I. Data Encryption Standard (DES)

Dikembangkan di IBM pada tahun 1972. Berdasarkan pada algoritma *Lucifer* yang dibuat oleh Horst Feistel. DES adalah standard, sedangkan algoritmanya adalah DEA (*Data EncryptionAlgorithm*). Kedua nama ini sering dikacaukan.

DES adalah sebuah "block cipher", artinya, DES bekerja dalam plaintext dengan ukuran yang telah diberikan (64 bit) dan mengembalikan ciphertext dengan ukuran yang sama pula. Cara kerja dari DES secara umum dapat dilihat dalam gambar berikut :

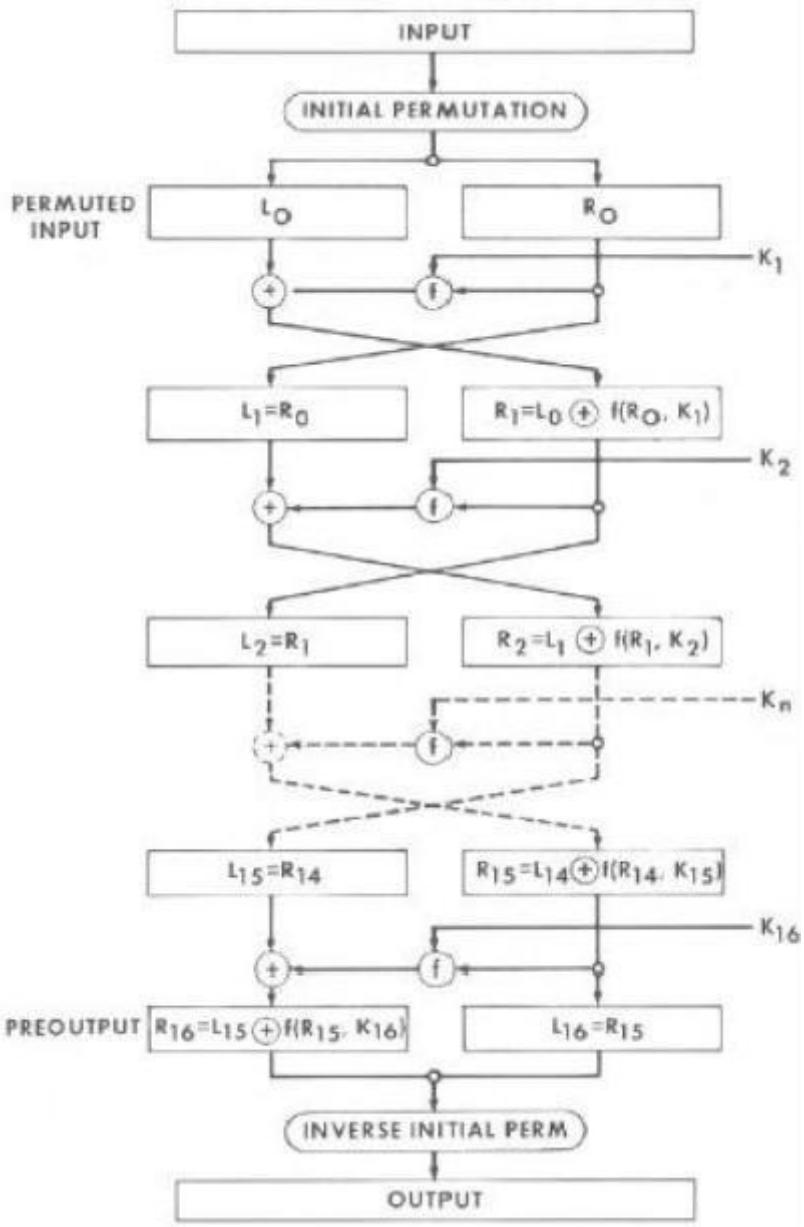
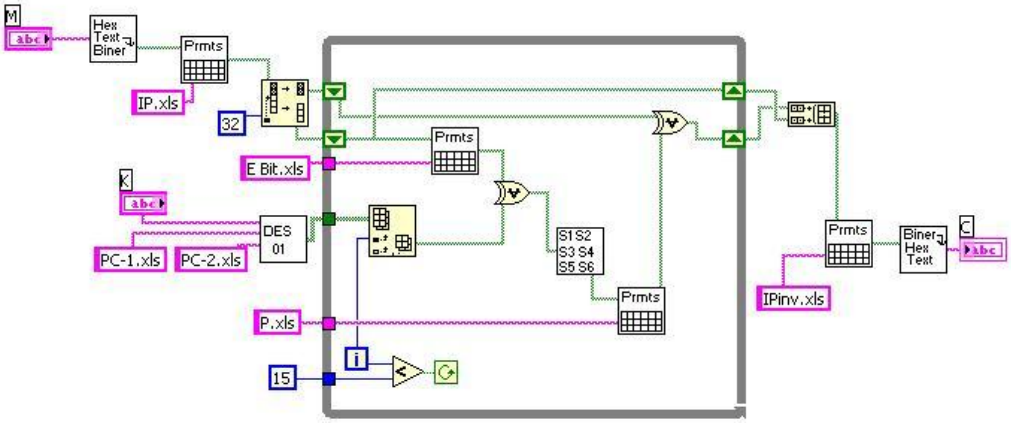
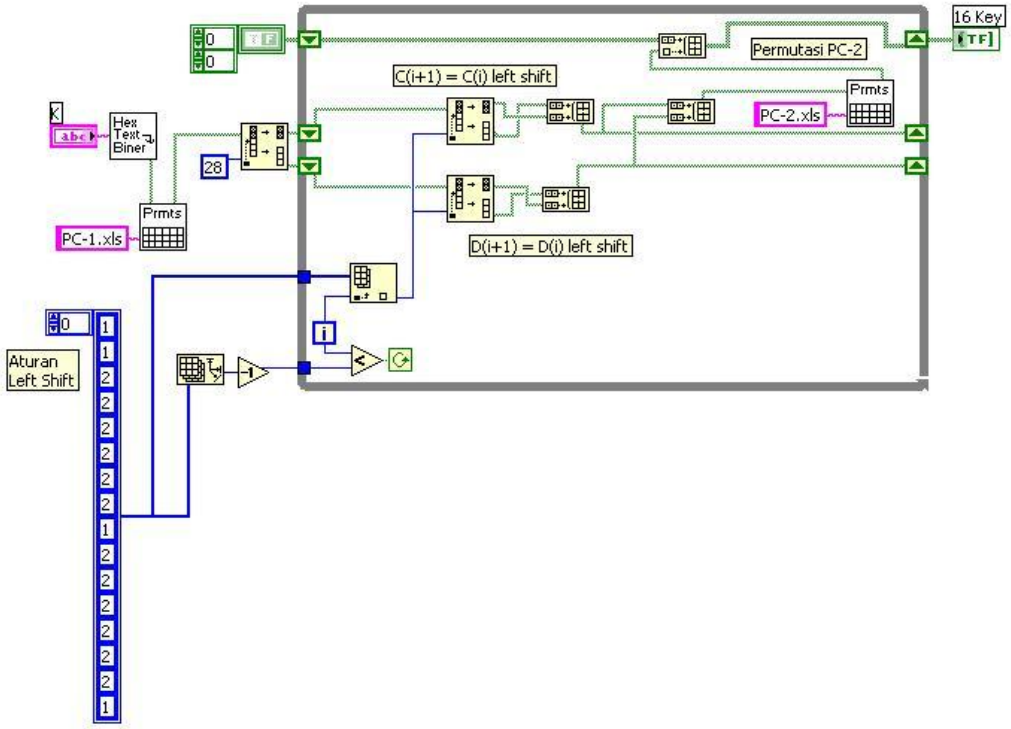


Figure *Enciphering computation.*

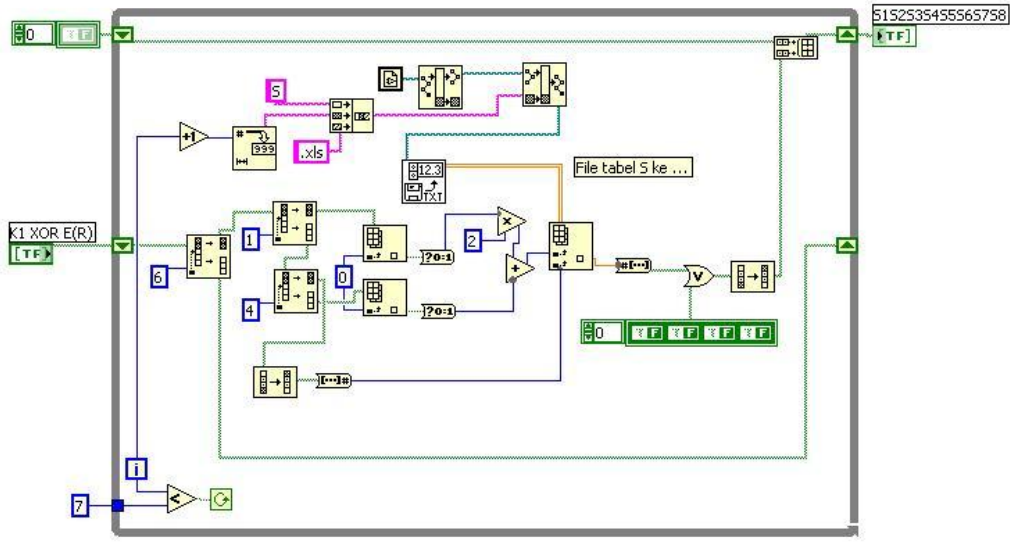
Contoh program LabVIEW DES utama :



Contoh program LabVIEW membangun anak kunci (DES01.vi)



Contoh program LabVIEW kalkulasi f(R,K)



Langkah DES 1 : Membangun 16 anak kunci

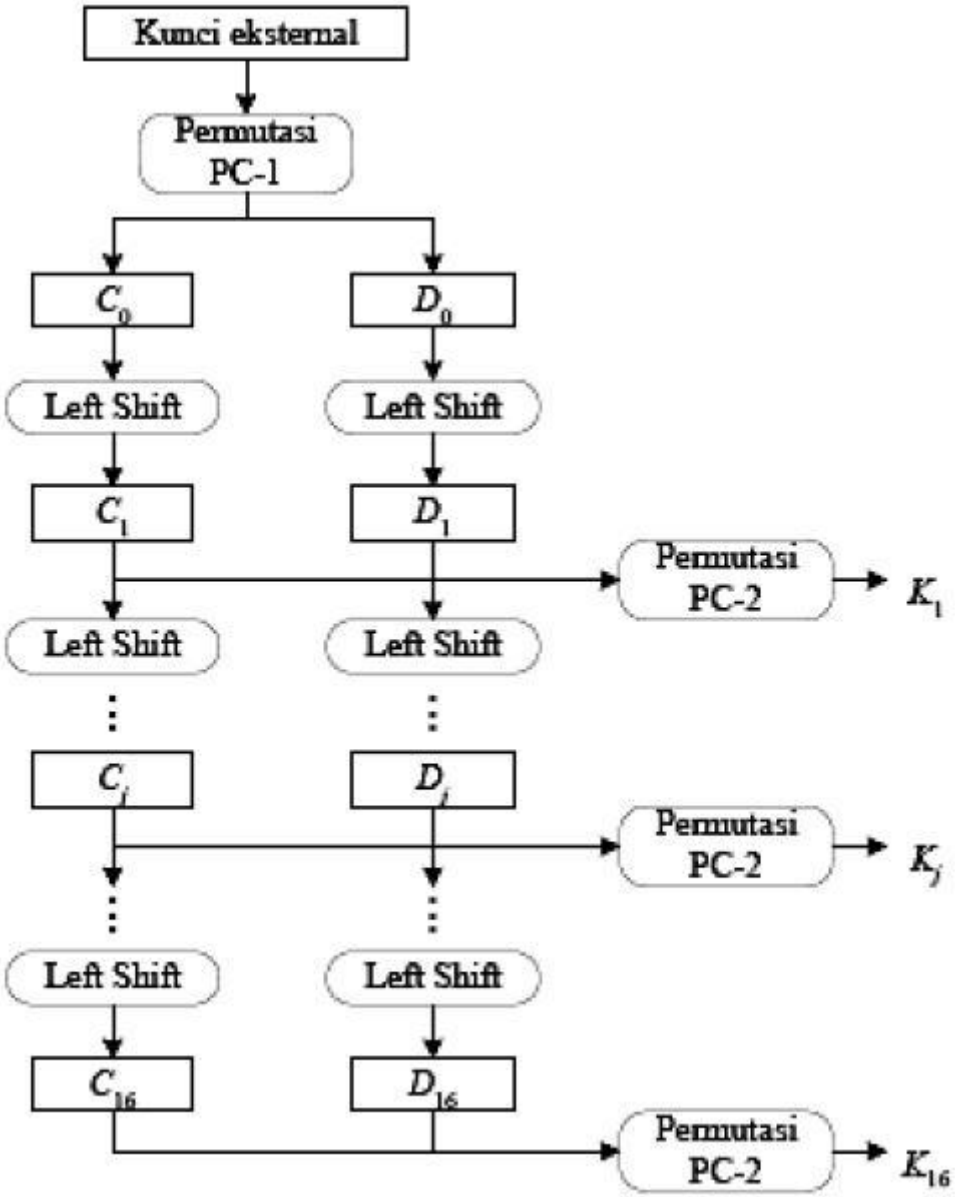


Figure Proses Pembangkitan Kunci-kunci Internal DES

Jika kita mempunyai kunci K = 133457799BBCDFF1, maka kita juga harus menuliskannya di dalam format angka biner sebagai berikut :
K = 00010011 00110100 01010111 01111001 10011011 10111100 11011111 11110001

Ke-64 bit kunci yang telah ditentukan dipermutasikan sesuai dengan tabel PC-1. Elemen pertama dari tabel tersebut adalah 57, artinya, bit ke-57 dari kunci menjadi bit pertama dari kunci yang telah dipermutasikan (K+). Bit ke 49 menjadi bit kedua pada K+.

