SIMPLE DES

Simple DES is a block cipher which encrypts an 8-bit block of plaintext using a 10-bit key and outputs an 8-bit block of ciphertext.

The encryption algorithm involves five functions executed in the following order:
1. an initial permutation IP,
2. a function f_K,
3. a switch function SW that switches two halves,
4. the function f_K again,
5. the inverse IP^{-1} of permutation IP.

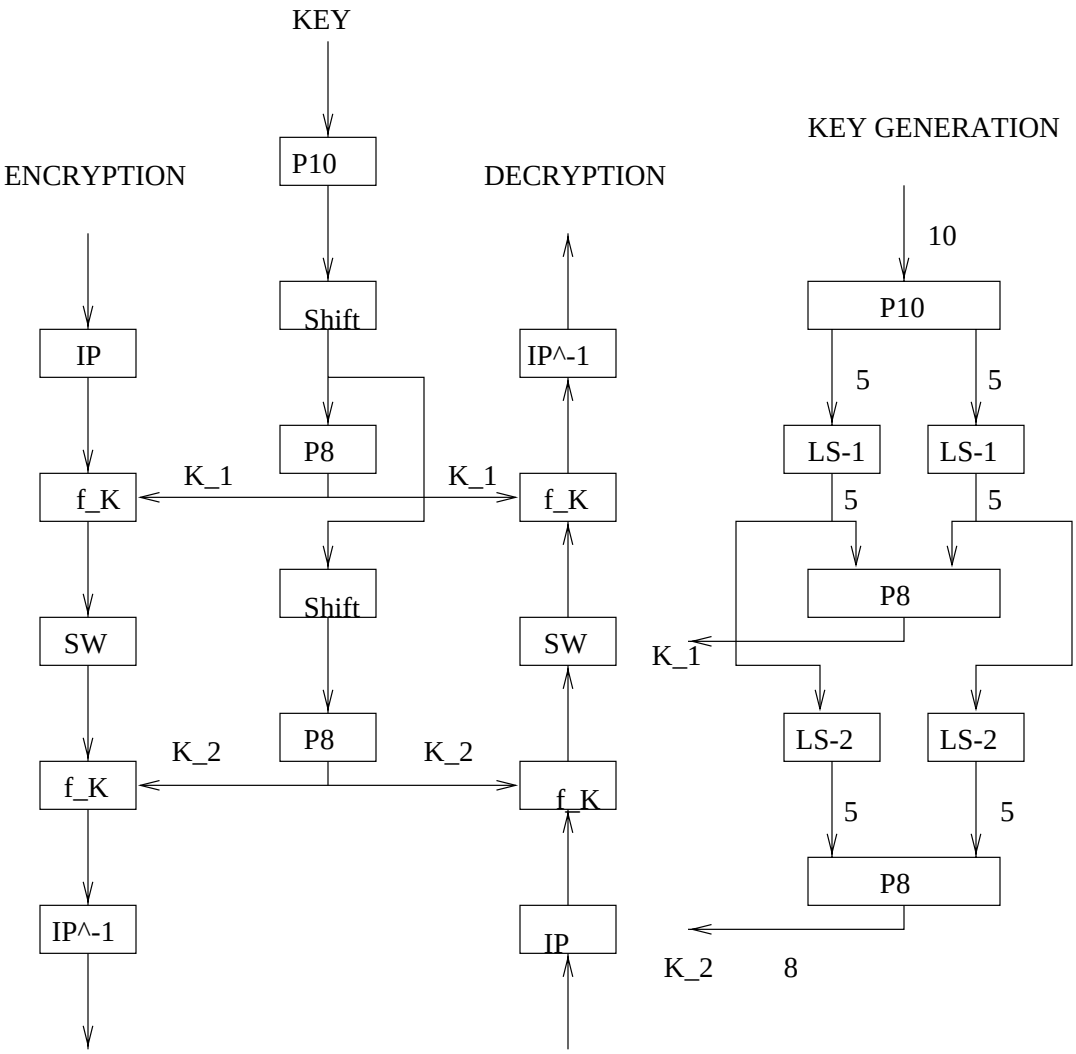
Steps 2 and 3 use keys K_1 and K_2, resp., which are generated via a key generation algorithm.

KEY GENERATION

Key generation involves three functions which are applied in a five step sequence in order to produce two subkeys:
1. a permutation $P10$ which permutes a 10-bit input,
2. a left shift operation,
3. an 8-bit permutation that produces an 8-bit output; this gives the first subkey K_1,
4. again the output from step 2 is subjected to a second double left shift
5. an 8-bit permutation that produces a second 8-bit output; this is the second subkey K_2.

Several alternatives could have been applied, like, either using a larger key or using two independent keys.

SIMPLE DES


KEY
P10
Shift
P8
Shift
P8
ENCRYPTION
IP
f_K
SW
f_K
IP^-1
K_1
K_2
DECRYPTION
IP^-1
f_K
SW
f_K
IP
K_1
K_2
KEY GENERATION
10
P10
5
5
LS-1
LS-1
5
5
P8
K_1
LS-2
LS-2
5
5
P8
K_2
8

STRUCTURE OF SIMPLE DES

S-Boxes:

$$S0 = \begin{bmatrix} 1 & 0 & 3 & 2 \\ 3 & 2 & 1 & 0 \\ 0 & 2 & 1 & 3 \\ 3 & 1 & 3 & 2 \end{bmatrix} \quad S1 = \begin{bmatrix} 0 & 1 & 2 & 3 \\ 2 & 0 & 1 & 3 \\ 3 & 0 & 1 & 0 \\ 2 & 1 & 0 & 3 \end{bmatrix}$$

Permutation P10:

$$\begin{pmatrix} 3 & 5 & 2 & 7 & 4 & 10 & 1 & 9 & 8 & 6 \\ 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 \end{pmatrix}$$

Permutation P8:

$$\begin{pmatrix} 6 & 3 & 7 & 4 & 8 & 5 & 10 & 9 \\ 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 \end{pmatrix}$$

Permutation P4:

$$\begin{pmatrix} 2 & 4 & 3 & 1 \\ 1 & 2 & 3 & 4 \end{pmatrix}$$

BASIC FUNCTIONS OF SIMPLE DES

Encryption Algorithm on Key K:

$$y = E_K(x) = IP^{-1} \circ f_{K_2} \circ SW \circ f_{K_1} \circ IP(x),$$

where

$$\begin{aligned} K_1 &= P8(Shift(P10(K)) \\ K_2 &= P8(Shift(Shift(P10(K))))) \end{aligned}$$

Example of Key Generation:

1.	**key:**	1010000010	$:= K$
2.	$P10$:	1000001100	
3.	Split:	10000	01100
4a.	L-Shift:	00001	11000
5a.	Merge:	0000111000	
6a.	$P8$:	10100100	$:= K_1$
4b.	Double L-Shift:	00100	00011
5b.	Merge:	0010000011	
6b.	$P8$:	01000011	$:= K_2$

Thus the original 10-bit key K is being used to generate two 8-bit keys K_1 and K_2.

