki/File_carving) involves searching for known file headers within the disk image and reconstructing deleted materials.

1. [**Stochastic forensics**](https://en.wikipedia.org/wiki/Stochastic_forensics)

A method which uses [stochastic](https://en.wikipedia.org/wiki/Stochastic) properties of the computer system to investigate activities lacking digital artifacts. Its chief use is to investigate [data theft](https://en.wikipedia.org/wiki/Data_theft).

1. [**Steganography**](https://en.wikipedia.org/wiki/Steganography)

One of the techniques used to hide data is via steganography, the process of hiding data inside of a picture or digital image.

* **Volatile data**

When seizing evidence, if the machine is still active, any information stored solely in [RAM](https://en.wikipedia.org/wiki/Random_access_memory) that is not recovered before powering down may be lost.

One application of "live analysis" is to recover RAM data (for example, using icrosoft's [COFEE](https://en.wikipedia.org/wiki/COFEE) tool, windd, [WindowsSCOPE](https://en.wikipedia.org/wiki/WindowsSCOPE)) prior to removing an exhibit. CaptureGUARD Gateway bypasses Windows login for locked computers, allowing for the analysis and acquisition of physical memory on a locked computer.

RAM can be analyzed for prior content after power loss, because the electrical charge stored in the memory cells takes time to dissipate, an effect exploited by the [cold boot attack](https://en.wikipedia.org/wiki/Cold_boot_attack). The length of time that data is recoverable is increased by low temperatures and higher cell voltages. Holding unpowered RAM below −60 °C helps preserve residual data by an order of magnitude, improving the chances of successful recovery. However, it can be impractical to do this during a field examination. Some of the tools needed to extract volatile data, however, require that a computer be in a forensic lab, both to maintain a legitimate chain of evidence, and to facilitate work on the machine. If necessary, law enforcement applies techniques to move a live, running desktop computer. These include a [mouse jiggler](https://en.wikipedia.org/w/index.php?title=Mouse_jiggler&action=edit&redlink=1), which moves the mouse rapidly in small movements and prevents the computer from going to sleep accidentally. Usually, an [uninterruptible power supply](https://en.wikipedia.org/wiki/Uninterruptible_power_supply) (UPS) provides power during transit.

* **Analysis tools**

A number of open source and commercial tools exist for computer forensics investigation. Typical forensic analysis includes a manual review of material on the media, reviewing the Windows registry for suspect information, discovering and cracking passwords, keyword searches for topics related to the crime, and extracting e-mail and pictures for review

1.CAINE(linux)

2.cofee(windows)

3.Digital forensics framework(unix)

**Software forensics**

It is the science of analyzing software [source code](https://en.wikipedia.org/wiki/Source_code) or [binary code](https://en.wikipedia.org/wiki/Binary_code) to determine whether  theft occurred. It is the centerpiece of lawsuits, trials, and settlements when companies are in dispute over issues involving software [patents](https://en.wikipedia.org/wiki/Patents), [copyrights](https://en.wikipedia.org/wiki/Copyrights), and [trade secrets](https://en.wikipedia.org/wiki/Trade_secrets). Software forensics tools can compare code to determine correlation, a measure that can be used to guide a software forensics expert.

Past methods of software forensics

Past methods of code comparison included [hashing](https://en.wikipedia.org/wiki/Hash_function), [statistical analysis](https://en.wikipedia.org/wiki/Statistical_analysis), text [matching](https://en.wikipedia.org/wiki/String_searching_algorithm), and [tokenization](https://en.wikipedia.org/wiki/Lexical_analysis). These methods compared software code and produced a single measure indicating whether copying had occurred. However, these measures were not accurate enough to be admissible in court because the results were not accurate, the algorithms could be easily fooled by simple substitutions in the code, and the methods did not take into account the fact that code could be similar for reasons other than copying.

* Copyright infringement

Following the use of software tools to compare code to determine the amount of correlation, an expert can use an iterative filtering process to determine that the correlated code is due to third-party code, code generation tools, commonly used names, common algorithms, common programmers, or copying. If the correlation is due to copying, and the copier did not have the authority from the rights holder, then [copyright infringement](https://en.wikipedia.org/wiki/Copyright_infringement) occurred.

* Trade secret protection and infringement

Software can contain trade secrets, which provide a competitive advantage to a business. To determine trade secret theft, the same tools and processes can be used to detect copyright infringement. If code was copied without authority, and that code has the characteristics of a trade secret—it is not generally known, the business keeps it secret, and its secrecy maintains its value to the business—then the copied code constitutes trade secret theft.

Trade secret theft can also involve the taking of code functionality without literally copying the code. Comparing code functionality is a very difficult problem that has yet to be accomplished by any algorithm in reasonable time. For this reason, finding the theft of code functionality is still mostly a manual process.

* Patent infringement

As with trade secret functionality, it is not currently possible to scientifically detect software patent infringement, as software patents cover general implementation rather than specific source code. For example, a program that implements a patented invention can be written in many available programming languages, using different function names and variable names and performing operations in different sequences. There are so many combinations of ways to implement inventions in software that even the most powerful modern computers cannot consider all combinations of code that might infringe a patent. This work is still left to human experts using their knowledge and experience, but it is a problem that many in software forensics are trying to automate by finding an algorithm or simplifying process.

* Objective facts before subjective evidence

One important rule of any forensic analysis is that the objective facts must be considered first. Reviewing comments in the code or searching the Internet to find information about the companies that distribute the code and the programmers who wrote the code are useful only after the objective facts regarding correlation have been considered. Once an analysis has been performed using forensic tools and procedures, analysts can then begin looking at subjective evidence like comments in the code. If the information in that subjective evidence conflicts with the objective analysis, analysts need to doubt the subjective evidence. Fake copyright notices, open source notifications, or programmer names that were added to source code after copying took place, in order to disguise the copying, are not uncommon in real-world cases of code theft.