TABLE 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

Maka dari kunci 64 bit yang asli diatas :
K = 00010011 00110100 01010111 01111001 10011011 10111100 11011111 11110001

Kita mendapat kunci permutasi 56 bit :
K+ = 1111000 0110011 0010101 0101111 0101010 1011001 1001111 0001111
(Kunci berubah dari 64 bit menjadi 56 bit karena ukuran table PC-1 hanya berukuran 7 x 8)

Selanjutnya dua kunci permutasi tersebut dibagi menjadi C_0 dan D_0, dimana setiap bagian mempunyai ukuran 28 bit
C_0 = 1111000 0110011 0010101 0101111
D_0 = 0101010 1011001 1001111 0001111

Dari C_0 dan D_0 tersebut, dibangun C_1 hingga D_{16} dengan aturan "left shift" dari blok sebelumnya. Untuk melakukan metode "left shift", pindahkan setiap bit ke kiri sesuai aturan berikut kecuali bit pertama yang akan berpindah ke bit terakhir.

Iteration number	Number of Left Shifts
1	1
2	1
3	2
4	2
5	2
6	2
7	2
8	2
9	1
10	2
11	2
12	2
13	2
14	2
15	2
16	1

Maka akan diperoleh nilai-nilai sebagai berikut :

C_0 = 11110000110011001010101011111
D_0 = 01010101011001100111100011111

C_1 = 11100001100110010101010111111
D_1 = 10101010110011001111100011110

C_2 = 11000011001100101010101111111
D_2 = 01010101100110011110001111101

C_3 = 00001100110010101010111111111
D_3 = 01010110011001111100011110101

C_4 = 00110011001010101011111111100
D_4 = 01011001100111100011110101101

C_5 = 11001100101010101111111110000
D_5 = 01100110011110001111010101101

C_6 = 00110010101010111111111000011
D_6 = 10011001111000111101010101101

C_7 = 11001010101011111111000011100
D_7 = 01100111100011110101010110110

C_8 = 00101010101111111110000110011
D_8 = 10011110001111010101010110011

C_9 = 01010101011111111100001100110
D_9 = 00111100011110101010110011011

C_{10} = 01010101111111100001100110011
D_{10} = 11110001111010101010110011001

C_{11} = 01010111111110000110011001011
D_{11} = 11000111101010101100110011011

C_{12} = 01011111111000011001100101011
D_{12} = 00011110101010101100110011111

C_{13} = 01111111100001100110010101011
D_{13} = 01111010101010110011001111100

C_{14} = 11111110000110011001010101011
D_{14} = 11101010101011001100111100011

C_{15} = 11111000011001100101010101111
D_{15} = 10101010101100110011110000111

C_{16} = 11110000110011001010101011111
D_{16} = 01010101011001100111100011111

Selanjutnya dibentuk kunci K_n, dengan menggabungkan setiap C_n dan D_n. Sebagai contoh, kita mempunyai

C₁D₁ = 1110000 1100110 0101010 1011111 1010101 0110011 0011110 0011110

Setiap pasang mempunyai ukuran 56 bit lalu dilakukan permutasi sesuai table PC-2 hingga akan diperoleh kunci dengan ukuran 48 bit.

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

Maka akan diperoleh kunci-kunci sebagai berikut

K₁ = 000110 110000 001011 101111 111111 000111 000001 110010
K₂ = 011110 011010 111011 011001 110110 111100 100111 100101
K₃ = 010101 011111 110010 001010 010000 101100 111110 011001
K₄ = 011100 101010 110111 010110 110110 110011 010100 011101
K₅ = 011111 001110 110000 000111 111010 110101 001110 101000
K₆ = 011000 111010 010100 111110 010100 000111 101100 101111
K₇ = 111011 001000 010010 110111 111101 100001 100010 111100
K₈ = 111101 111000 101000 111010 110000 010011 101111 111011
K₉ = 111000 001101 101111 101011 111011 011110 011110 000001
K₁₀ = 101100 011111 001101 000111 101110 100100 011001 001111
K₁₁ = 001000 010101 111111 010011 110111 101101 001110 000110
K₁₂ = 011101 010111 000111 110101 100101 000110 011111 101001
K₁₃ = 100101 111100 010111 010001 111110 101011 101001 000001
K₁₄ = 010111 110100 001110 110111 111100 101110 011100 111010
K₁₅ = 101111 111001 000110 001101 001111 010011 111100 001010
K₁₆ = 110010 110011 110110 001011 000011 100001 011111 110101

Langkah DES 2 : Enkripsi setiap 64 bit blok plaintext

Jika dimiliki plaintext : M = 0123456789ABCDEF, dimana M adalah dalam format hexadesimal, maka kita dapat menuliskan M di dalam format angka biner.

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

Langkah yang harus kita lakukan adalah menyusun 64 bit data plaintext M sesuai dengan tabel *Initial Permutation* di bawah ini.

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

Dengan permutasi plaintext M sesuai dengan tabel IP tersebut kita dapatkan :

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

Selanjutnya kita bagi blok IP tersebut menjadi dua bagian L_0 dan R_0 dengan masingmasing mempunyai ukuran 32 bit.
L_0 = 1100 1100 0000 0000 1100 1100 1111 1111
R_0 = 1111 0000 1010 1010 1111 0000 1010 1010

Dari nilai L_0 dan R_0 dicari nilai L_1 dan R_1 dengan aturan berikut :

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

Tanda + menunjukkan operasi penambahan XOR (bit demi bit dengan modulo 2)
Sebagai contoh untuk n=1 kita dapatkan :
K_1 = 000110 110000 001011 101111 111111 000111 000001 110010
$L_1 = R_0$ = 1111 0000 1010 1010 1111 0000 1010 1010
$R_1 = L_0 + f(R_0, K_1)$
Fungsi f diperoleh melalui langkah-langkah berikut :

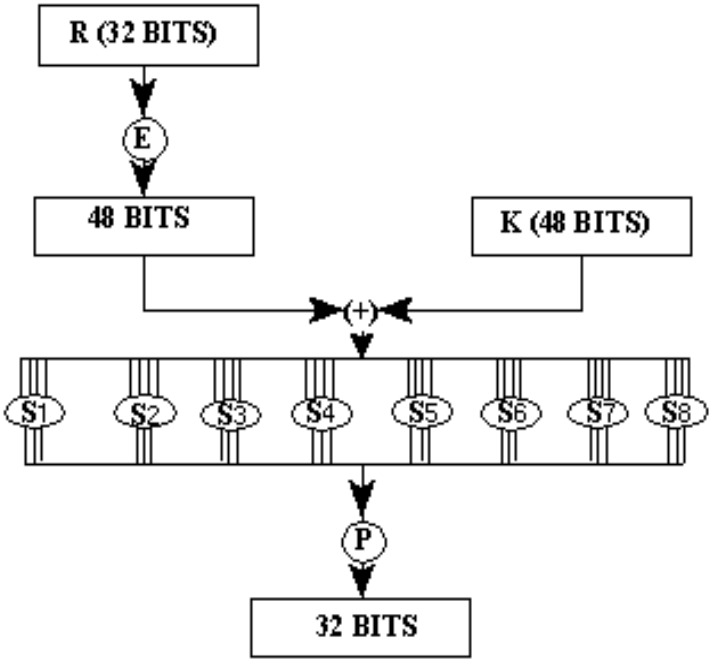


Figure *Calculation of f*(R, K)

Langkah Pertama : mengembangkan blok R dari 32 bit menjadi 48 bit dengan menggunakan fungsi seleksi tabel E.

E BIT-SELECTION TABLE

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

Untuk
R_0 = 1111 0000 1010 1010 1111 0000 1010 1010
Diperoleh nilai berikut :
E(R_0) = 011110 100001 010101 010101 011110 100001 010101 010101

Langkah Kedua: melakukan operasi XOR pada $E(R_{n-1})$ dengan kunci K_n
Sebagai contoh untuk K_1, $E(R_0)$:
$\boldsymbol{K_1}$ = 000110 110000 001011 101111 111111 000111 000001 110010
$\mathbf{E}(\boldsymbol{R_0})$ = 011110 100001 010101 010101 011110 100001 010101 010101
$\boldsymbol{K_1}+\mathbf{E}(\boldsymbol{R_0})$ = 011000 010001 011110 111010 100001 100110 010100 100111

Langkah Ketiga : di lakukan sebuah operasi pada setiap grup yang terdiri dari 6 bit. Digunakan setiap grup tersebut sebagai alamat pada tabel "S-boxes".

Jika $\boldsymbol{K_n} + \mathbf{E}(\boldsymbol{R_{n-1}}) = \boldsymbol{B_1B_2B_3B_4B_5B_6B_7B_8}$
Maka akan dicari $\boldsymbol{S_1(B_1)S_2(B_2)S_3(B_3)S_4(B_4)S_5(B_5)S_6(B_6)S_7(B_7)S_8(B_8)}$
dengan $\boldsymbol{S_i(B_i)}$ menunjukkan keluaran dari **S** box ke-$\boldsymbol{I}$
Blok-blok S yang digunakan adalah sebagai berikut :

S1

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

S2

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

S3

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

S4

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

S5

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

S6

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

S7

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

S8

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

Contoh penggunaan S1 adalah sebagai berikut :

S1

No. Row	Column Number															
	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	14	4	13	1	2	15	11	8	3	10	6	12	5	9	0	7
1	0	15	7	4	14	2	13	1	10	6	12	11	9	5	3	8
2	4	1	14	8	13	6	2	11	15	12	9	7	3	10	5	0
3	15	12	8	2	4	9	1	7	5	11	3	14	10	0	6	13

Jika B adalah 6 bit blok, maka $S_1(B)$ ditentukan sebagai berikut :
Bit pertama dan terakhir dari ***B*** mewakili bilangan desimal 0-3 (00-11). Anggap bilangan ini sebagai ***i***. 4 bit tengah lainnya mewakili bilangan desimal 0-15 (0000-1111) dan anggap bilangan ini sebagai ***j***. Dengan demikian kita dapat melihat ***i*** sebagai baris dan ***j*** sebagai kolom. Sebagai contoh jika input B = 011011, maka ***i*** = 01 = 1 dan ***j*** = 1101 =13. Pada baris 1 dan kolom 13 kita dapatkan nilai 5, jadi outputnya adalah 0101.

Contoh untuk perhitungan pertama :
$K_1 + E(R_0)$ = 011000 010001 011110 111010 100001 100110 010100 100111.
$S_1(B_1)S_2(B_2)S_3(B_3)S_4(B_4)S_5(B_5)S_6(B_6)S_7(B_7)S_8(B_8)$ = 0101 1100 1000 0010 1011 0101 1001 0111

Langkah Keempat : melakukan permutasi P dari output tersebut

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

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

Maka akan diperoleh :
f = 0010 0011 0100 1010 1010 1001 1011 1011

$R_1 = L_0 + f(R_0, K_1)$
= 1100 1100 0000 0000 1100 1100 1111 1111
+ 0010 0011 0100 1010 1010 1001 1011 1011
= 1110 1111 0100 1010 0110 0101 0100 0100

Selanjutnya kita harus menghitung $L_2 = R_1$ dan kemudian menghitung $R_2 = L_1 + f(R_1,K_2)$ dan seterusnya sampai iterasi ke-16 sehingga pada iterasi terakhir kita sudah mempunyai L_{16} dan R_{16}. Kemudian kita balik urutannya menjadi $R_{16}L_{16}$.