Decryption Algorithm on Key K:

$$x = D_K(y) = IP^{-1} \circ f_{K_1} \circ SW \circ f_{K_2} \circ IP(y)$$

The functions are defined as follows:

1. Initial Permutation:

$$\mathbf{IP}: \begin{pmatrix} 2 & 6 & 3 & 1 & 4 & 8 & 5 & 7 \\ 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 \end{pmatrix}$$

2. Inverse of the Initial Permutation:

$$\mathbf{IP}^{-1}: \begin{pmatrix} 4 & 1 & 3 & 5 & 7 & 2 & 8 & 6 \\ 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 \end{pmatrix}$$

Note that $IP(IP^{-1}(x)) = IP^{-1}(IP(x)) = x$, for all x. We must show that

$$D_K(E_K(x)) = x$$

The proof of this depends on the definition of the function f_K, which we define in the sequel. A proof will be given when we outline Feistel ciphers.

3. The Switch Function SW: Interchanges the left and right 4 bits so that the second application of f_K operates on a different set of 4 bits. (In the second instance $E, S0, S1, P4$ remain the same and the key input is K_2.)

4. The Shift Function $Shift$: This is a circular left shift (rotation) by one position on the first 5 bits or last 5 bits.

5. The Function f_K:

$$f_K(L, R) = (L \oplus F(R, SK), R),$$

where L, R are the leftmost and rightmost 4-bit strings of the 8-bit input string to f_K, and F is a mapping from 4-bit strings to 4-bit strings (not necessarily $1-1$), and SK is a subkey (either K_1 or K_2 depending on the case).

Example: Assuming $F(R, SK) = 1110$ and $L = 1011, R = 1101$ we have

$$\begin{aligned} f_K(L, R) &= (L \oplus F(R, SK), R) \\ &= (1011 \oplus 1110, 1101) \\ &= (0101, 1101) \end{aligned}$$

5a. Expansion Operation E: expands a four bit string into an 8-bit string

$$\begin{pmatrix} 4 & 1 & 2 & 3 & 2 & 3 & 4 & 1 \\ 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 \end{pmatrix}$$

(so E is not a permutation). The output on input $n_1n_2n_3n_4$ is represented by

$$\mathbf{E}(n_1n_2n_3n_4) = \begin{matrix} n_4 & n_1 & n_3 & n_4 \\ n_2 & n_3 & n_4 & n_1 \end{matrix}$$

$K_1 = (k_{11}, k_{12}, k_{13}, k_{14}, k_{15}, k_{16}, k_{17}, k_{18})$ is now XOR-ed to obtain

$$\begin{matrix} n_4 \oplus k_{11} & n_1 \oplus k_{12} & n_3 \oplus k_{13} & n_4 \oplus k_{14} \\ n_2 \oplus k_{15} & n_3 \oplus k_{16} & n_4 \oplus k_{17} & n_1 \oplus k_{18} \end{matrix}$$

which is abbreviated by

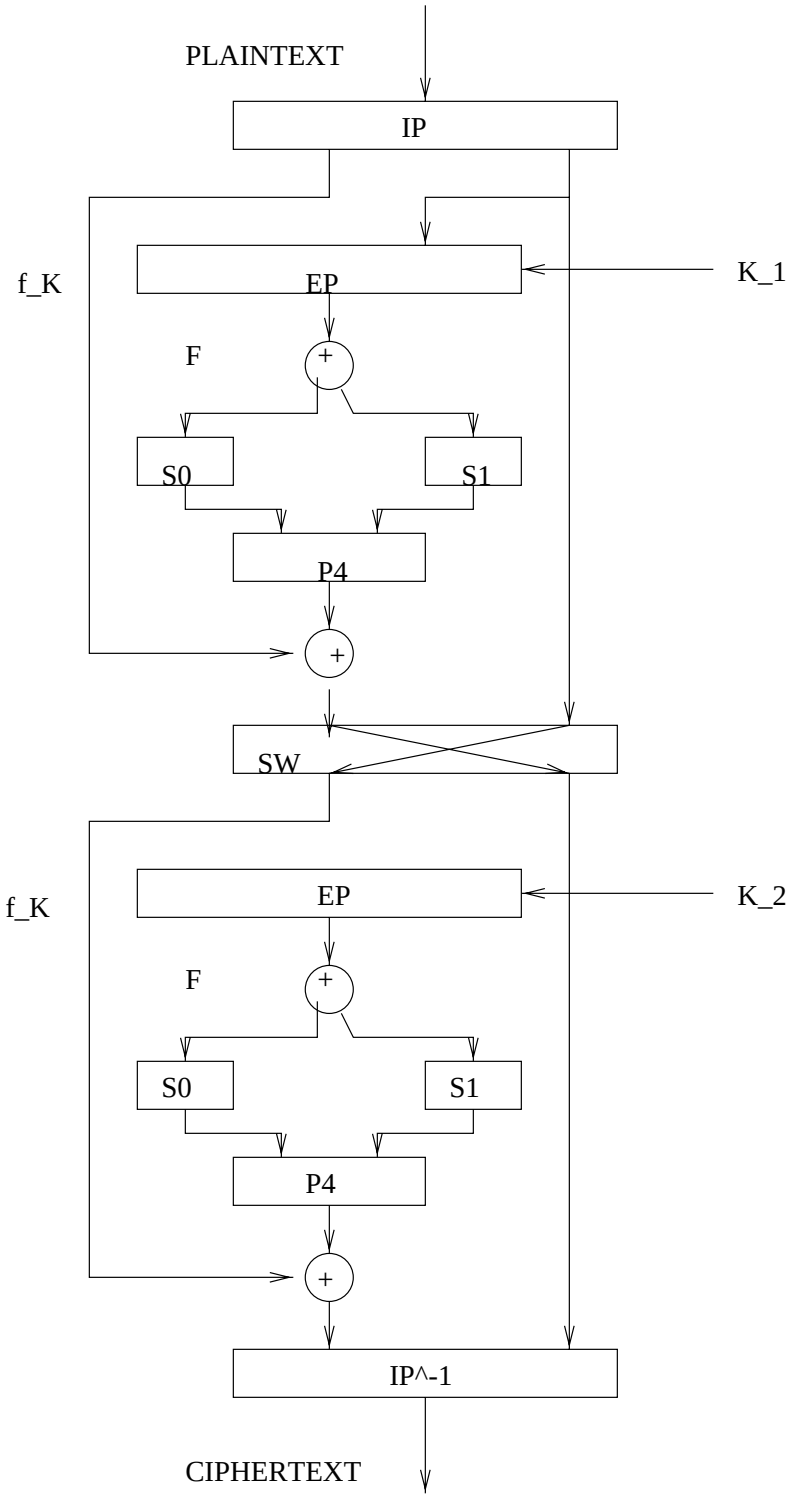
$$\begin{matrix} p_{00} & p_{01} & p_{02} & p_{03} \\ p_{10} & p_{11} & p_{12} & p_{13} \end{matrix}$$

These are now fed into the S-boxes: the top row into $S0$ and the bottom row into $S1$.

