

How DES Works in Detail

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

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

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

L = 0000 0001 0010 0011 0100 0101 0110 0111

R = 1000 1001 1010 1011 1100 1101 1110 1111

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

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

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

K = 00010011 00110100 01010111 01111001 10011011 10111100 11011111 11110001

The DES algorithm uses the following steps:

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

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

PC-1

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

Example: From the original 64-bit key

K = 00010011 00110100 01010111 01111001 10011011 10111100 11011111
11110001

we get the 56-bit permutation

K₊ = 1111000 0110011 0010101 0101111 0101010 1011001 1001111 0001111

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

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

C₀ = 1111000 0110011 0010101 0101111

D₀ = 0101010 1011001 1001111 0001111

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

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

14	2
15	2
16	1

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

Example: From original pair pair C_0 and D_0 we obtain:

$C_0 = 1111000011001100101010101111$
 $D_0 = 0101010101100110011110001111$

$C_1 = 1110000110011001010101011111$
 $D_1 = 1010101011001100111100011110$

$C_2 = 1100001100110010101010111111$
 $D_2 = 0101010110011001111000111101$

$C_3 = 0000110011001010101011111111$
 $D_3 = 0101011001100111100011110101$

$C_4 = 0011001100101010101111111100$
 $D_4 = 0101100110011110001111010101$

$C_5 = 1100110010101010111111110000$
 $D_5 = 0110011001111000111101010101$

$C_6 = 0011001010101011111111000011$
 $D_6 = 1001100111100011110101010101$

$C_7 = 1100101010101111111100001100$
 $D_7 = 0110011110001111010101010110$

$C_8 = 0010101010111111110000110011$
 $D_8 = 1001111000111101010101011001$

$C_9 = 0101010101111111100001100110$
 $D_9 = 0011110001111010101010110011$

$C_{10} = 0101010111111110000110011001$
 $D_{10} = 1111000111101010101011001100$

$C_{11} = 0101011111111000011001100101$
 $D_{11} = 1100011110101010101100110011$

$C_{12} = 0101111111100001100110010101$
 $D_{12} = 0001111010101010110011001111$

$C_{13} = 0111111110000110011001010101$
 $D_{13} = 0111101010101011001100111100$

$C_{14} = 1111111000011001100101010101$
 $D_{14} = 1110101010101100110011110001$

$C_{15} = 1111100001100110010101010111$
 $D_{15} = 1010101010110011001111000111$

$C_{16} = 1111000011001100101010101111$
 $D_{16} = 0101010101100110011110001111$

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

PC-2

14	17	11	24	1	5
3	28	15	6	21	10
23	19	12	4	26	8
16	7	27	20	13	2
41	52	31	37	47	55
30	40	51	45	33	48
44	49	39	56	34	53
46	42	50	36	29	32

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

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

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

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

For the other keys we have

$K_2 = 011110\ 011010\ 111011\ 011001\ 110110\ 111100\ 100111\ 100101$
 $K_3 = 010101\ 011111\ 110010\ 001010\ 010000\ 101100\ 111110\ 011001$
 $K_4 = 011100\ 101010\ 110111\ 010110\ 110110\ 110011\ 010100\ 011101$
 $K_5 = 011111\ 001110\ 110000\ 000111\ 111010\ 110101\ 001110\ 101000$
 $K_6 = 011000\ 111010\ 010100\ 111110\ 010100\ 000111\ 101100\ 101111$
 $K_7 = 111011\ 001000\ 010010\ 110111\ 111101\ 100001\ 100010\ 111100$
 $K_8 = 111101\ 111000\ 101000\ 111010\ 110000\ 010011\ 101111\ 111011$
 $K_9 = 111000\ 001101\ 101111\ 101011\ 111011\ 011110\ 011110\ 000001$
 $K_{10} = 101100\ 011111\ 001101\ 000111\ 101110\ 100100\ 011001\ 001111$
 $K_{11} = 001000\ 010101\ 111111\ 010011\ 110111\ 101101\ 001110\ 000110$
 $K_{12} = 011101\ 010111\ 000111\ 110101\ 100101\ 000110\ 011111\ 101001$
 $K_{13} = 100101\ 111100\ 010111\ 010001\ 111110\ 101011\ 101001\ 000001$
 $K_{14} = 010111\ 110100\ 001110\ 110111\ 111100\ 101110\ 011100\ 111010$
 $K_{15} = 101111\ 111001\ 000110\ 001101\ 001111\ 010011\ 111100\ 001010$
 $K_{16} = 110010\ 110011\ 110110\ 001011\ 000011\ 100001\ 011111\ 110101$

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

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

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

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

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

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

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

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

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

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

L₀ = 1100 1100 0000 0000 1100 1100 1111 1111
R₀ = 1111 0000 1010 1010 1111 0000 1010 1010

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

$$\begin{aligned} L_n &= R_{n-1} \\ R_n &= L_{n-1} + f(R_{n-1}, K_n) \end{aligned}$$

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

Example: For $n = 1$, we have

K₁ = 000110 110000 001011 101111 111111 000111 000001 110010
L₁ = **R₀** = 1111 0000 1010 1010 1111 0000 1010 1010
R₁ = **L₀** + **f(R₀, K₁)**

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

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

E BIT-SELECTION TABLE

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

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

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

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

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

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

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

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

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

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

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

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

We have not yet finished calculating the function f . To this point we have expanded \mathbf{R}_{n-1} from 32 bits to 48 bits, using the selection table, and XORed the result with the key \mathbf{K}_n . We now have 48 bits, or eight groups of six bits. We now do something strange with each group of six bits: we use them as addresses in tables called "**S boxes**". Each group of six bits will give us an address in a different **S** box. Located at that address will be a 4 bit number. This 4 bit number will replace the original 6 bits. The net result is that the eight groups of 6 bits are transformed into eight groups of 4 bits (the 4-bit outputs from the **S** boxes) for 32 bits total.