Lalu menerapkan permutasi terakhir IP^{-1} dengan tabel berikut :

IP^{-1}

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

Contoh : jika kita telah menyelesaikan ke 16 iterasi, kita dapatkan L_{16} dan R_{16}

L_{16} = 0100 0011 0100 0010 0011 0010 0011 0100
R_{16} = 0000 1010 0100 1100 1101 1001 1001 0101

Selanjutnya kita balik dan terapkan permutasi akhir :
$R_{16}L_{16}$ = 00001010 01001100 11011001 10010101 01000011 01000010 00110010 00110100
IP^{-1} = 10000101 11101000 00010011 01010100 00001111 00001010 10110100 00000101
Di dalam bilangan hexa kita dapatkan 85E813540F0AB405

Catatan mengenai DES

Pengisian kotak-S DES masih menjadi misteri.
Delapan putaran sudah cukup untuk membuat cipherteks sebagai fungsi acak dari setiap bit plainteks dan setiap bit cipherteks
Dari penelitian, DES dengan jumlah putaran yang kurang dari 16 ternyata dapat dipecahkan dengan *known-plaintext attack.*
Deskripsi chipertext DES dapat dilakukan dengan algoritma yang sama tetapi dengan urutan penggunaan kunci yang dibalik (dari K16 dahulu hingga ke K1)

Perbandingan perubahan masukan plaintext terhadap chipertext menggunakan DES

Plaintext	Key	Chipertext
0000000000000000	0000000000000000	8CA64DE9C1B123A7
0000000000000001	0000000000000000	166B40B44ABA4BD6
0000000000000002	0000000000000000	06E7EA22CE92708F

DES memiliki prinsip Diffusion. Konsep ini dilakukan dengan mengusahakan setiap bit dari setiap *plaintext* mempengaruhi setiap bit *ciphertext* dan setiap bit kunci untuk mempengaruhi setiap bit *ciphertext*.

II. Triple DES Block Diagram (ECB Mode)

DES is broken

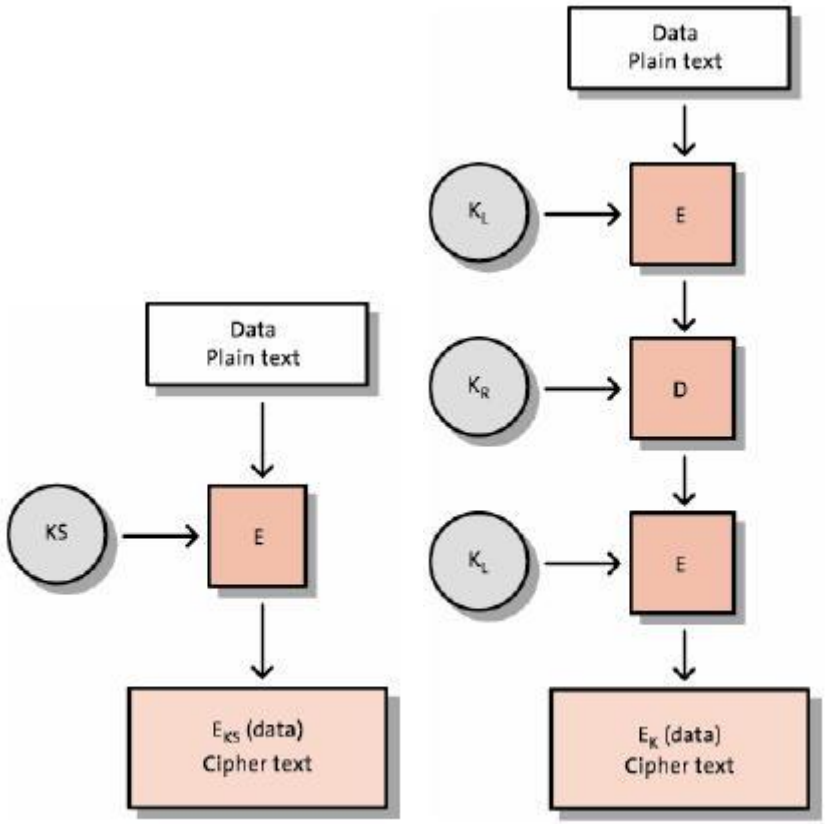
Due to the available power computation DES is not safe today
Brute force:
– Given a cipher text and a plain text
– How much does it takes to try all the keys?
– 2^56 encryption!
How many days does it takes to compute 2^56 encryption?
If 1 encryption per millisecond 833,999,931
per microsecond 833,999
per nanosecond 833
If 100 devices in parallel 8 days
Solutions? 3DES

Tahun 1998, *Electronic Frontier Foundation* (*EFE*) merancang dan membuat perangkat keras khusus untuk menemukan kunci DES secara *exhaustive search key* dengan biaya $250.000 dan diharapkan dapat menemukan kunci selama 5 hari

Tahun 1999, kombinasi perangkat keras *EFE* dengan kolaborasi internet yang melibatkan lebih dari 100.000 komputer dapat menemukan kunci DES kurang dari 1 hari.

triple-DES is secure, operationally, longer keys mean more security

Kelemahan DES biasanya terletak pada 56 bit kuncinya.
Perbaikan dilakukan dengan menerapkan Triple DES dengan panjang kunci 56 x 3 = 168 bit kunci
Algoritma 3DES adalah suatu algoritma pengembangan dari algoritma DES (*Data Encryption Standard*). Perbedaan DES dengan 3DES terletak pada panjangnya kunci yang digunakan. Pada DES menggunakan satu kunci yang panjangnya 56-bit, sedangkan pada 3DES menggunakan 3 kunci yang panjangnya 168- bit (masing-masing panjangnya 56-bit). Pada 3DES, 3 kunci yang digunakan bisa bersifat saling bebas ($K_1 \neq K_2 \neq K_3$) atau hanya dua buah kunci yang saling bebas dan satu kunci lainnya sama dengan kunci pertama ($K_1 \neq K_2$ dan $K_3 = K_1$). Karena tingkat kerahasiaan algoritma 3DES terletak pada panjangnya kunci yang digunakan, maka penggunaan algoritma 3DES dianggap lebih aman dibandingkan dengan algoritma DES



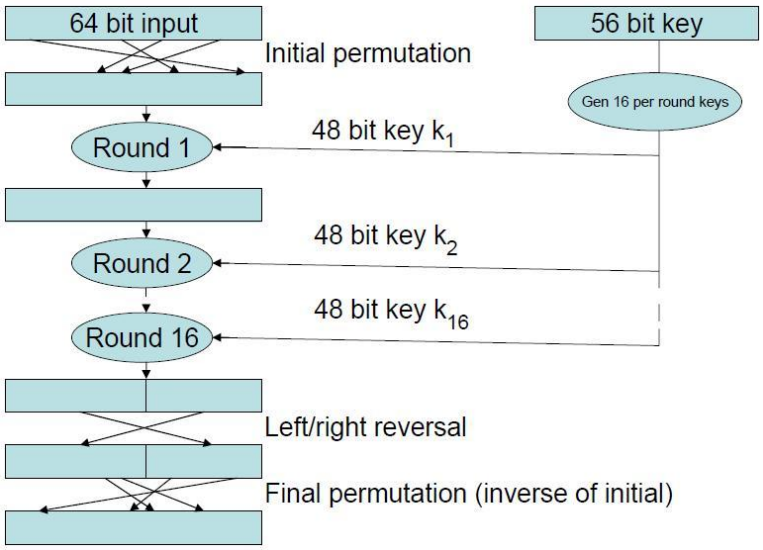
TDEA Encryption Operation:

$$I \rightarrow \boxed{DES\ E_{K1}} \rightarrow \boxed{DES\ D_{K2}} \rightarrow \boxed{DES\ E_{K3}} \rightarrow O$$

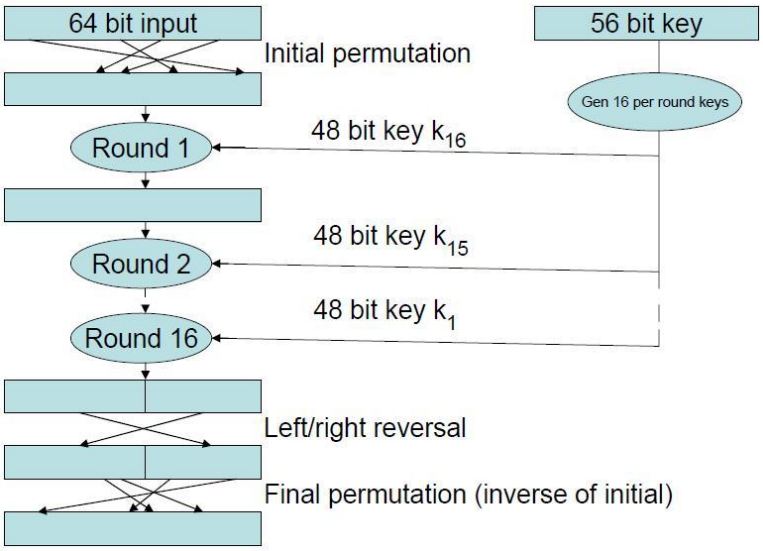
TDEA Decryption Operation:

$$I \rightarrow \boxed{DES\ D_{K3}} \rightarrow \boxed{DES\ E_{K2}} \rightarrow \boxed{DES\ D_{K1}} \rightarrow O$$

DES Enkripsi



DES Deskripsi



III. International Data Encryption Algorithm

Cipher International Data Encryption Algorithm (IDEA) adalah algoritma enkripsi blok kunci yang aman dan rahasia yang dikembangkan oleh James Massey dan Xuejia Lai. Algoritma ini berkembang pada 1992 dari algoritma semula yang disebut dengan Proposed Encryption Standard and The Inproved Proposed Encryption Standard. IDEA beroperasi pada blok plaintext 64 bit dan menggunakan kunci 128 bit.
Algoritma IDEA menggunakan delapan round dan beroperasi pada subblok 16 bit dengan menggunakan kalkulasi aljabar yang dapat digunakan untuk implementasi hardware. Operasi ini adalah penjumlahan modulo 216, perkalian modulo 216 + 1, dan XOR. Dengan kunci 128 bitnya, cipher IDEA lebih sulit untuk dibobol daripada DES.

Pada Algoritma IDEA, plaintext memiliki panjang 64 bit dan kunci sepanjang 128 bit. Metodologi dari algoritma IDEA menggunakan operasi yang berbeda seperti berikut ini :
⊕ Bit per bit XOR 16 bit sub-block
Penambahan 16 bit integer modulo 2^{16}
Perkalian 16 bit integer modulo $2^{16}+1$
Operasi ini tidak berlaku hukum distributif atau hukum asosiatif.

Langkah-langkah IDEA adalah sebagai berikut :

Blok pesan terbuka dengan lebar 64-bit , X, dibagi menjadi 4 sub-blok 16-bit, X_1, X_2, X_3, X_4, sehingga X = (X_1, X_2, X_3, X_4). Keempat sub-blok 16-bit itu ditransformasikan menjadi sub-blok 16-bit, Y_1, Y_2, Y_3, Y_4, sebagai pesan rahasia 64-bit Y = (Y_1, Y_2, Y_3, Y_4) yang berada dibawah kendali 52 sub_blok kunci 16-bit yang dibentuk dari blok kunci 128 bit.

Keempat sub-blok 16-bit, X_1, X_2, X_3, X_4, digunakan sebagai masukan untuk putaran pertama dari algoritma IDEA. Dalam setiap putaran dilakukan operasi XOR, penjumlahan, perkalian antara dua sub-blok 16-bit dan diikuti pertukaran antara sub-blok 16-bit putaran kedua dan ketiga. Keluaran putaran sebelumnya menjadi masukan putaran berikutnya. Setelah putaran kedelapan dilakukan transformasi keluaran yang dikendalikan oleh 4 sub-blok kunci 16-bit.