5b. S-Boxes: The first four bits (first row) are now fed into S-box $S0$ to produce a 2-bit output and the remaining four bits (second row) are fed into $S1$ to produce another 2-bit output. The S-Boxes operate as follows: the 1st and 4th input bits are treated as a 2-bit number that specifies a row of the S-box and the 2nd and 3rd bits specify a column. The output is now the entry of the S-box in that (row,column). Similarly for S-box $S1$.

Example: Let $p_{00}p_{01}p_{02}p_{03} = 0110$ be the top row. Then $p_{00}p_{03} = 00 = 0$ and $p_{01}p_{02} = 11 = 3$ and the output is from row 0 and column 3 of $S0$, which is 2 ($= 10$ in binary).

Next the four bits produced undergo permutation $P4$ and this output is also the output of F.


PLAINTEXT
IP
f_K
EP
K_1
F
+
S0
S1
P4
+
SW
f_K
EP
K_2
F
+
S0
S1
P4
+
IP^-1
CIPHERTEXT

CRYPTANALYSIS OF SIMPLE DES

Ciphertext-only Attack: Brute-force attack is feasible, since there are only 2^{10} possibilities with 10-bit keys.

Known Plaintext Attack: We can describe the relationship between a single plaintext block

$$p_1p_2p_3p_4p_5p_6p_7p_8$$

and a single ciphertext block

$$c_1c_2c_3c_4c_5c_6c_7c_8$$

in terms of (nonlinear) mathematical equations with unknowns the 10 bits of the key

$$k_1k_2k_3k_4k_5k_6k_7k_8k_9k_{10}.$$

This involves expressions of the previously described encryption.

For example, if

$$p_0p_1p_2p_3$$
$$q_0q_1q_2q_3$$

are the two 4-bit output rows from the expansion operator in (5a) then we get a 4-bit output $xyzw$ from the two S-boxes.

If xy is the 2-bit output of $S0$ then the following equations are true

$$\begin{aligned} x &= p_0p_1p_2p_3 + p_0p_1 + p_0p_2 + p_3 \\ y &= p_0p_1p_2p_3 + p_0p_1p_3 + p_0p_1 + \\ &\quad p_0p_2 + p_0p_3 + p_0 + p_2 + 1 \end{aligned}$$

where additions are modulo 2.

A similar pair of equations is derived for zw from the S-box $S1$.

On the surface, this is not an efficient cryptanalysis because it involves too many equations and unknowns.

FEISTEL CIPHERS

Feistel ciphers are based on designs of block ciphers that maximize the effect of Shannon's "Confusion" and "Diffusion".

In order for the encryption to be reversible (i.e. decryption) blocks generated must be unique. This means the transformation is $1-1$.

The block size cannot be too small because the resulting cipher may not be sufficiently complicated. At the same time a large permutation of a block is not practical. Feistel proposed an approximation to the ideal block cipher system, for large block size, out of smaller easily constructed components.

Input to the cipher is a plaintext of length $2n$ and a key K.

The plaintext block is divided into two parts: L_0 (left) and R_0 (right).

The two halves pass through r rounds of processing and then combine to produce the ciphertext block.

The input (L_{i-1}, R_{i-1}) to the i-th round is obtained from the output of the $(i-1)$-round as well as a subkey. Subkeys are different from K and from each other.

Each round is parametrized by the round subkey and has the same structure:

A substitution is performed on the left half by applying a "round" function F to the right half of the data. Following substitution a permutation is performed that consists of the interchange of the two halves of the data.

DESIGN PARAMETERS

Block & Key Size: The larger the block (respectively, key) size the greater the security provided and the smaller the e(de-)encryption speed. 64 bits is the currently accepted block size, while 128 bits is the currently accepted key size.

Number of Rounds: A typical size is 16 rounds.

Subkey Generation & Round Function: The greater the complexity of the algorithm the greater the difficulty of cryptanalysis.

Encryption/Decryption Speed: This is a major concern in applications.

Ease of Analysis: The algorithm should be easy to analyze in order to understand its weaknesses and increase user confidence.

FEISTEL ALGORITHM

Encryption:
$$\begin{aligned} LE_{i+1} &= RE_i \\ RE_{i+1} &= LE_i \oplus F(RE_i, K_{i+1}), \\ & \quad i = 0, 1, \ldots, 15. \end{aligned}$$

Decryption:
$$\begin{aligned} LD_{i+1} &= RD_i \\ RD_{i+1} &= LD_i \oplus F(RD_i, K_{16-i-1}), \\ & \quad i = 0, 1, \ldots, 15. \end{aligned}$$

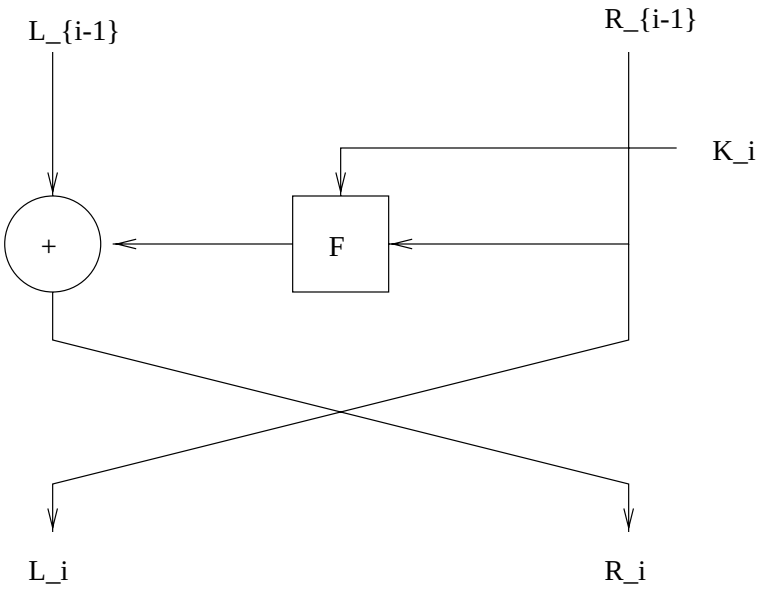
We can show by induction that $LD_i = RE_{16-i}$, and $RD_i = LE_{16-i}$. This is true for the initial step $i = 0$. Assume it is true for i

$$\begin{aligned} LD_{i+1} &= RD_i \\ &= LE_{16-i} \\ &= RE_{16-i-1} \end{aligned}$$

$$\begin{aligned} RD_{i+1} &= LD_i \oplus F(RD_i, K_{16-i-1}) \\ &= RE_{16-i} \oplus F(LD_{i+1}, K_{16-i-1}) \\ &= RE_{16-i} \oplus F(RE_{16-i-1}, K_{16-i-1}) \\ &= LE_{16-i-1} \oplus F(RE_{16-i-1}, K_{16-i}) \\ &\qquad \oplus F(RE_{16-i-1}, K_{16-i-1}) \\ &= LE_{16-i-1} \end{aligned}$$

Hence, decryption is the inverse of encryption.

