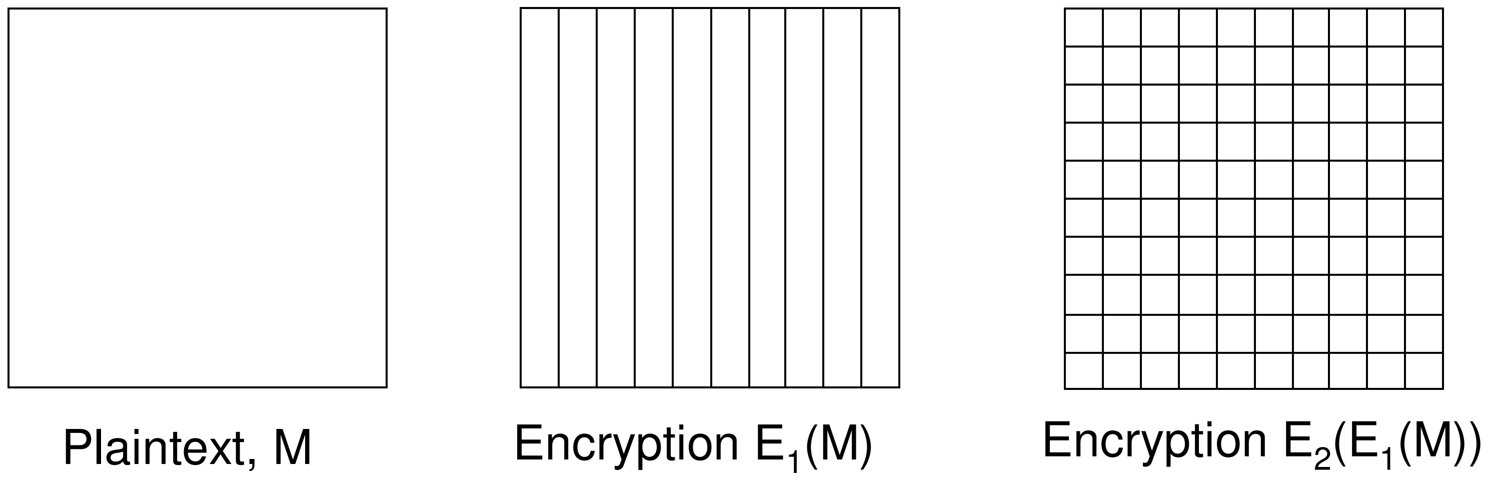
Module 2 – Advanced Symmetric Ciphers

Dr. Natarajan Meghanathan
Professor of Computer Science
Jackson State University
E-mail: natarajan.meghanathan@jsums.edu

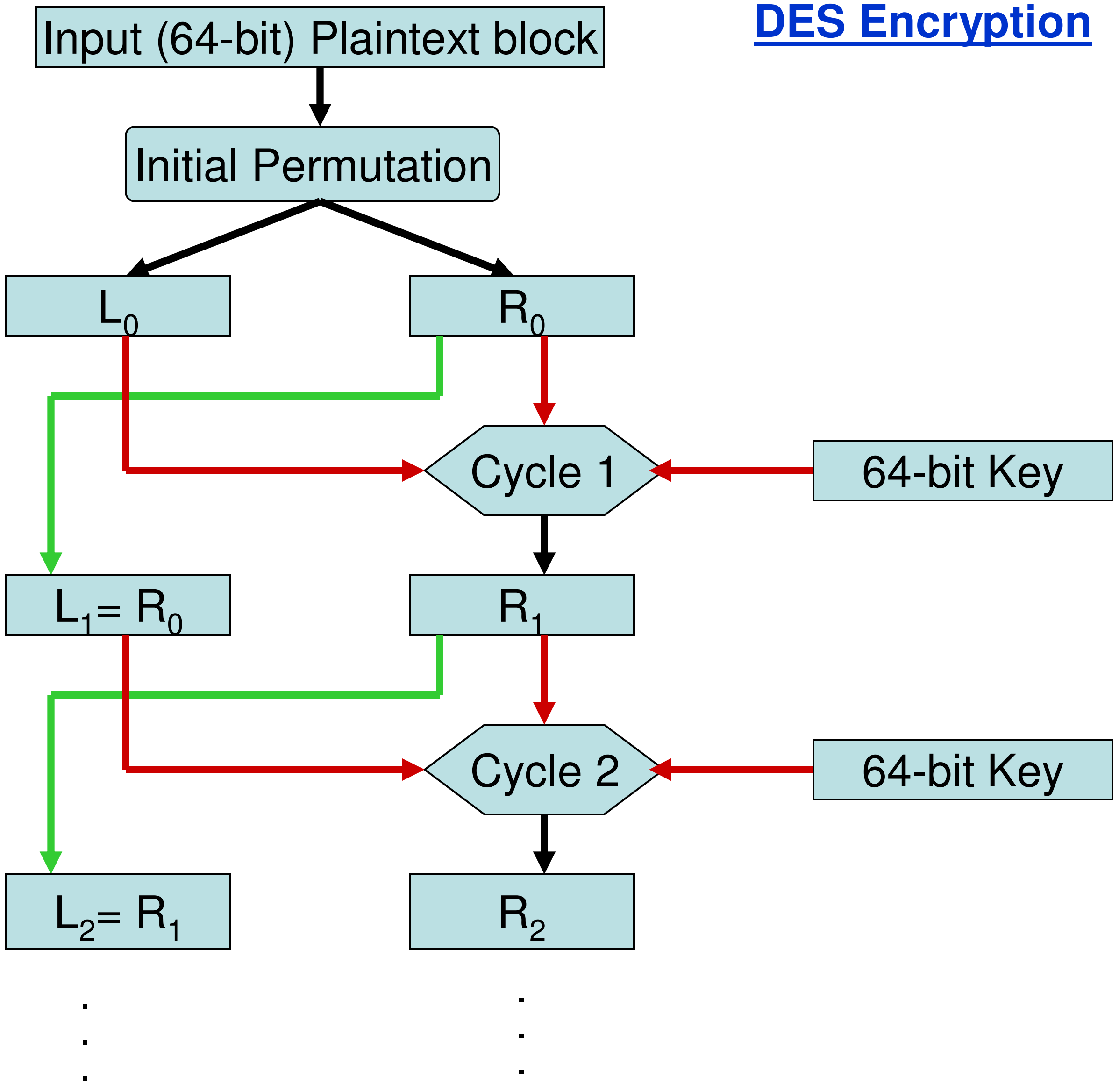
Data Encryption Standard (DES)