Pada setiap putaran dilakukan operasi-operasi sebagai berikut :

1) Perkalian X_1 dengan sub-kunci pertama
2) Penjumlahan X_2 dengan sub-kunci kedua
3) Pejumlahan X_3 dengan sub kunci ketiga
4) Perkalian X_4 dengan sub kunci keempat
5) Operasi XOR hasil langkah 1) dan 3)
6) Operasi XOR hasil angkah 2) dan 4)
7) Perkalian hasil langkah 5) dengan sub-kunci kelima
8) Penjumlahan hasil langkah 6) dengan langkah 7)
9) Perkalian hasil langkah 8) dengan sub-kunci keenam
10)Penjumlahan hasil langah 7) dengan 9)
11)Operasi XOR hasil langkah 1) dan 9)
12)Operasi XOR hasil langkah 3) dan 9)
13)Operasi XOR hasil langkah 2) dan 10)
14)Operasi XOR hasil langkah 4) dan 10)

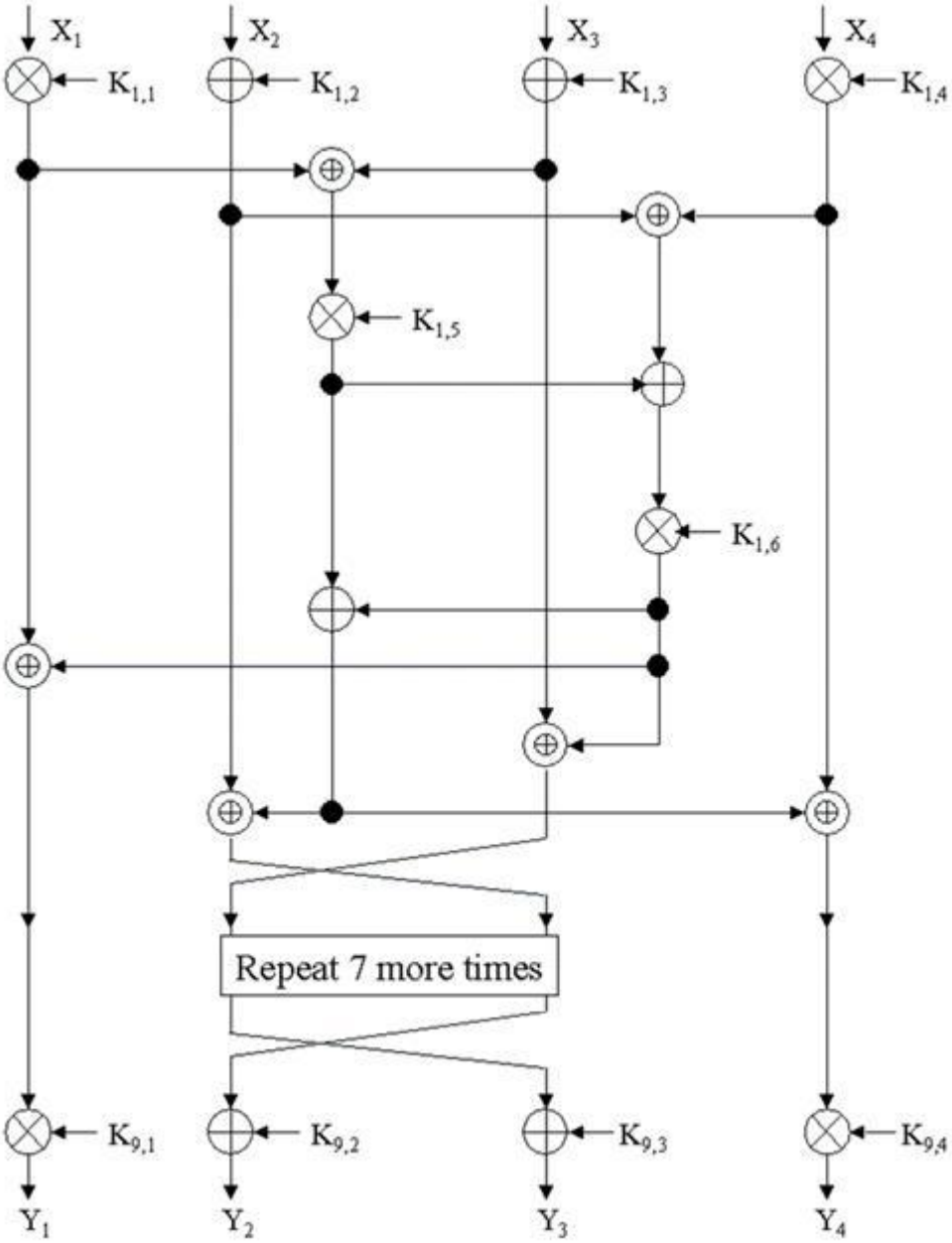
Untuk lebih jelasnya bisa dilihat dari notasi-notasi berikut :

IDEA Round

- 14 Langkah
- © = Xor
- ¤ = Perkalian Modulo $2^{16} + 1$
- + = Penambahan modulo 2^{16}

1. A = X1 ¤ K1
2. B = X2 + K2
3. C = X3 + K3
4. D = X4 ¤ K4
5. E = A © C
6. F = B © D
7. G = E ¤ K5
8. H = G + F
9. J = H ¤ K6
10. L = J + G
11. R1 = A © J
12. R2 = C © J
13. R3 = B © L
14. R4 = D © L

Di bawah ini adalah gambar dari putaran yang dimiliki oleh IDEA :



Keluaran setiap putaran adalah 4 sub-blok yang dihasilkan pada langkah 11), 12), 13), dan 14) dan menjadi masukan putaran berikutnya.

Setelah putaran kedelapan terdapat transformasi keluaran, yaitu :

1) Perkalian X_1 dengan sub-kunci pertama

2) Penjumlahan X_2 dengan sub-kuci ketiga

3) Penjumlahan X_3 dengan sub-kunci kedua

4) Perkalian X_4 dengan sub-kunci keempat

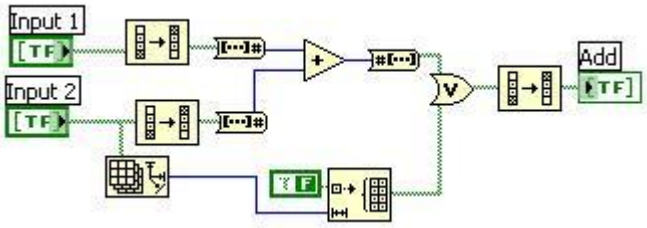
Terahir, keempat sub-blok 16-bit 16-bit yang merupakan hasil operasi 1), 2), 3), dan 4) ii digabung kembali menjadi blok pesan rahasia 64-bit. Di bawah ini adalah gambar dari transformasi luaran.

Untuk proses Deskripsi IDEA :
Proses dekripsi menggunakan algoritma yang sama dengan proses enkripsi tetapi 52 buah sub-blok kunci yang digunakan masing-masing merupakan hasil turunan 52 buah sub-blok kunci enkripsi.

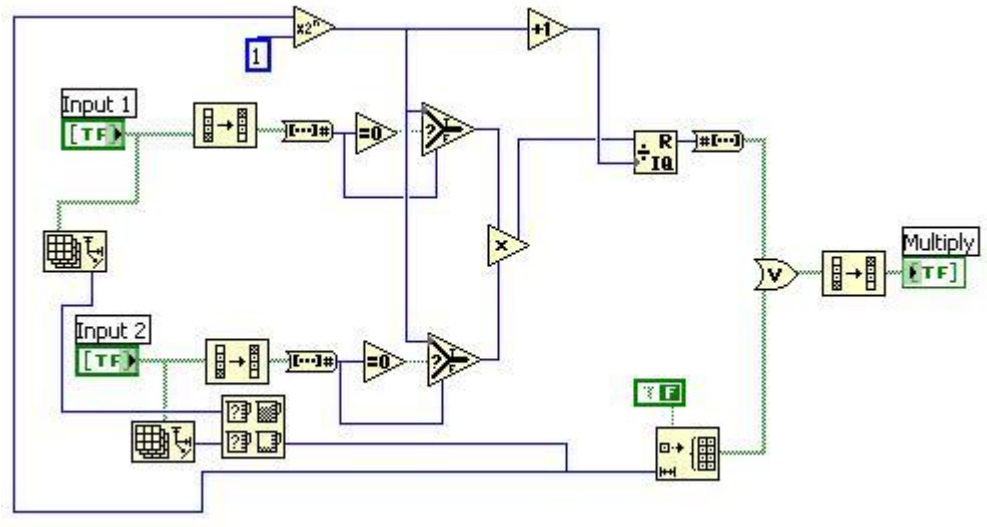
Sub-blok Kunci Enkripsi		Sub-blok Kunci Dekripsi	
Putaran ke-1	$Z_{11}\ Z_{21}\ Z_{31}\ Z_{41}\ Z_{51}\ Z_{61}$	Putaran ke-1	$(Z_{19})^{-1}\ -Z_{39}\ -Z_{29}\ (Z_{49})^{-1}\ Z_{58}\ Z_{68}$
Putaran ke-2	$Z_{12}\ Z_{22}\ Z_{32}\ Z_{42}\ Z_{52}\ Z_{62}$	Putaran ke-2	$(Z_{18})^{-1}\ -Z_{38}\ -Z_{28}\ (Z_{48})^{-1}\ Z_{57}\ Z_{67}$
Putaran ke-3	$Z_{13}\ Z_{23}\ Z_{33}\ Z_{43}\ Z_{53}\ Z_{63}$	Putaran ke-3	$(Z_{17})^{-1}\ -Z_{37}\ -Z_{27}\ (Z_{47})^{-1}\ Z_{56}\ Z_{66}$
Putaran ke-4	$Z_{14}\ Z_{24}\ Z_{34}\ Z_{44}\ Z_{54}\ Z_{64}$	Putaran ke-4	$(Z_{16})^{-1}\ -Z_{36}\ -Z_{26}\ (Z_{46})^{-1}\ Z_{55}\ Z_{65}$
Putaran ke-5	$Z_{15}\ Z_{25}\ Z_{35}\ Z_{45}\ Z_{55}\ Z_{65}$	Putaran ke-5	$(Z_{15})^{-1}\ -Z_{35}\ -Z_{25}\ (Z_{45})^{-1}\ Z_{54}\ Z_{64}$
Putaran ke-6	$Z_{16}\ Z_{26}\ Z_{36}\ Z_{46}\ Z_{56}\ Z_{66}$	Putaran ke-6	$(Z_{14})^{-1}\ -Z_{34}\ -Z_{24}\ (Z_{44})^{-1}\ Z_{53}\ Z_{63}$
Putaran ke-7	$Z_{17}\ Z_{27}\ Z_{37}\ Z_{47}\ Z_{57}\ Z_{67}$	Putaran ke-7	$(Z_{13})^{-1}\ -Z_{33}\ -Z_{23}\ (Z_{43})^{-1}\ Z_{52}\ Z_{62}$
Putaran ke-8	$Z_{18}\ Z_{28}\ Z_{38}\ Z_{48}\ Z_{58}\ Z_{68}$	Putaran ke-8	$(Z_{12})^{-1}\ -Z_{32}\ -Z_{22}\ (Z_{42})^{-1}\ Z_{51}\ Z_{61}$
Transformasi output	$Z_{1\ 8,5}\ Z_{2\ 8,5}\ Z_{3\ 8,5}\ Z_{4\ 8,5}$	Transformasi output	$(Z_{11})^{-1}\ -Z_{21}\ -Z_{31}\ (Z_{41})^{-1}$

Penjelasan operasi-operasi yang digunakan pada IDEA :

- XOR (cukup jelas)
- Penambahan modulo 2^{16}
  Kedua sub-blok itu dianggap sebagai representasi biner dari integer biasa
  Penjumlahan dilakukan per-setiap bit-nya.
  Jika diakhir penjumlahan terdapat bit ke-17 (lebih dari 16), bit ini ditiadakan
  Contoh program LabVIEW-nya :

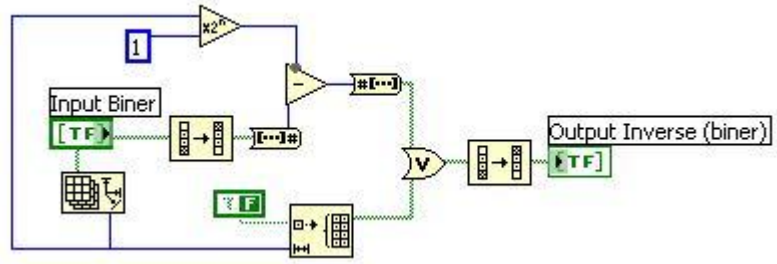


- Perkalian Modulo $2^{16} + 1$
  Kedua sub-blok 16-bit itu dianggap sebagai representasi biner dari integer biasa kecuali sub-blok nol dianggap mewakili integer 2^{16}
  Hasilnya kemudian di-mod-kan dengan $2^{16}+1$.
  Contoh program LabVIEW-nya :