FEISTEL ROUND



FEISTEL ROUND_i

This is iterated for 16 steps. The reverse is used for decryption.

In practical block ciphers confusion and diffusion is amplified on some rounds with the application of additional substitutions.

FEISTEL TYPE ALGORITHMS

There are several algorithms differing in Block- and Key-size used as well as the number of Rounds.

	$Block\ Size$	$Key\ Size$	$\#Rounds$
DES	64	56	16
$Double-DES$	64	112	32
$Triple-DES$	64	168	48
$IDEA$	64	128	8
$Blowfish$	64	32..448	16
$RC5$	$32, 64, 128$	$0..2,040$	vbl
$CAST-128$	64	40..128	16
$RC2$	64	$8..1,024$	16

DESIGN GUIDELINES

The National Bureau of Standards (NBS) suggested the following guidelines in May 15, 1973:

1. High level of security
2. Complete specification and easy to understand
3. Security must be based on the key, not on the secrecy of the algorithm
4. System available to all users
5. Easily adaptable for diverse applications
6. Economical implementation in electronic devices
7. Algorithm efficient to use
8. Algorithm must be easy to validate
9. Algorithm must be exportable

These principles were meant to enhance public confidence and widespread use of the cryptosystem.

DATA ENCRYPTION STANDARD

DES was adapted as a standard in Jan. 1977, and is the most widely used cryptosystem, especially in financial transactions, PIN code generation, etc.

First published in the Federal Register of March 17, 1975.

Developed by IBM, it is a modification of an older system known as **LUCIFER**.

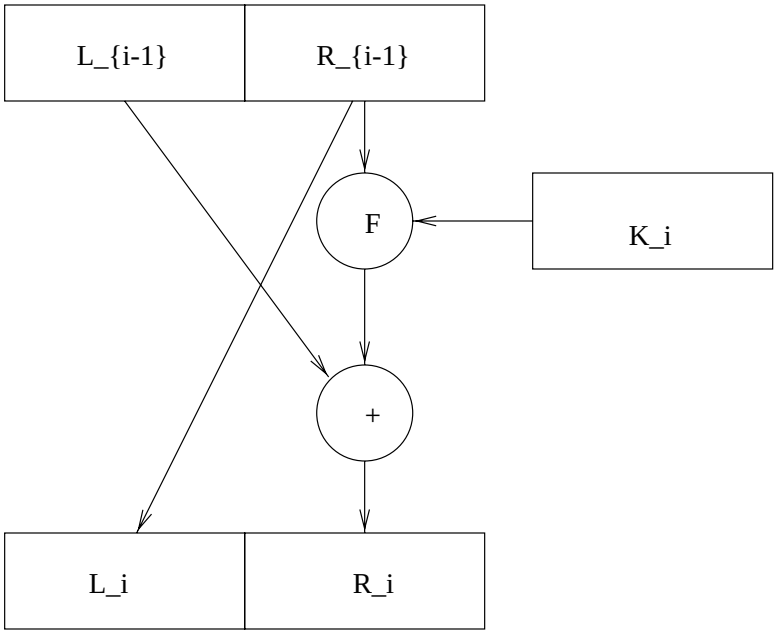
Its most recent renewal was Jan. 1994, but it will not be renewed again.

It is not considered "secure" for future transactions. A recent call of proposals is expected to lead to a successor of DES.

DES ALGORITHM

The DES algorithm is in three steps.

1. Given a plaintext x of 64 bits we compute $IP(x) = x_0 = L_0R_0$, where L_0 is the left half and R_0 the right half of x_0.



2. 16 iterations of a certain function are then computed: $L_i = R_{i-1}, R_i = L_{i-1} \oplus F(R_{i-1}, K_i)$

3. Compute $IP^{-1}(R_{16}L_{16})$. This is the ciphertext block.

FUNCTION $F : \{0,1\}^{32} \times \{0,1\}^{48} \to \{0,1\}^{32}$

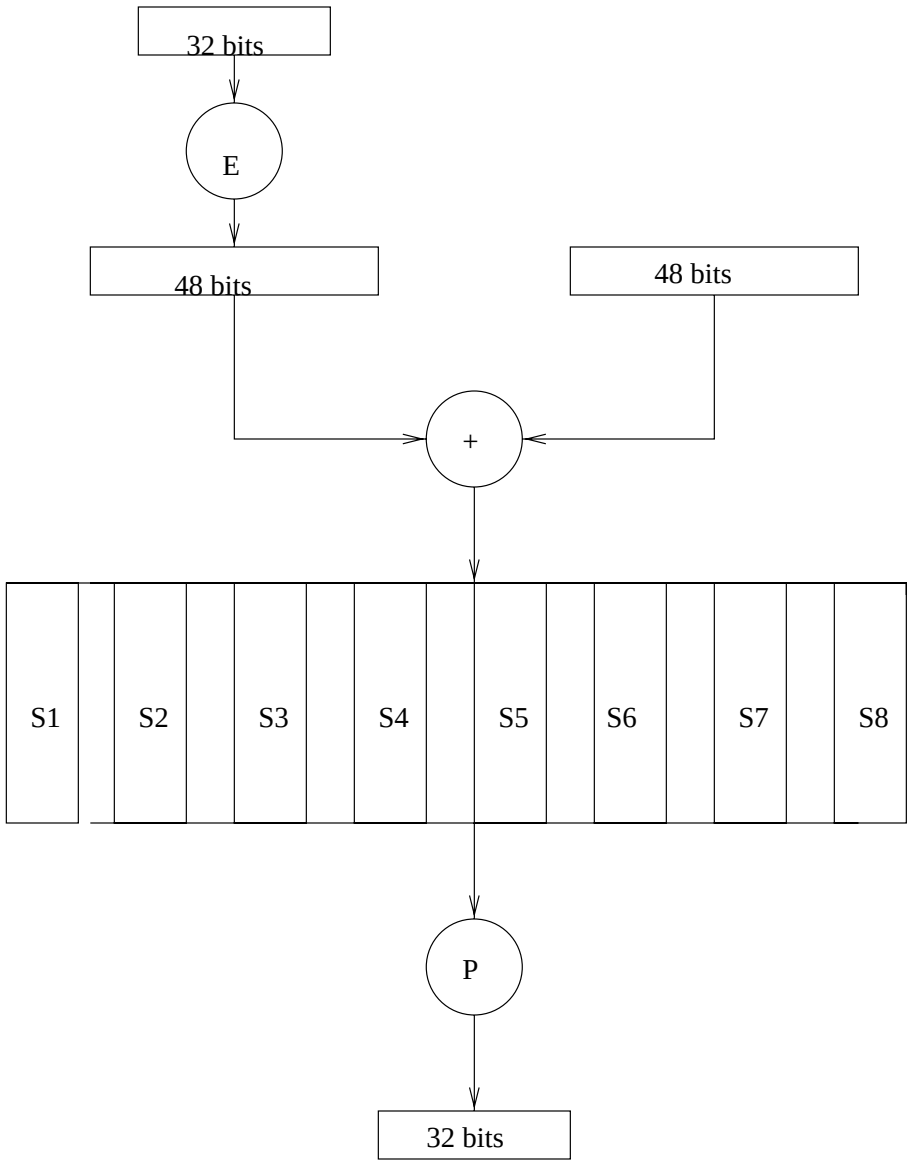
1. The first argument of F, say A, is expanded according to a function $E : \{0,1\}^{32} \to \{0,1\}^{48}$. $E(A)$ is a permutation of A with 16 of the bits repeated twice.