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

$$\mathbf{K}_n + \mathbf{E}(\mathbf{R}_{n-1}) = \mathbf{B}_1\mathbf{B}_2\mathbf{B}_3\mathbf{B}_4\mathbf{B}_5\mathbf{B}_6\mathbf{B}_7\mathbf{B}_8,$$

where each \mathbf{B}_i is a group of six bits. We now calculate

$$S_1(B_1)S_2(B_2)S_3(B_3)S_4(B_4)S_5(B_5)S_6(B_6)S_7(B_7)S_8(B_8)$$

where $S_i(B_i)$ refers to the output of the i -th **S** box.

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

S1																
Column Number																
Row No.	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	14	4	13	1	2	15	11	8	3	10	6	12	5	9	0	7
1	0	15	7	4	14	2	13	1	10	6	12	11	9	5	3	8
2	4	1	14	8	13	6	2	11	15	12	9	7	3	10	5	0
3	15	12	8	2	4	9	1	7	5	11	3	14	10	0	6	13

If S_1 is the function defined in this table and B is a block of 6 bits, then $S_1(B)$ is determined as follows: The first and last bits of B represent in base 2 a number in the decimal range 0 to 3 (or binary 00 to 11). Let that number be i . The middle 4 bits of B represent in base 2 a number in the decimal range 0 to 15 (binary 0000 to 1111). Let that number be j . Look up in the table the number in the i -th row and j -th column. It is a number in the range 0 to 15 and is uniquely represented by a 4 bit block. That block is the output $S_1(B)$ of S_1 for the input B . For example, for input block $B = 011011$ the first bit is "0" and the last bit "1" giving 01 as the row. This is row 1. The middle four bits are "1101". This is the binary equivalent of decimal 13, so the column is column number 13. In row 1, column 13 appears 5. This determines the output; 5 is binary 0101, so that the output is 0101. Hence $S_1(011011) = 0101$.

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

S1															
14	4	13	1	2	15	11	8	3	10	6	12	5	9	0	7
0	15	7	4	14	2	13	1	10	6	12	11	9	5	3	8
4	1	14	8	13	6	2	11	15	12	9	7	3	10	5	0
15	12	8	2	4	9	1	7	5	11	3	14	10	0	6	13

S2															
15	1	8	14	6	11	3	4	9	7	2	13	12	0	5	10
3	13	4	7	15	2	8	14	12	0	1	10	6	9	11	5
0	14	7	11	10	4	13	1	5	8	12	6	9	3	2	15
13	8	10	1	3	15	4	2	11	6	7	12	0	5	14	9

S3

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

S4

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

S5

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

S6

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

S7

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

S8

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

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

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

$$S_1(\mathbf{B}_1)S_2(\mathbf{B}_2)S_3(\mathbf{B}_3)S_4(\mathbf{B}_4)S_5(\mathbf{B}_5)S_6(\mathbf{B}_6)S_7(\mathbf{B}_7)S_8(\mathbf{B}_8) = 0101 \ 1100 \ 1000 \ 0010 \ 1011 \\ 0101 \ 1001 \ 0111$$

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

$$f = \mathbf{P}(S_1(\mathbf{B}_1)S_2(\mathbf{B}_2)...S_8(\mathbf{B}_8))$$

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

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

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

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

we get

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

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

$$\begin{aligned} &= 1100\ 1100\ 0000\ 0000\ 1100\ 1100\ 1111\ 1111 \\ &+ 0010\ 0011\ 0100\ 1010\ 1010\ 1001\ 1011\ 1011 \\ &= 1110\ 1111\ 0100\ 1010\ 0110\ 0101\ 0100\ 0100 \end{aligned}$$

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

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

and apply a final permutation **IP⁻¹** as defined by the following table:

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

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

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

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

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

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

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

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

which in hexadecimal format is

$$85E813540F0AB405.$$

This is the encrypted form of $\mathbf{M} = 0123456789ABCDEF$: namely, $\mathbf{C} = 85E813540F0AB405$.

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

DES Modes of Operation

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

Cracking DES

Before DES was adopted as a national standard, during the period NBS was soliciting comments on the proposed algorithm, the creators of public key cryptography, Martin Hellman and Whitfield Diffie, registered some objections to the use of DES as an encryption algorithm. Hellman wrote: "Whit Diffie and I have become concerned that

the proposed data encryption standard, while probably secure against commercial assault, may be extremely vulnerable to attack by an intelligence organization" (letter to NBS, October 22, 1975).

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

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

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

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

Triple-DES

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

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