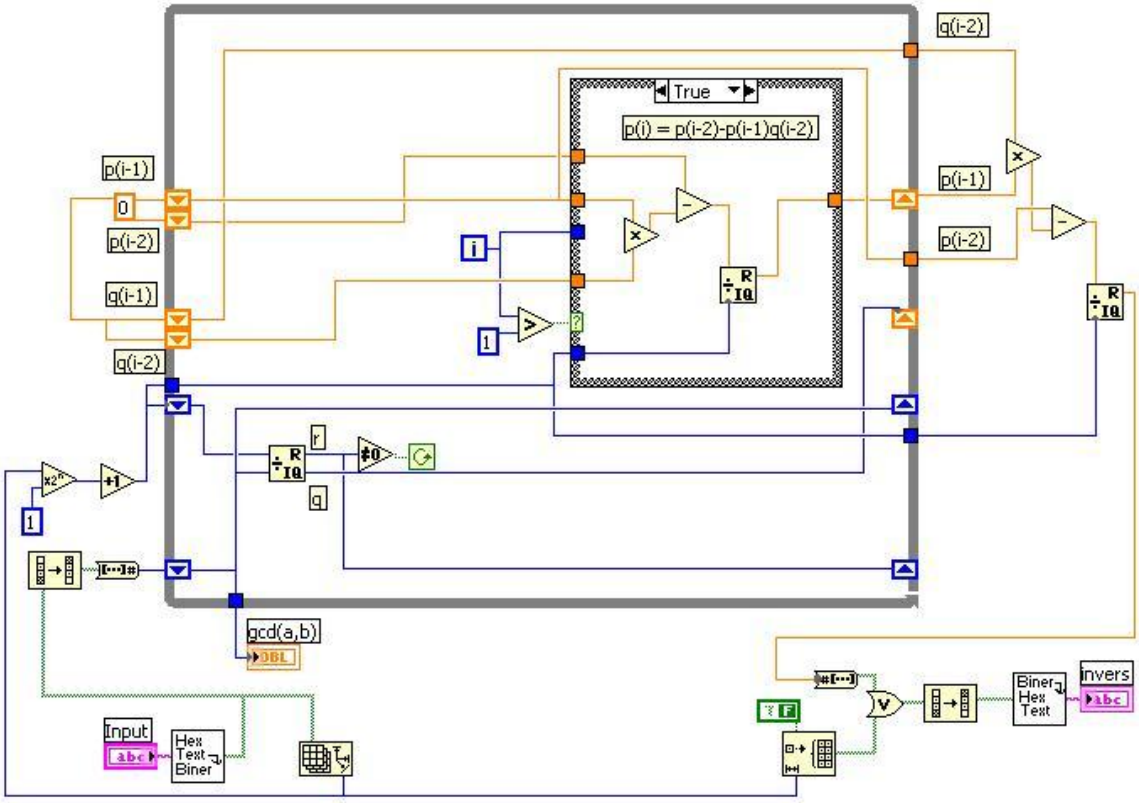
- Invers Penambahan modulo 2^{16}

  -Z merupakan invers penjumlahan modulo 2^{16} dri Z, dimana Z (-Z)= 0
  Maka operasinya adalah dengan melakukan pengurangan (-Z) terhadap 2^{16}
  Contoh program LabVIEW-nya :



- Invers Perkalian Modulo $2^{16} + 1$

  Z^{-1} merupakan invers perkalian modulo $2^{16}+1$ dari Z, dimana $Z\,Z^{-1} = 1$
  Nilai Z^{-1} diperoleh melalui algoritma *Extended Euclidean*
  Contoh program LabVIEW-nya :



THE EXTENDED EUCLIDEAN ALGORITHM

DIVISION

If a and b are positive integers, there exist unique non-negative integers q and r so that,

$a = qb + r$, where $0 \leq r < b$.

q is called the ***quotient*** and r the ***remainder***.

EUCLIDEAN ALGORITHM

The ***greatest common divisor*** of integers a and b, denoted by *gcd(a,b)*, is the largest integer that divides (without remainder) both a and b. So, for example:

gcd(15, 5) = 5, gcd(7, 9) = 1,gcd(12, 9) = 3,gcd(81, 57) = 3.

The gcd of two integers can be found by repeated application of the division algorithm, this is known as the ***Euclidean Algorithm***. You repeatedly divide the divisor by the remainder until the remainder is 0. The gcd is the last non-zero remainder in this algorithm. The following example shows the algorithm.

Finding the gcd of 81 and 57 by the Euclidean Algorithm:

81 = 1(57) + 24
57 = 2(24) + 9
24 = 2(9) + 6
9 = 1(6) + **3**
6 = 2(3) + 0.

THE EUCLIDEAN ALGORITHM

It is well known that if the gcd(a, b) = r then there exist integers p and s so that:

p(a) + s(b) = r.

INVERSE MOD n

The inverse of x exists if and only if gcd(x, n) = 1. We now know that if this is true, there exist integers p and s so that

px + sn = 1.

But this says that px = 1 + (-s)n, or in other words, $px \equiv 1 \pmod{n}$. So, p (reduced mod n if need be) is the inverse of x mod n. The extended Euclidean algorithm will give us a method for calculating p efficiently

EXTENDED EUCLIDEAN ALGORITHM

We will number the steps of the Euclidean algorithm starting with step 0. The quotient obtained at step i will be denoted by q_i. As we carry out each step of the Euclidean algorithm, we will also calculate an auxillary number, p_i. For the first two steps, the value of this number is given: $p_0 = 0$ and $p_1 = 1$. For the remainder of the steps, we recursively calculate $p_i = p_{i-2} - p_{i-1}\, q_{i-2} \pmod{n}$. Continue this calculation for one step beyond the last step of the Euclidean algorithm.

The algorithm starts by "dividing" n by x. If the last non-zero remainder occurs at step k, then if this remainder is 1, x has an inverse and it is pk+2. (If the remainder is not 1, then x does not have an inverse.)

EXAMPLE

Find the inverse of 15 mod 26. (15 yang mau dicar inversnya) (15 adalah b, 26 adalah a)

a = q.b + r

Step 0: 26 = 1(15) + 11 $\quad p_0 = 0$

Step 1: 15 = 1(11) + 4 $\quad p_1 = 1$

Step 2: 11 = 2(4) + 3 $\quad p_2 = 0 - 1(1) \bmod 26 = 25$

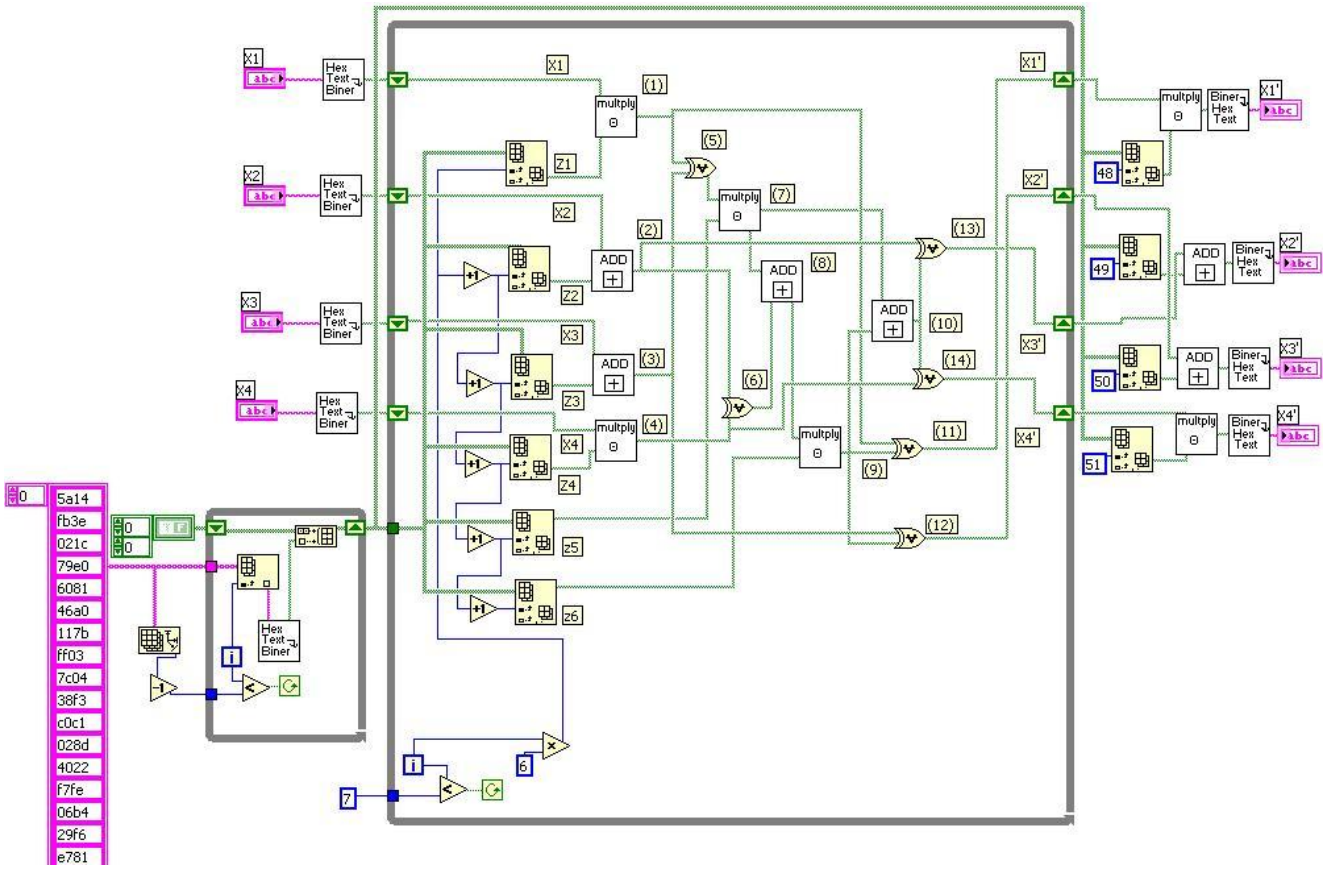
Step 3: 4 = 1(3) + 1 $\quad p_3 = 1 - 25(1) \bmod 26 = -24 \bmod 26 = 2$

Step 4: 3 = 3(1) + 0 $\quad p_4 = 25 - 2(2) \bmod 26 = 21$

$p_5 = 2 - 21(1) \bmod 26 = -19 \bmod 26 = 7$

Notice that $15(7) = 105 = 1 + 4(26) \equiv 1 \pmod{26}$.