2. The other argument, say J, of F is 48 bits long. We compute $E(A) \oplus J$ and write the result as eight six-bit strings $B = B_1 B_2 \cdots B_8$.

3. There are 8 S-boxes, $S_1, \ldots, S_8$ which are 4×16 arrays with entries from 0 to 15 and can be thought of as functions $S_j : \{0,1\}^2 \times \{0,1\}^4 \to \{0,1\}^4$. Given $B_j = b_1 b_2 \cdots b_6$ we compute $S_j(B_j)$ as follows: $b_1 b_6$ are the binary representation of a row and $b_2 b_3 b_4 b_5$ of a column of S_j. $C_j := S_j(B_j)$ is the entry of S_j written in binary.

4. $C = C_1 C_2 \cdots C_8$ (32 bits long) is permuted according to a permutation P and we define $F(A, J) = P(C)$.

The function F used in the DES algorithm is based on the following figure.



BASIC FUNCTIONS

IP is the initial permutation:

$$IP(x_1, x_2, \ldots, x_{64}) = (x_{58}, x_{50}, \ldots, x_7) :$$

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

and $\mathbf{IP}^{-1}$ is its inverse. The expansion **E** and permutation **P** functions are

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	16	19	20	21
20	21	22	23	24	25
24	25	26	27	28	29
28	29	30	31	32	1

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

S-box 1:

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

S-box 2:

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

S-box 3:

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

S-box 4:

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

S-box 5:

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

S-box 6:

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

S-box 7:

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

S-box 8:

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

The eight S-boxes $S_1, \ldots, S_8$.

COMPUTATION OF KEY SCHEDULE

The key K is a bitstring of length 64: 56 bits are used for the key and 8 for parity check. Bits in positions $8, 16, \ldots, 64$ are defined so that the number of 1s in each byte is odd. The parity check bits are ignored in the computation.

1. Given K discard the parity check bits and permute the remaining according to permutation $PC-1$: $PC-1(K) = C_0D_0$, where C_0, D_0 are the two 28-bit long halves of K.

2. For $i = 1..16$, $C_i = LeftShift_i(C_{i-1}), D_i = LeftShift_i(D_{i-1})$ and $K_i = PC{-}2(C_iD_i)$. Here, $LeftShift_i$ is a left-shift one position if $i = 1, 2, 9, 16$, and two positions, otherwise. Also $PC-2$ is a fixed permutation. K_i has 48 bits.

Decryption is done by using the key schedule in reverse order: $K_{16}, \ldots, K_1$.

PERMUTATIONS PC-1 and PC-2

$PC-1:$

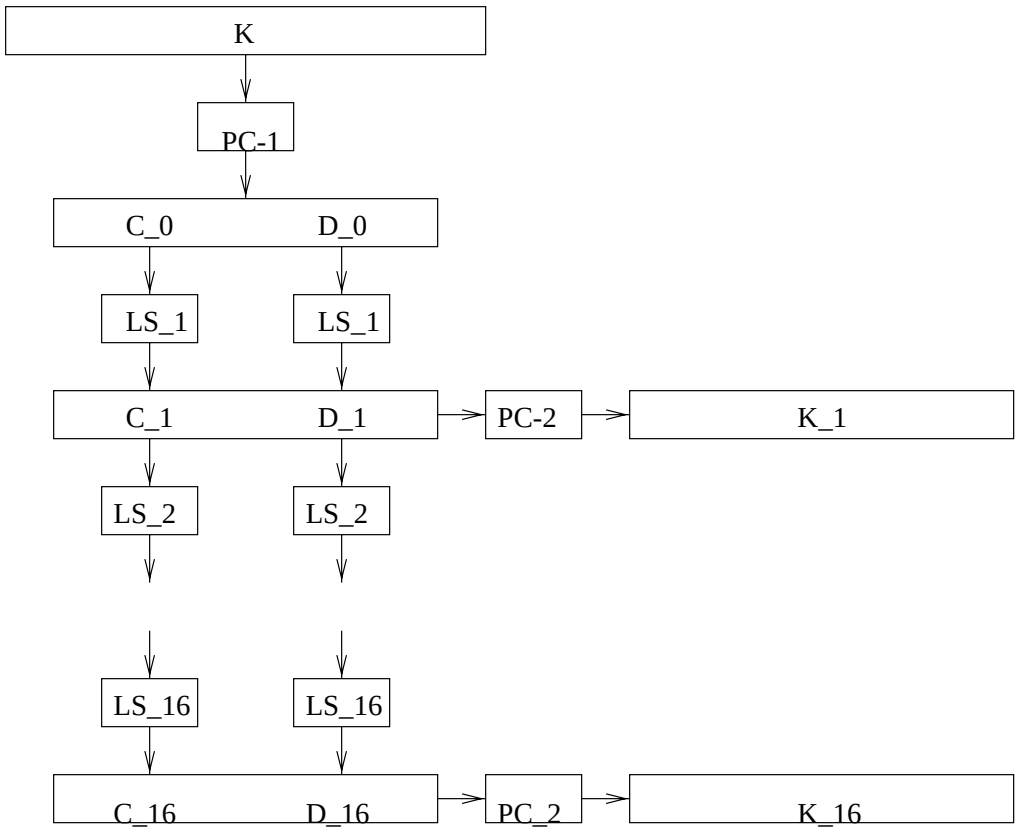
57	49	41	33	25	17	9
1	58	50	42	34	26	18
10	2	59	51	43	35	27
19	11	3	60	52	44	36
63	55	47	39	31	23	15
7	62	54	46	38	30	22
14	6	61	53	45	37	29
21	13	5	28	20	12	4

$PC-2:$

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

Note: $PC-2$ generates 48 bits.