Contoh program IDEA dalam LabVIEW :



Contoh hasil proses IDEA adalah sebagai berikut :

Plaintext	7fa9 (X_1)	1c37 (X_2)	ffb3 (X_3)	df05 (X_4)

Round	Round Output			
	X_1	X_2	X_3	X_4
	7fa9	1c37	ffb3	df05
1	c579	f2ff	0fbd	0ffc
2	d7a2	80cb	9a61	27c5
3	ab6c	e2f9	f3be	36bd
4	ef5b	9cd2	6808	3019
5	7e09	2445	d223	d639
6	4a6e	d7ac	ac8c	8b09
7	244d	6f5c	4459	3a9c
8	0f86	7b0b	54df	759f
9	106b	dbfd	f323	0876
	(Y_1)	(Y_2)	(Y_3)	(Y_4)

Ciphertext	106b (Y_1)	dbfd (Y_2)	f323 (Y_3)	0876 (Y_4)

Subkey untuk melakukan enkripsi yang digunakan adalah sbb :

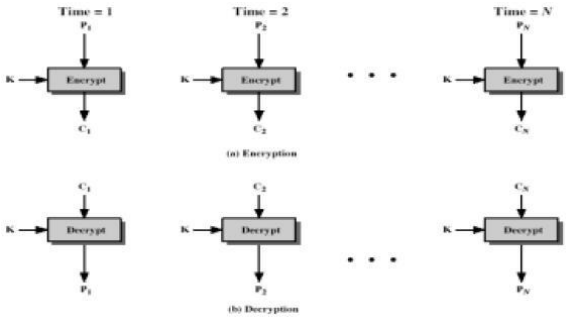
Z1	=	5a14	Z14	=	f7fe	Z27	=	dff8	Z40	=	6a01
Z2	=	fb3e	Z15	=	06b4	Z28	=	1ad0	Z41	=	6b42
Z3	=	021c	Z16	=	29f6	Z29	=	a7d9	Z42	=	9f67
Z4	=	79e0	Z17	=	e781	Z30	=	f010	Z43	=	c043
Z5	=	6081	Z18	=	8205	Z31	=	e3cf	Z44	=	8f3c
Z6	=	46a0	Z19	=	1a80	Z32	=	0304	Z45	=	0c10
Z7	=	117b	Z20	=	45ef	Z33	=	17bf	Z46	=	28d4
Z8	=	ff03	Z21	=	fc0d	Z34	=	f035	Z47	=	022f
Z9	=	7c04	Z22	=	6853	Z35	=	a14f	Z48	=	7fe0
Z10	=	38f3	Z23	=	ecf8	Z36	=	b3e0	Z49	=	Cf80
Z11	=	c0c1	Z24	=	0871	Z37	=	21c7	Z50	=	871e
Z12	=	028d	Z25	=	0a35	Z38	=	9e06	Z51	=	7818
Z13	=	4022	Z26	=	008d	Z39	=	0814	Z52	=	2051

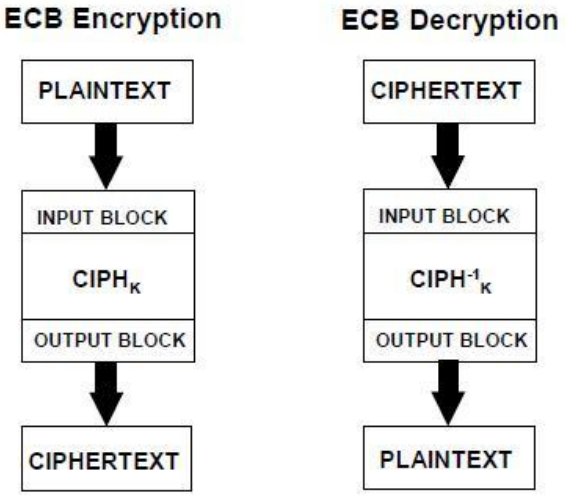
Block cipher

Block Cipher adalah algoritma enkripsi yang akan membagi-bagi *plaintext* yang akan dikirimkan dengan ukuran tertentu (disebut blok) dengan panjang t, dan setiap blok dienkripsi dengan menggunakan kunci yang sama. Pada umumnya, *block cipher* memproses *plaintext* dengan blok yang relatif panjang lebih dari 64 bit, untuk mempersulit penggunaan pola-pola serangan yang ada untuk membongkar kunci.
Contoh block chipper : DES, 3DES, GOST, RC5, AES, Blowfish, IDEA, LOKI, RC2, FEAL, Lucifer, CAST, CRAB, SAFER, Twofish, Serpent, RC6, MARS, Camellia, 3-WAY, MMB, SkipJack, dll

Untuk menambah kehandalan algoritma ini, dikembangkan pula beberapa tipe proses enkripsi, yaitu :

- ECB (*Electronic Code Book*)
  Model operasi EBC adalah implementasi DES yang paling sederhana. Setiap 64 bit plaintext dikodekan sendiri-sendiri menjadi chipertext dengan kunci yang sama. Jadi, jika plaintext terdiri dari 128 bit, maka setiap 64 bit akan dikodekan secara terpisah. Model semacam ini biasa diimplementasikan pada aplikasi yang terkait dengan transmisi data tunggal atau data yang tidak terlalu besar seperti transmisi kunci. Jadi, jika anda ingin mentransmisikan kunci untuk enkripsi atau dekripsi, anda dapat menggunakan model operasi ECB ini.





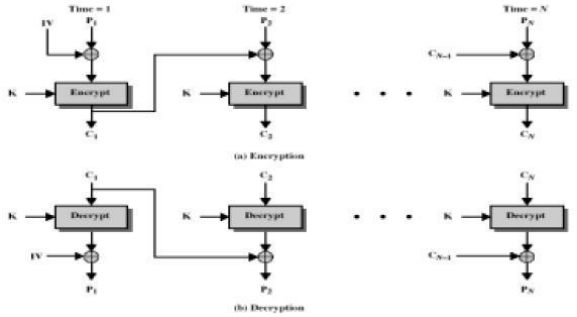
Example :

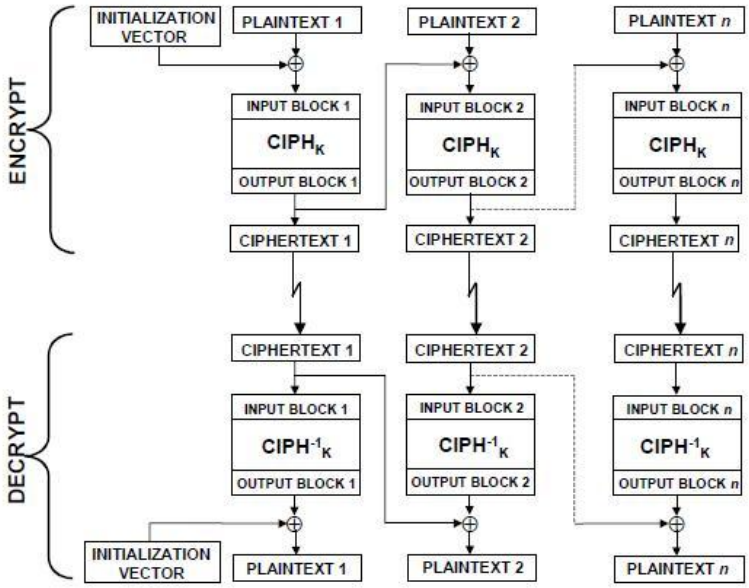
F.1.1 ECB-AES128.Encrypt
```
Key 2b7e151628aed2a6abf7158809cf4f3c
Block #1
Plaintext 6bc1bee22e409f96e93d7e117393172a
Input Block 6bc1bee22e409f96e93d7e117393172a
Output Block 3ad77bb40d7a3660a89ecaf32466ef97
Ciphertext 3ad77bb40d7a3660a89ecaf32466ef97
Block #2
Plaintext ae2d8a571e03ac9c9eb76fac45af8e51
Input Block ae2d8a571e03ac9c9eb76fac45af8e51
Output Block f5d3d58503b9699de785895a96fdbaaf
Ciphertext f5d3d58503b9699de785895a96fdbaaf
Block #3
Plaintext 30c81c46a35ce411e5fbc1191a0a52ef
Input Block 30c81c46a35ce411e5fbc1191a0a52ef
Output Block 43b1cd7f598ece23881b00e3ed030688
Ciphertext 43b1cd7f598ece23881b00e3ed030688
Block #4
Plaintext f69f2445df4f9b17ad2b417be66c3710
Input Block f69f2445df4f9b17ad2b417be66c3710
Output Block 7b0c785e27e8ad3f8223207104725dd4
Ciphertext 7b0c785e27e8ad3f8223207104725dd4
```

F.1.2 ECB-AES128.Decrypt
```
Key 2b7e151628aed2a6abf7158809cf4f3c
Block #1
Ciphertext 3ad77bb40d7a3660a89ecaf32466ef97
Input Block 3ad77bb40d7a3660a89ecaf32466ef97
```

```
Output Block 6bc1bee22e409f96e93d7e117393172a
Plaintext 6bc1bee22e409f96e93d7e117393172a
Block #2
Ciphertext f5d3d58503b9699de785895a96fdbaaf
Input Block f5d3d58503b9699de785895a96fdbaaf
Output Block ae2d8a571e03ac9c9eb76fac45af8e51
Plaintext ae2d8a571e03ac9c9eb76fac45af8e51
Block #3
Ciphertext 43b1cd7f598ece23881b00e3ed030688
Input Block 43b1cd7f598ece23881b00e3ed030688
Output Block 30c81c46a35ce411e5fbc1191a0a52ef
Plaintext 30c81c46a35ce411e5fbc1191a0a52ef
Block #4
Ciphertext 7b0c785e27e8ad3f8223207104725dd4
Input Block 7b0c785e27e8ad3f8223207104725dd4
Output Block f69f2445df4f9b17ad2b417be66c3710
Plaintext f69f2445df4f9b17ad2b417be66c3710
```

- CBC (*Cipher Block Chaining*)
  Model ini mempunyai mekanisme yang sedikit berbeda dengan ECB. Di dalam model ini, data tetap dibagi menjadi beberapa blok, masing-masing 64 bit. Perbedaannya adalah input dari algoritma DES adalah hasil operasi XOR dari chipertext yang telah dihasilkan sebelumnya dengan plaintext yang akan dienkripsi. Untuk enkripsi pada 64 bit blok pertama diperlukan vektor inisialisasi yang digunakan untuk operasi XOR dengan plaintext yang akan dienkripsi. Maka, baik pengirim maupun penerima harus mengetahui nila vektor inisialisasi untuk dapat melakukan enkripsi maupun dekripsi. Aplikasi yang biasa diimplentasikan adalah aplikasi transmisi data yang berbasis blok atau aplikasi otentifikasi.





Example :

F.2.1 CBC-AES128.Encrypt

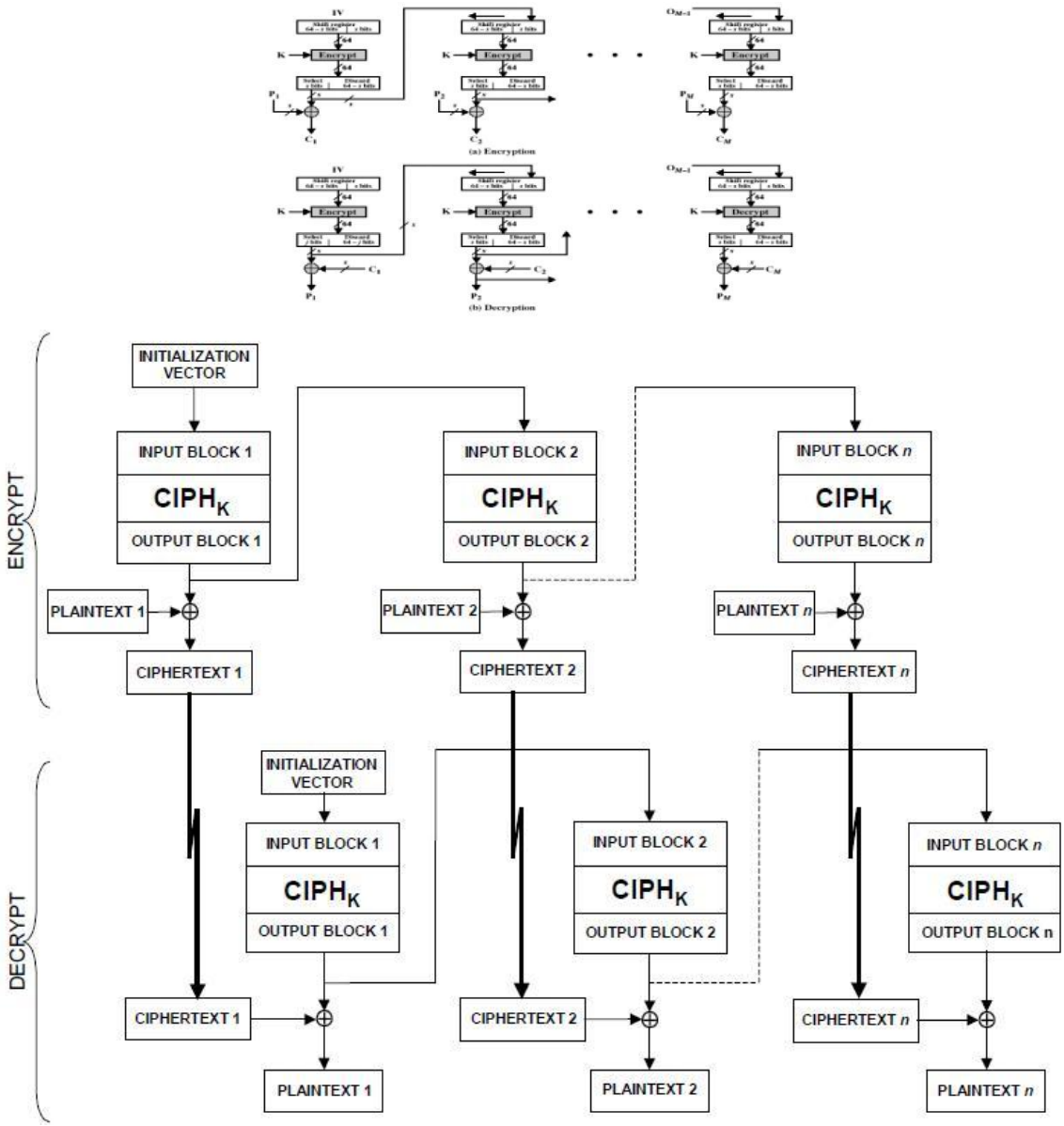
```
Key 2b7e151628aed2a6abf7158809cf4f3c
IV 000102030405060708090a0b0c0d0e0f
Block #1
```

```
Plaintext 6bc1bee22e409f96e93d7e117393172a
Input Block 6bc0bce12a459991e134741a7f9e1925
Output Block 7649abac8119b246cee98e9b12e9197d
Ciphertext 7649abac8119b246cee98e9b12e9197d
Block #2
Plaintext ae2d8a571e03ac9c9eb76fac45af8e51
Input Block d86421fb9f1a1eda505ee1375746972c
Output Block 5086cb9b507219ee95db113a917678b2
Ciphertext 5086cb9b507219ee95db113a917678b2
Block #3
Plaintext 30c81c46a35ce411e5fbc1191a0a52ef
Input Block 604ed7ddf32efdff7020d0238b7c2a5d
Output Block 73bed6b8e3c1743b7116e69e22229516
Ciphertext 73bed6b8e3c1743b7116e69e22229516
Block #4
Plaintext f69f2445df4f9b17ad2b417be66c3710
Input Block 8521f2fd3c8eef2cdc3da7e5c44ea206
Output Block 3ff1caa1681fac09120eca307586e1a7
Ciphertext 3ff1caa1681fac09120eca307586e1a7
```

F.2.2 CBC-AES128.Decrypt

```
Key 2b7e151628aed2a6abf7158809cf4f3c
IV 000102030405060708090a0b0c0d0e0f
Block #1
Ciphertext 7649abac8119b246cee98e9b12e9197d
Input Block 7649abac8119b246cee98e9b12e9197d
Output Block 6bc0bce12a459991e134741a7f9e1925
Plaintext 6bc1bee22e409f96e93d7e117393172a
Block #2
Ciphertext 5086cb9b507219ee95db113a917678b2
Input Block 5086cb9b507219ee95db113a917678b2
Output Block d86421fb9f1a1eda505ee1375746972c
Plaintext ae2d8a571e03ac9c9eb76fac45af8e51
Block #3
Ciphertext 73bed6b8e3c1743b7116e69e22229516
Input Block 73bed6b8e3c1743b7116e69e22229516
Output Block 604ed7ddf32efdff7020d0238b7c2a5d
Plaintext 30c81c46a35ce411e5fbc1191a0a52ef
Block #4
Ciphertext 3ff1caa1681fac09120eca307586e1a7
Input Block 3ff1caa1681fac09120eca307586e1a7
Output Block 8521f2fd3c8eef2cdc3da7e5c44ea206
Plaintext f69f2445df4f9b17ad2b417be66c3710
```

- OFB (Output Feed Back)
  Model ini mempunyai struktur yang sama dengan model EFB, tetapi input dari algoritma adalah output dari algoritma DES sebelumnya. Model ini dapat diimplementasikan pada aplikasi transmisi yang berorientasi pada stream yang berada pada chanel yang sibuk atau padat (satelit, dsb).



Example :

F.4.1 OFB-AES128.Encrypt

Key 2b7e151628aed2a6abf7158809cf4f3c

IV 000102030405060708090a0b0c0d0e0f

Block #1

Input Block 000102030405060708090a0b0c0d0e0f

Output Block 50fe67cc996d32b6da0937e99bafec60

Plaintext 6bc1bee22e409f96e93d7e117393172a

Ciphertext 3b3fd92eb72dad20333449f8e83cfb4a

Block #2

Input Block 50fe67cc996d32b6da0937e99bafec60

53

Output Block d9a4dada0892239f6b8b3d7680e15674

Plaintext ae2d8a571e03ac9c9eb76fac45af8e51

Ciphertext 7789508d16918f03f53c52dac54ed825

Block #3

Input Block d9a4dada0892239f6b8b3d7680e15674

Output Block a78819583f0308e7a6bf36b1386abf23

Plaintext 30c81c46a35ce411e5fbc1191a0a52ef

Ciphertext 9740051e9c5fecf64344f7a82260edcc

Block #4

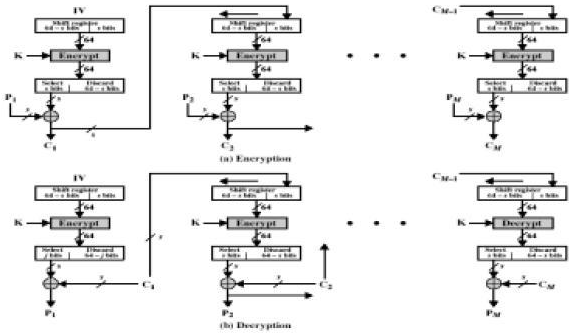
Input Block a78819583f0308e7a6bf36b1386abf23

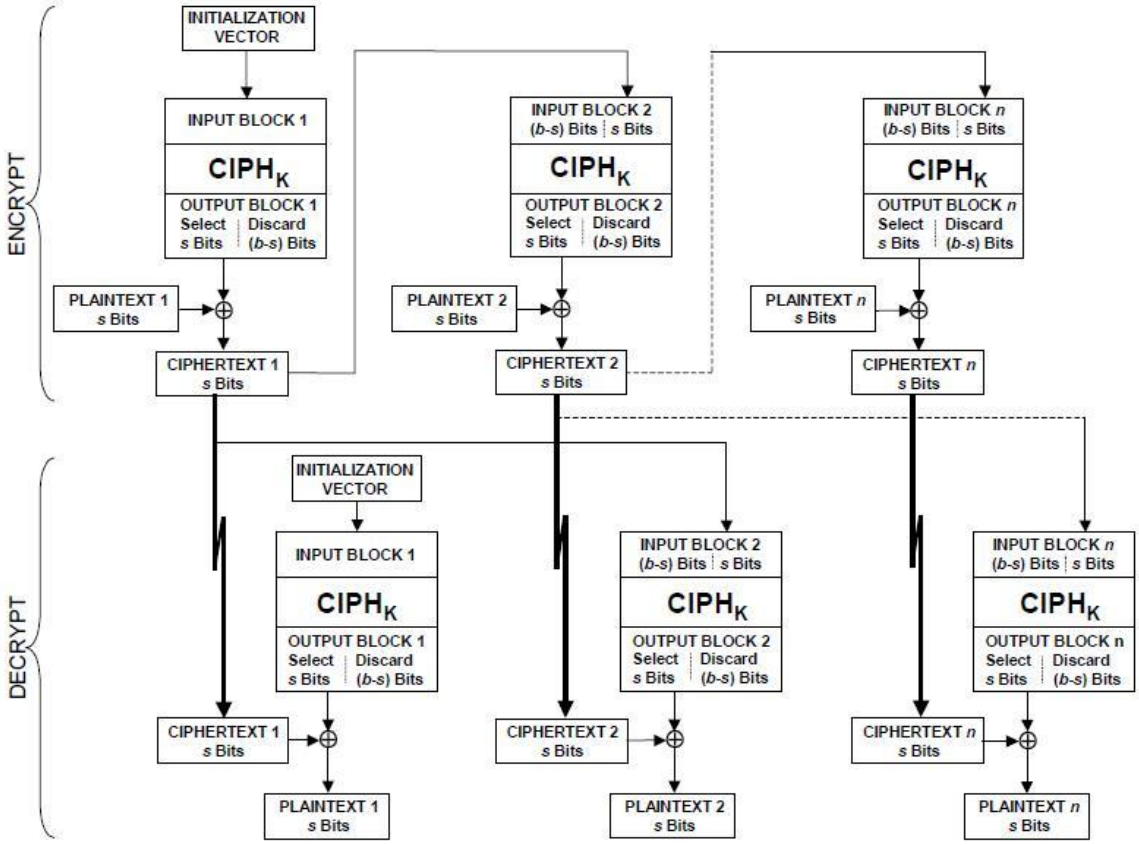
Output Block c6d3416d29165c6fcb8e51a227ba994e

Plaintext f69f2445df4f9b17ad2b417be66c3710

```
Ciphertext 304c6528f659c77866a510d9c1d6ae5e
F.4.2 OFB-AES128.Decrypt
Key 2b7e151628aed2a6abf7158809cf4f3c
IV 000102030405060708090a0b0c0d0e0f
Block #1
Input Block 000102030405060708090a0b0c0d0e0f
Output Block 50fe67cc996d32b6da0937e99bafec60
Ciphertext 3b3fd92eb72dad20333449f8e83cfb4a
Plaintext 6bc1bee22e409f96e93d7e117393172a
Block #2
Input Block 50fe67cc996d32b6da0937e99bafec60
Output Block d9a4dada0892239f6b8b3d7680e15674
Ciphertext 7789508d16918f03f53c52dac54ed825
Plaintext ae2d8a571e03ac9c9eb76fac45af8e51
Block #3
Input Block d9a4dada0892239f6b8b3d7680e15674
Output Block a78819583f0308e7a6bf36b1386abf23
Ciphertext 9740051e9c5fecf64344f7a82260edcc
Plaintext 30c81c46a35ce411e5fbc1191a0a52ef
Block #4
Input Block a78819583f0308e7a6bf36b1386abf23
Output Block c6d3416d29165c6fcb8e51a227ba994e
Ciphertext 304c6528f659c77866a510d9c1d6ae5e
Plaintext f69f2445df4f9b17ad2b417be66c3710
```

- CFB (Cipher Feed Back)
  Model ini tidak perlu menunggu pembagian data menjadi beberapa blok yang berukuran 64 bit, sehingga dapat diimplementasikan secara real time. Model ini menggunakan metode enkripsi untuk beberapa karakter dari plaintext. Karakter-karakter pada plaintext dapat dienkripsi dan ditransmisikan secara langsung dengan menggunakan stream chiper yang berorientasi karakter. Model ini sering diimplementasikan di dalam aplikasi transmisi yang berorientasi pada stream atau juga pada aplikasi otentifikasi.





Example :

F.3.1 CFB1-AES128.Encrypt

Key 2b7e151628aed2a6abf7158809cf4f3c
IV 000102030405060708090a0b0c0d0e0f
Segment #1
Input Block 000102030405060708090a0b0c0d0e0f
Output Block 50fe67cc996d32b6da0937e99bafec60
Plaintext 0
Ciphertext 0
Segment #2
Input Block 00020406080a0c0e10121416181a1c1e
Output Block 19cf576c7596e702f298b35666955c79
Plaintext 1
Ciphertext 1
Segment #3
Input Block 0004080c1014181c2024282c3034383d
Output Block 59e17759acd02b801fa321ea059e331f
Plaintext 1
Ciphertext 1
Segment #4
Input Block 000810182028303840485058606870 7b
Output Block 71f415b0cc109e8b0faa14ab740c22f4
Plaintext 0
Ciphertext 0
Segment #5
Input Block 00102030405060708090a0b0c0d0e0f6
Output Block 3fb76d3d1048179964597a0f64d5adad
Plaintext 1
Ciphertext 1
Segment #6
Input Block 0020406080a0c0e10121416181a1c1ed
Output Block 4c943b4bac54ab974e3e52326d29aaa1
Plaintext 0
Ciphertext 0
Segment #7

Input Block 004080c1014181c2024282c3034383da
Output Block c94da41eb3d3acf1993a512ab1e8203f
Plaintext 1
Ciphertext 0
Segment #8
Input Block 00810182028303840485058606870 7b4
Output Block e07f5e98778f75dbb2691c3f582c3953
Plaintext 1
Ciphertext 0
Segment #9
Input Block 010203040506070 8090a0b0c0d0e0f68
Output Block 02ef5fc8961efcce8568bc0731262dc7
Plaintext 1
Ciphertext 1
Segment #10
Input Block 020406080a0c0e10121416181a1c1ed1
Output Block 9f5a30367065efbe914b53698c8716b7
Plaintext 1
Ciphertext 0
Segment #11
Input Block 04080c1014181c2024282c3034383da2
Output Block d018cfb81d0580edbff955ed74d382db
Plaintext 0
Ciphertext 1
Segment #12
Input Block 08101820283038404850586068707b45
Output Block 81272ab351e08e0b695b94b8164d86f4
Plaintext 0
Ciphertext 1
Segment #13
Input Block 1020304050607080 90a0b0c0d0e0f68b
Output Block 094d33f856483d3fa01ba94f7e5ab3e7
Plaintext 0
Ciphertext 0
Segment #14
Input Block 20406080a0c0e10121416181a1c1ed16
Output Block 609900ad61923c8c102cd8d0d7947a2c
Plaintext 0
Ciphertext 0
Segment #15
Input Block 4080c1014181c2024282c3034383da2c
Output Block 9e5a154de966ab4db9c88b22a398134e
Plaintext 0
Ciphertext 1
Segment #16
Input Block 810182028303840485058606870 7b459
Output Block 7fe16252b338bc4de3725c4156dfed20
Plaintext 1
Ciphertext 1