Computation of Key schedules



There are 16 rounds to the key computation.

DES DESIGN PRINCIPLES

Do the S-boxes contain hidden trapdoors to allow NSA to decrypt easily? NSA asserted the following properties in 1976:

P0: All the rows of all the S-boxes are permutations of $0, 1, \ldots, 15$.

P1: S-boxes are not affine transformations of their input.

P2: Change in an input bit, changes at least two output bits of the S-box.

P3: For any x and any S-box S, $S(x), S(x \oplus 001100)$ differ by at least two bits.

P4: For any string x, bits b, b' and S-box S, $S(x) \neq S(x \oplus 11bb'00)$.

P5: For any S-box, and any fixed input bit the number of inputs for which a fixed output bit has the value 0 (or 1) is always between 13 and 19.

BREAKING DES

No further properties have been acknowledged.

There is a lot of controversy regarding DES security. Is a keyspace of size 2^{56} large enough?

Diffie and Hellman, as early as 1977, proposed the construction of special purpose machines for breaking DES, at a cost of $ 20 million.

In 1993, Mike Wiener (of Entrust, an Ottawa based software firm) proposed a detailed design of a machine based on a key search chip which is pipelined so that all 16 encryptions take place simultaneously.

In Jan. 29, 1997, RSA-Labs issued a challenge (with a ten thousand dollar reward) to find a DES key for a plaintext message preceded by three known blocks containing the phrase "the unknown message is". A project began Feb. 18, 1997, involving 70,000 systems worldwide. It ended 96 days later with the correct key!

DES has strong "diffusion" behavior. Small change in plaintext or key causes significant change in ciphertext (avalanche effect). As a test, two plaintexts that differ on only one bit

$$\begin{array}{cccccccc} 0^8 & 0^8 & 0^8 & 0^8 & 0^8 & 0^8 & 0^8 & 0^8 \\ 10^7 & 0^8 & 0^8 & 0^8 & 0^8 & 0^8 & 0^8 & 0^8 \end{array}$$

and a key

$$\begin{array}{cccc} 0^6 1 & 10^2 10 1^2 & 010^2 10^2 & 1^2 0^3 10 \\ 0^2 1^3 0^2 & 0^2 1^2 0^3 & 0^2 1^3 0^2 & 01^2 0^2 10 \end{array}$$

were used and generated blocks that differ as follows:

Round #	**# of Bits that differ**
0	1
4	39
8	29
12	30
16	34

MODES OF OPERATION

DES is used in banking, government and private industry.

Implementations are either in Software or Hardware (specially designed chips).

Four modes of operation have been developed in order to satisfy a variety of requirements. On input string $x_1, x_2, \ldots$ of blocks the output is $y_1, y_2, \ldots$.

ECB (Electronic CodeBook): Same key K is used throughout. Since only one key is used it is less secure, but it is useful for the transmission of small amounts of data, e.g., transmission of encrypted keys.

CFB (Cipher FeedBack): Start with initial vector $y_0 = IV$ and define $y_i = E_K(y_{i-1} \oplus x_i)$. So the ciphertext is used in the encryption like a stream cipher.

CBC (Cipher Block Chaining): A key stream is generated from initial value $z_0 = IV$ and rule $z_i = E_K(z_{i-1})$. The ciphertext is $y_i = z_i \oplus x_i$.

OFB (Output FeedBack): Set $y_0 = IV, z_i = E_K(y_{i-1})$, and $y_i = z_i \oplus x_i$.

There are also k-FeedBack modes for CBC and OFB.

OFB is used frequently in satellite transmissions.

CBC and CFB are useful for Message Authentication Codes (MACs) appended to the end of the message.

CHOSEN PLAINTEXT ATTACK

Given x, y (plaintext, ciphertext pair) such that $y = E_K(x)$ we are interested in computing K.

With Exhaustive Search: Try all 2^{56} keys by exhaustive search. This requires zero memory but on average 2^{56} keys will be tried before we succeed.

With Large Memory: Given plaintext x tabulate (y_K, K), $y_K = E_K(x)$ for all 2^{56} keys K, sorted lexicographically. Later, given a ciphertext y which encrypts x, compute key with table lookup. Here time is constant but memory requirement is large. This approach has advantage only when several keys are to be found.

A Time/Memory Tradeoff: Since the block size of plain-/cipher-text is 56 bits but key-size is 64 bits we need a reduction function $R : \{0,1\}^{64} \rightarrow \{0,1\}^{56}$.

Given string x define the function $\{0,1\}^{56} \rightarrow \{0,1\}^{56} : K \rightarrow g_x(K) := R(E_K(x))$. Let $m.t$ be parameters, which are chosen to satisfy $m \approx t \approx N^{1/3}$, where $N = 2^{56}$.

Construct an $m \times t$ matrix of bitstrings as follows: 1. The first column is chosen at random, i.e., choose $K_{i,0}, i = 1, \ldots, m$, at random.
2. Define recursively $K_{i,j} = g_x(K_{i,j-1})$, for $t \geq j \geq 1$.
3. Construct a table of ordered pairs (first and last columns) $T = (K_{i,t}, K_{i,0})$ sorted by their first coordinate.

Rather than search the whole $m \times t$ matrix, search for the key K by looking only at the table T (which of course has only $2m$ entries). To do this we need an algorithm!

K may not occur in the $m \times t$ matrix. However, if it did we could argue as follows.

$K_{1,0}$	$\rightarrow^{g_x}$	$K_{1,1}$	$\rightarrow^{g_x} \ldots$	$\rightarrow^{g_x}$	$K_{1,t}$
$K_{2,0}$	$\rightarrow^{g_x}$	$K_{2,1}$	$\rightarrow^{g_x} \ldots$	$\rightarrow^{g_x}$	$K_{2,t}$
	$\ldots$		$\ldots$		
$K_{m,0}$	$\rightarrow^{g_x}$	$K_{m,1}$	$\rightarrow^{g_x} \ldots$	$\rightarrow^{g_x}$	$K_{m,t}$

Assume $K = K_{i,t-j}$, for some $i \leq m, j \leq t$.

$$\begin{aligned} K_{i,t} &= g_x^j(K) \\ &= g_x^{j-1}(g_x(K)) \\ &= g_x^{j-1}(R(E_K(x))) \\ &= g_x^{j-1}(R(y)) \end{aligned}$$

Let y_j, $1 \leq j \leq t$, be computed from

$$y_j = \begin{cases} R(y) & \text{if } j = 1 \\ g_x(y_{j-1}) & \text{if } 2 \leq j \leq t \end{cases}$$

Thus, if $K = K_{i,t-j}$ then $y_j = K_{i,t}$ (the reverse may not be true).