F.3.2 CFB1-AES128.Decrypt

Key 2b7e151628aed2a6abf7158809cf4f3c
IV 000102030405060708090a0b0c0d0e0f
Segment #1
Input Block 000102030405060708090a0b0c0d0e0f
Output Block 50fe67cc996d32b6da0937e99bafec60
Ciphertext 0
Plaintext 0
Segment #2
Input Block 00020406080a0c0e10121416181a1c1e
Output Block 19cf576c7596e702f298b35666955c79
Ciphertext 1
Plaintext 1
Segment #3
Input Block 0004080c1014181c2024282c3034383d

```
Output Block 59e17759acd02b801fa321ea059e331f
Ciphertext 1
Plaintext 1
Segment #4
Input Block 000810182028303840485058606870 7b
Output Block 71f415b0cc109e8b0faa14ab740c22f4
Ciphertext 0
Plaintext 0
Segment #5
Input Block 00102030405060708090a0b0c0d0e0f6
Output Block 3fb76d3d1048179964597a0f64d5adad
Ciphertext 1
Plaintext 1
Segment #6
Input Block 0020406080a0c0e10121416181a1c1ed
Output Block 4c943b4bac54ab974e3e52326d29aaa1
Ciphertext 0
Plaintext 0
Segment #7
Input Block 004080c1014181c2024282c3034383da
Output Block c94da41eb3d3acf1993a512ab1e8203f
Ciphertext 0
Plaintext 1
Segment #8
Input Block 008101820283038404850586068707b4
Output Block e07f5e98778f75dbb2691c3f582c3953
Ciphertext 0
Plaintext 1
Segment #9
Input Block 0102030405060708090a0b0c0d0e0f68
Output Block 02ef5fc8961efcce8568bc0731262dc7
Ciphertext 1
Plaintext 1
Segment #10
Input Block 020406080a0c0e10121416181a1c1ed1
Output Block 9f5a30367065efbe914b53698c8716b7
Ciphertext 0
Plaintext 1
Segment #11
Input Block 04080c1014181c2024282c3034383da2
Output Block d018cfb81d0580edbff955ed74d382db
Ciphertext 1
Plaintext 0
Segment #12
Input Block 081018202830384048505860687 07b45
Output Block 81272ab351e08e0b695b94b8164d86f4
Ciphertext 1
Plaintext 0
Segment #13
Input Block 10203040506070809 0a0b0c0d0e0f68b
Output Block 094d33f856483d3fa01ba94f7e5ab3e7
Ciphertext 0
Plaintext 0
Segment #14
Input Block 20406080a0c0e10121416181a1c1ed16
Output Block 609900ad61923c8c102cd8d0d7947a2c
Ciphertext 0
Plaintext 0
Segment #15
Input Block 4080c1014181c2024282c3034383da2c
Output Block 9e5a154de966ab4db9c88b22a398134e
Ciphertext 1
Plaintext 0
Segment #16
```

```
Input Block 810182028303840485058606870 7b459
33
Output Block 7fe16252b338bc4de3725c4156dfed20
Ciphertext 1
Plaintext 1
```

Stream cipher

Stream Cipher adalah algoritma enkripsi yang mengenkripsi data persatuan data, seperti bit, byte, nible atau per 5 bit. Setiap mengenkripsi satu satuan data digunakan kunci yang merupakan hasil pembangkitan dari kunci sebelumnya.
Contoh : RC4 (Rivest Cipher 4), A5, A2, SEAL, WAKE, Cellular Automaton

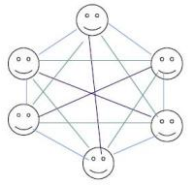
Algoritma Simetrik (kunci privat)

Algoritma simetris *(symmetric algorithm)* adalah suatu algoritma dimana kunci enkripsi yang digunakan sama dengan kunci dekripsi sehingga algoritma ini disebut juga sebagai *single-key algorithm.*
Contoh : DES, IDEA, AES
Kelebihan algoritma simetrik :

- Kecepatan operasi lebih tinggi bila dibandingkan dengan algoritma asimetrik. Algoritma ini dirancang sehingga proses enkripsi/dekripsi membutuhkan waktu yang singkat.
- Ukuran kunci relatif lebih pendek.
- Karena kecepatannya yang cukup tinggi, maka dapat digunakan pada sistem *real-time*

Kelemahan algoritma simetrik :

- Untuk tiap pengiriman pesan dengan pengguna yang berbeda dibutuhkan kunci yang berbeda juga, sehingga akan terjadi kesulitan dalam manajemen kunci tersebut.
- Jumlah kunci meledak secara eksponensial: n (n-1)/2: (lihat ilustrasi / gambar di bawah)



Meledaknya jumlah kunci :

n (jumlah orang)	*n * (n-1)/2 ≈ n^2/2* (jumlah kunci)
10	50
100	5000
1000	500000
10000	50000000
100000	5000000000

Jika 1 kunci memiliki ukuran 1 kByte, maka untuk 100.000 orang dibutuhkan 5 TBytes untuk kunci saja

- Kunci harus dikirim melalui saluran yang aman. Kedua entitas yang berkomunikasi harus menjaga kerahasiaan kunci ini. Permasalahan dalam pengiriman kunci itu disebut *"key distribution problem"*
- Kunci harus sering diubah, mungkin pada setiap sesi komunikasi.

Algoritma Asimetrik (kunci public)

Algoritma asimetris *(asymmetric algorithm)* adalah suatu algoritma dimana kunci enkripsi yang digunakan tidak sama dengan kunci dekripsi. Pada algoritma ini menggunakan dua kunci yakni kunci publik *(public key)* dan kunci privat *(private key).* Bila *plaintext* dienkripsi dengan menggunakan kunci pribadi maka *ciphertext* yang dihasilkannya hanya bisa didekripsikan dengan menggunakan pasangan kunci pribadinya. Kunci publik disebarkan secara umum sedangkan kunci privat disimpan secara rahasia oleh si pengguna. Walau kunci publik telah diketahui namun akan sangat sukar mengetahui kunci privat yang digunakan.

Contohnya adalah Knapsack, RSA (Rivert-Shamir-Adelman), Diffie-Hellman, ECC (Elliptic Curve Cryptosystem), DSA (Digital Signature Algorithm).

Apabila Ahmad dan Bejo hendak bertukar berkomunikasi, maka:

1. Ahmad dan Bejo masing-masing membuat 2 buah kunci
    1. Ahmad membuat dua buah kunci, kunci-publik $K_{publik[Ahmad]}$ dan kunci-privat $K_{privat[Ahmad]}$
    2. Bejo membuat dua buah kunci, kunci-publik $K_{publik[Bejo]}$ dan kunci-privat $K_{privat[Bejo]}$
2. Mereka berkomunikasi dengan cara:
    1. Ahmad dan Bejo saling bertukar kunci-publik. Bejo mendapatkan $K_{publik[Ahmad]}$ dari Ahmad, dan Ahmad mendapatkan $K_{publik[Bejo]}$ dari Bejo.
    2. Ahmad mengenkripsi teks-terang P ke Bejo dengan fungsi $C = E(P, K_{publik[Bejo]})$
    3. Ahmad mengirim teks-sandi C ke Bejo
    4. Bejo menerima C dari Ahmad dan membuka teks-terang dengan fungsi $P = D(C, K_{privat[Bejo]})$

Hal yang sama terjadi apabila Bejo hendak mengirimkan pesan ke Ahmad

1. Bejo mengenkripsi teks-terang P ke Ahmad dengan fungsi $C = E(P, K_{publik[Ahmad]})$
2. Ahmad menerima C dari Bejo dan membuka teks-terang dengan fungsi $P = D(C, K_{privat[Ahmad]})$

Kelebihan algoritma asimetrik :

- Hanya *Private key* yang harus benar-benar rahasia/aman.
- Sangat jarang untuk perlu merubah *public key* dan *private key.*
- Masalah keamanan pada distribusi kunci dapat lebih baik
- Masalah manajemen kunci yang lebih baik karena jumlah kunci yang lebih sedikit
- Jumlah kunci yang lebih sedikit dibandingkan enkripsi dengan kunci privat

Kelemahan algoritma asimetrik :

- Kecepatan yang lebih rendah bila dibandingkan dengan algoritma simetris (Membutuhkan komputasi yang tinggi /membutuhkan waktu yang lebih lama)
- Untuk tingkat keamanan sama, kunci yang digunakan lebih panjang dibandingkan dengan algoritma simetris (Ukuran kunci lebih besar dari pada *symmetric encryption*)
- Tidak adanya jaminan bahwa *public key* benar-benar aman

Fungsi Hash

Fungsi *hash* ini sering juga disebut sebagai fungsi *hash* kriptografis, yaitu fungsi yang secara efisien mengubah *string input* dengan panjang berhingga menjadi *string output* dengan panjang tetap yang disebut nilai *hash*. Fungsi ini bersifat satu arah sehingga inputan yang telah dienkripsi tidak dapat dibalikkan atau didekripsikan. Contohnya adalah penggunaan MD5 untuk melindungi *password*, SHA
Perubahan satu bit saja akan mengubah keluaran hash secara drastis
Digunakan untuk menjamin integritas dan digital signature

Kripto System yang baik

Faktor penentu dari kekuatan system sandi adalah seberapa sanggup algoritma enkripsi atau deskripsinya memproduksi rangkaian kunci yang kompleks dan panjang tidak berulang

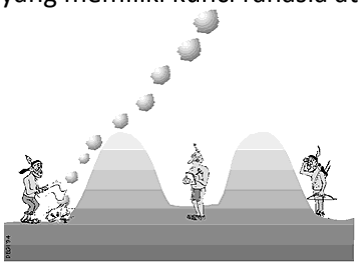
Kripto system yang baik harus memiliki karakteristik sbb :

- Keamanan sistem terletak pada kerahasiaan kunci dan bukan pada kerahasiaan algoritma yang digunakan.
- Cryptosystem yang baik memiliki ruang kunci (keyspace) yang besar.
- Cryptosystem yang baik akan menghasilkan ciphertext yang terlihat acak dalam seluruh tes statistik yang dilakukan terhadapnya.
- Cryptosystem yang baik mampu menahan seluruh serangan yang telah dikenal sebelumnya.

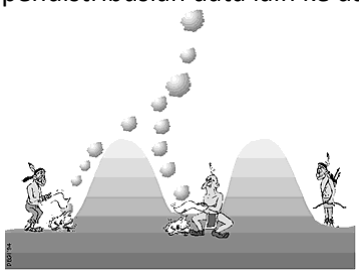
Kegunaan Kriptografi

Ada empat tujuan mendasar dari kriptografi yang juga merupakan aspek keamanan informasi, yaitu sebagai berikut.

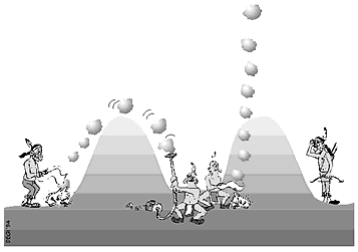
1. Kerahasiaan, adalah layanan yang digunakan untuk menjaga isi dari informasi dari siapapun kecuali yang memiliki kunci rahasia atau otoritas untuk membuka informasi yang telah disandikan.



2. Integritas Data, berhubungan dengan penjagaan dari perubahan data secara tidak sah. Untuk dapat menjaga integritas data, suatu system harus memiliki kemampuan untuk mendeteksi manipulasi data yang dilakukan pihak-pihak yang tidak berhak, antara lain penyisipan, penghapusan, dan pendistribusian data lain ke dalam data yang asli.



3. Autentifikasi, berhubungan dengan identifikasi, baik secara kesatuan system maupun informasi itu sendiri. Dua pihak yang saling berkomuniasi harus saling memperkenalkan diri. Informasi yang dikirimkan harus diautentikasi keasliannya, isi datanya, waktu pengiriman dan lain sebagainya.



4. Non-repudiasi, merupakan usaha untuk mencegah terjadinya penyangkalan terhadap pengiriman/terciptanya suatu informasi oleh yang mengirimkan/ membuat.