Since, $R : \{0,1\}^{64} \rightarrow \{0,1\}^{56}$, on the average any 56-bit string has $2^8 = 256$ preimages. We need to check whether or not $E_{K_{i,t-j}}(x) = y$ to see if indeed $K_{i,t-j}$ is the key K. $K_{i,t-j}$ was not stored, but is easily computed from $K_{i,0}$.

Choose $m \approx t \approx (2^{56})^{1/3}$ and execute the following

Algorithm :
1. *compute* $y_1 = R(y)$
2. **for** $j = 1$ **to** t **do**
   if $y_j = K_{i,t}$ *for some* i **then**
3. *compute* $K_{i,t-j}$ *from* $K_{i,0}$ *by iterating* g_x
4. **if** $y = E_{K_{i,t-j}}(x)$ **then**
5. *put* $K = K_{i,t-j}$ *and* **quit**
6. *compute* $y_{j+1} = g_x(y_j)$

It is easily seen that if $N = mt^2 \approx 2^{56}$ then

$$\Pr[\exists i, j(K = K_{i,t-j})] \approx (0.8)mt/N = (0.8)N^{-1/3}.$$

Construct $N^{1/3}$ tables using $N^{1/3}$ different reduction functions. Each table has two columns and each column has $56 \cdot N^{1/3}$ bits, which gives a total storage requirement of $112 \cdot N^{2/3}$ bits. Precomputation time is $O(N)$.

With binary search step 3 takes $O(\log m)$ time units. If step 3 is never successful it takes $N^{2/3}$ time units.

MULTIPLE DES (USED IN INTERNET)

Double DES requires two keys K_1, K_2 applied to a plaintext:

$$x \rightarrow E_{K_1}(x) \rightarrow E_{K_2}(E_{K_1}(x))$$

Triple DES requires three keys K_1, K_2, K_3 applied to a plaintext:

$$x \rightarrow E_{K_1}(x) \rightarrow E_{K_2}(E_{K_1}(x)) \rightarrow E_{K_3}(E_{K_2}(E_{K_1}(x)))$$

Triple DES with two keys, requires two keys K_1, K_2 applied to a plaintext:

$$x \rightarrow E_{K_1}(x) \rightarrow D_{K_2}(E_{K_1}(x)) \rightarrow E_{K_1}(D_{K_2}(E_{K_1}(x)))$$

Is double DES reducible to DES? I.e., given K_1, K_2 does there exist K such that $E_K(x) = E_{K_2}(E_{K_1}(x))$? **No!** It can be shown that the set of DES encryption functions under composition is not a group! (see CRYPTO'92).

MEET IN THE MIDDLE ATTACK

The idea of meet-in-the-middle attack is due to (Diffie-Hellman, 1977):
If $y = E_{K_2}(E_{K_1}(x))$ then $E_{K_1}(x) = D_{K_2}(y)$
Procedure Plaintext Attack on (x, y)**:**
1. Encrypt x with 2^{56} keys.
2. Decrypt y with 2^{56} keys.
3. Tabulate results and check for matching pairs. Check entries of one table against entries of another. If a match occurs on a pair (K_1, K_2), check whether this pair works on a **new** plaintext/ciphertext pair. If not discard!
For the given plaintext x, there are 2^{64} possible ciphertexts that could be produced by double DES using 2^{112} possible keys. On average, $2^{112}/2^{64} = 2^{48}$ keys will produce the given ciphertext y. This procedure will produce about 2^{48} "false alarms" on the given pair (x, y).

With additional 64 bits of plaintext/ciphertext pairs the false alarm rate is reduced to $2^{48}/2^{64} = 2^{-16}$, i.e. the probability the correct keys are determined is $1 - 2^{-16}$.

LINEAR CRYPTANALYSIS

Linear cryptanalysis for DES was first described by Matsui in Eurocrypt 1993. This is based on the idea of finding a linear approximation to describe the DES transformation.

1. Consider a Cipher with n-bit plaintext and ciphertext blocks and m-bit keys. Let $x = x_1 \cdots x_n$ be the plaintext, $y = y_1 \cdots y_n$ the ciphertext, and $K = K_1 \cdots K_m$ the key.

2. Find fixed locations $\alpha_1 < \alpha_2 < \cdots < \alpha_a \leq n, \beta_1 < \beta_2 < \cdots < \beta_b \leq n, \gamma_1 < \gamma_2 < \cdots < \gamma_c \leq m$ such that

$$(x_{\alpha_1} \oplus \cdots \oplus x_{\alpha_a}) \oplus (y_{\beta_1} \oplus \cdots \oplus y_{\beta_b}) = K_{\gamma_1} \oplus \cdots \oplus K_{\gamma_c}$$

holds with probability $\neq 1/2$ (the further from 1/2 the more effective the equation).

3. Compute results of LHS for a large number of pairs (x, y). If result is 0 more than half the time then guess RHS is 0 (Same for 1). This gives a linear equation on key bits. Now try to get more such equations. DES key can be found given 2^{47} known plaintexts.

DIFFERENTIAL CRYPTANALYSIS

First proposed by Murphy, 1990, for "the Cryptanalysis of FEAL-4 with 20 chosen plaintexts" and further elaborated by Biham and Shamir in a series of papers. **The idea is this:**

Consider the initial plaintext x as two 32-bit halves x_0, x_1. Each round of DES produces only one new 32-bit half. If x_i is the new 32-bit block then $x_{i+1} = x_{i-1} \oplus F(x_i, K_i)$.

Start with two known 64-bit messages x, x^* with known XOR difference $\Delta x = x \oplus x^*$ and consider the difference of intermediate halves

$$\begin{aligned} \Delta x_{i+1} &= x_{i+1} \oplus x^*_{i+1} \\ &= (x_{i-1} \oplus F(x_i, K_i)) \oplus x^*_{i-1} \oplus F(x^*_i, K_i) \\ &= \Delta x_{i-1} \oplus (F(x_i, K_i) \oplus F(x^*_i, K_i)) \end{aligned}$$

Assume that many pairs (x_i, x_i^*) with a given Δx_i yield the same output difference if the same subkey K_i is used.

I.e, $F(x_i, K_i) \oplus F(x_i^*, K_i)$ is a "function" of Δx_i, with high probability.

Therefore if we know Δx_{i-1} and Δx_i then we know Δx_{i+1}, with high probability.

If a number of such differences is determined then it is possible to determine the subkey used.

Outline of Differential Cryptanalysis:

1. Begin with two plaintext messages x, x^* with a given difference and trace through a probable pattern of differences after each round to yield a difference for the ciphertext.

2. Next submit x, x^* for encryption to determine the actual differences under the unknown key.

3. If there is a match between the two values we suspect all intermediate rounds are correct.

4. Repeat many times to determine all key bits.

This method can be used to "break" DES with a small number (up to eight) of rounds.

IDEA

The International Data Encryption Algorithm is based on sound theoretical foundations and is considered one of the most secure block ciphers available today. It is also used in e-mail security program PGP.

Plaintext blocks are 64 bits, Keys are 128 bits, and the same algorithm is being used for encryption and decryption.

Diffusion and confusion is created by three algebraic groups: (1) XOR, (2) Addition mod $(2^{16}+1)$, and (3) Multiplication mod $(2^{16}+1)$ (this last one is also viewed as an S-box).

All operations "operate" on 16-bit blocks. Data blocks are divided into four sixteen bit blocks X_1, X_2, X_3, X_4. There are 8 rounds, and each round has 14 steps.

IDEA Rounds: $X_1X_2X_3X_4 \rightarrow Z_1Z_2Z_3Z_4$.

1. $(X_1 \cdot K_1) \bmod (2^{16}+1)$
2. $(X_2 + K_2) \bmod (2^{16}+1)$
3. $(X_3 + K_3) \bmod (2^{16}+1)$
4. $(X_4 \cdot K_4) \bmod (2^{16}+1)$
5. XOR$(1,3)$
6. XOR$(2,4)$
7. Multiply$(5, K_5)$
8. Add$(6,7)$
9. Multiply$(8, K_6)$
10. Add$(7,9)$
11. $Y_1 :=$ XOR$(1,9)$
12. $Y_2 :=$ XOR$(3,9)$
13. $Y_3 :=$ XOR$(2,10)$
14. $Y_4 :=$ XOR$(4,10)$

Except for the eighth (last) round, swap Y_2 and Y_3, and this is the output of the round. Let the output be $Y_1Y_2Y_3Y_4$. After the eighth round also perform the **Output** transformation:

1. $Z_1 := (Y_1 \cdot K_1) \bmod (2^{16}+1)$
2. $Z_2 := (Y_2 + K_2) \bmod (2^{16}+1)$
3. $Z_3 := (Y_3 + K_3) \bmod (2^{16}+1)$
4. $Z_4 := (Y_4 \cdot K_4) \bmod (2^{16}+1)$

Subkey Generation: Algorithm uses a total of 52 subkeys: six in each of the eight rounds and four in the last output round.
Original key K is divided into eight 16-bit subkeys. First six are used in Round1 and remaining two in Round2.
Rotate key 28 bits to the left. Subdivide into eight subkeys. The previous two subkeys and the first four subkeys are used for Round2. The remaining four subkeys are used for Round3.
Rotate key 28 bits to the left and so on until the end of the algorithm.

Decryption: The first four decryption subkeys DK of decryption Round$i$ are derived from the first four encryption subkeys EK of encryption Round$(10-i)$ according to the following rule:

$$DK_1 = EK_1^{-1} \quad DK_4 = EK_4^{-1}$$
$$DK_2 = -EK_3 \quad DK_3 = -EK_2 \text{ (Rounds 2..8)}$$
$$DK_2 = -EK_2 \quad DK_3 = -EK_3 \text{ (Rounds 1 \& 9)}$$

For the first eight rounds, the last two subkeys of decryption Round$i$ are equal to the last two subkeys of encryption Round$(9-i)$.

Speed: Current software implementations of IDEA run twice as fast as DES.

Cryptanalysis: Brute force attacks are impossible because the search space has size 2^{128}.

Few theoretical studies have been done, but by design it seems to be immune to differential cryptanalysis.

A class of weak keys has been discovered such that if used an attacker can easily identify them in a chosen plaintext attack. However these are "specially" constructed and it is unlikely they will ever be used.

IDEA has several variants and modes of operation.

Many open questions remain: Is IDEA a group? Can the cipher be broken?

BLOWFISH ALGORITHM

A 64-bit, variable key, Feistel-type cipher.
Characteristics:
1. **Speed:** encrypts data on 32-bit microprocessors at the rate of 16 c;clock cycles per byte.
2. **Compactness:** Can run in less than $5K$ of memory.
3. **Simplicity:** Easy to implement.
4. **Variability:** Flexible key size as long as 448 bits to enhance security.

Key Expansion: Converts key of up to 448 bits into several subkey arrays totaling 4168 bytes. Keys stored in a P-array of 18 32-bit subkeys $P_1, \ldots, P_{18}$. They are generated with the **Blowfish** algorithm.

Data Encryption: A single function is iterated 16 times; each round uses a key- and data-dependent permutation. Two types of operations are performed: XORs and additions on 32-bit words.

S-boxes: Four 32-bit S-boxes with 256 bits each: $S_{i,0}, S_{i.1}, \ldots, S_{i,255}, \quad i = 1, 2, 3, 4.$

Function $F : \{0,1\}^{32} \to \{0,1\}^{32}$. Divide input u into four parts a, b, c, d and define $F(u)$ by

$$((S_{1,a} + S_{2,b}) \bmod 2^{32} \oplus S_{3,c}) + S_{4,d} \bmod 2^{32}$$

Encryption $x \in \{0,1\}^{64}$:

Divide x into two halves $x = x_L x_R$
for $i = 1$ **to** 16
    $x_L = x_L \oplus P_i$
    $x_R = F(x_L) \oplus x_R$
    swap x_L and x_R
swap x_L and x_R (i.e., undo last swap)
    $x_R = x_R \oplus P_{17}$
    $x_L = x_L \oplus P_{18}$
Output: $x_L x_R$

Decryption: Same as Encryption except for reversing $P_1, \ldots, P_{18}$.

No successful cryptanalysis known. Certain weak keys have been discovered.

CAST

Variants exist depending on key size: CAST-64, CAST-128, CAST-256. CAST-64 uses six S-boxes with 8-bit input and 32-bit output.

Function $f : \{0,1\}^{32} \to \{0,1\}^{32}$: Divide input x into four 8-bit quarters x_1, x_2, x_3, x_4 and 16-bit subkey into two 8-bit halves ex_5x_6. Process x_i through i-th S-box, for $i = 1, \ldots, 6$. XOR six S-box outputs to form output $f(x)$.

S-boxes & Subkeys: S-boxes are implementation dependent and rather complicated. Let $K_1, \ldots, K_8$ be the eight bytes of the key K. Then the subkeys are:

$$\begin{array}{ll} \text{Round}_i: & K_{2i-1}, K_{2i}, i = 1,2,3,4 \\ \text{Round}_5: & K_4, K_3, \quad \text{Round}_6: K_2, K_1 \\ \text{Round}_7: & K_8, K_7, \quad \text{Round}_8: K_6, K_5 \end{array}$$

Encryption: Divide input into two halves. Algorithm has 8 rounds. In each round the right half is combined with some key using a function f and then XORed with left half to form a new right half.

OTHER BLOCK CIPHERS

MADRYGA

RC5 ALGORITHM

RC2 ALGORITHM

FEAL

REDOC

LOKI

KHUFU and KHAFRE

MMB

SKIPJACK

GHOST

SAFER