- The DES algorithm was developed by IBM based on the Lucifer algorithm it has been using before.
- The DES is a careful and complex combination of the two fundamental building blocks of encryption: substitution and transposition.
- The algorithm derives its strength from repeated application of these two techniques (16 cycles), one on top of the other.
- <u>Product cipher:</u> Two complementary ciphers can be made more secure by being applied together alternately



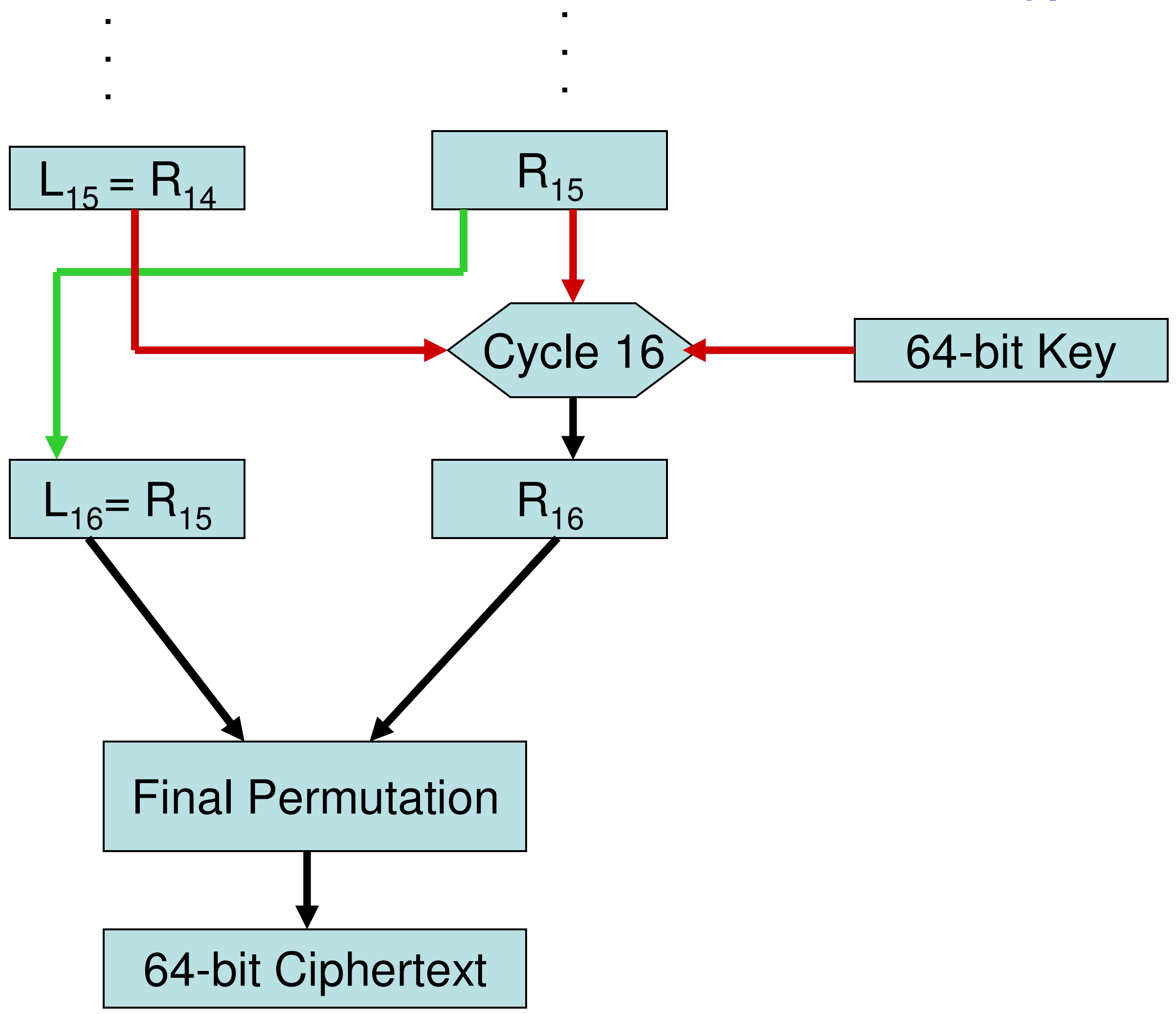
Cycles of Substitution and Permutation

DES Encryption

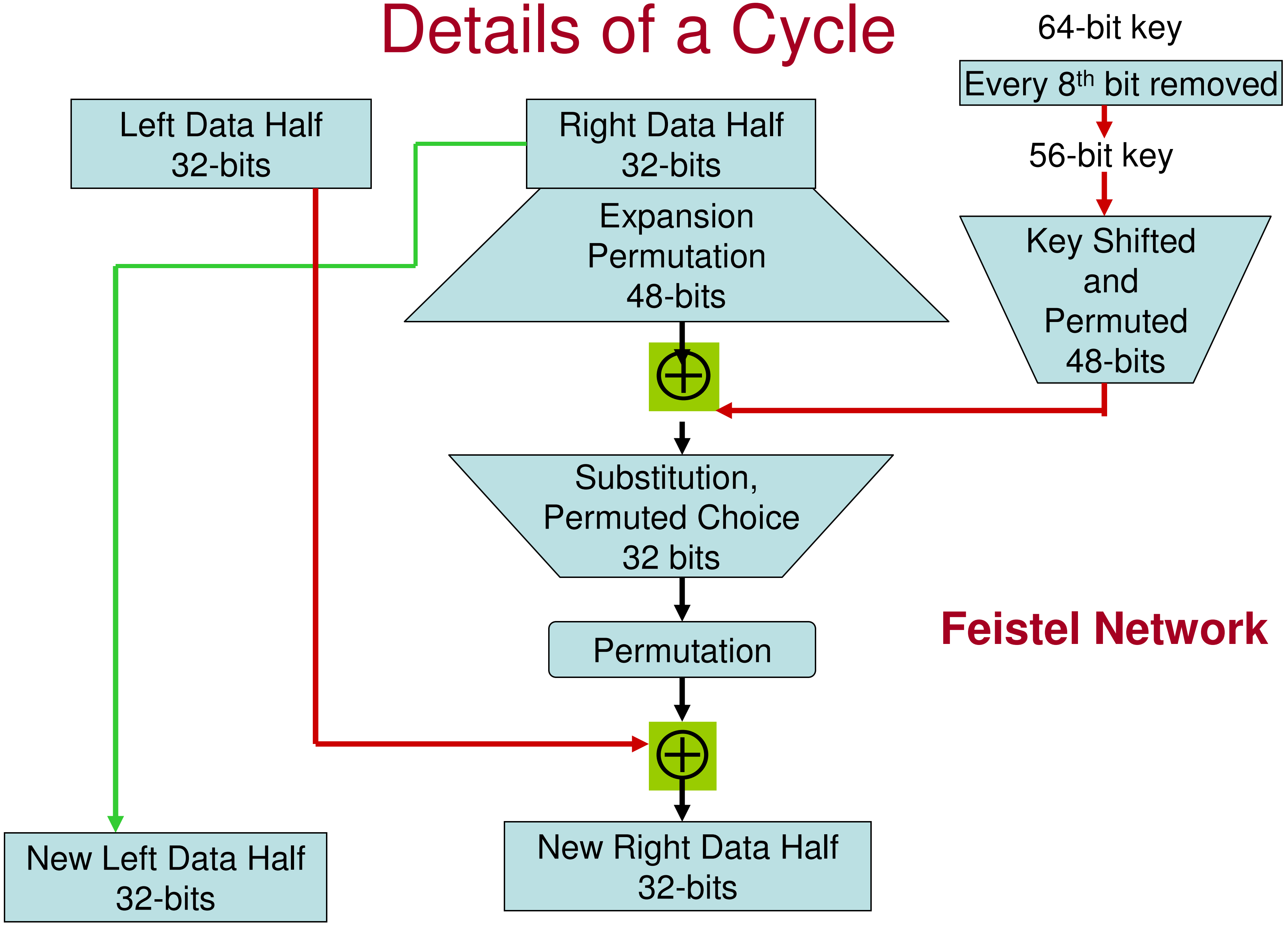


Cycles of Substitution and Permutation

DES Encryption

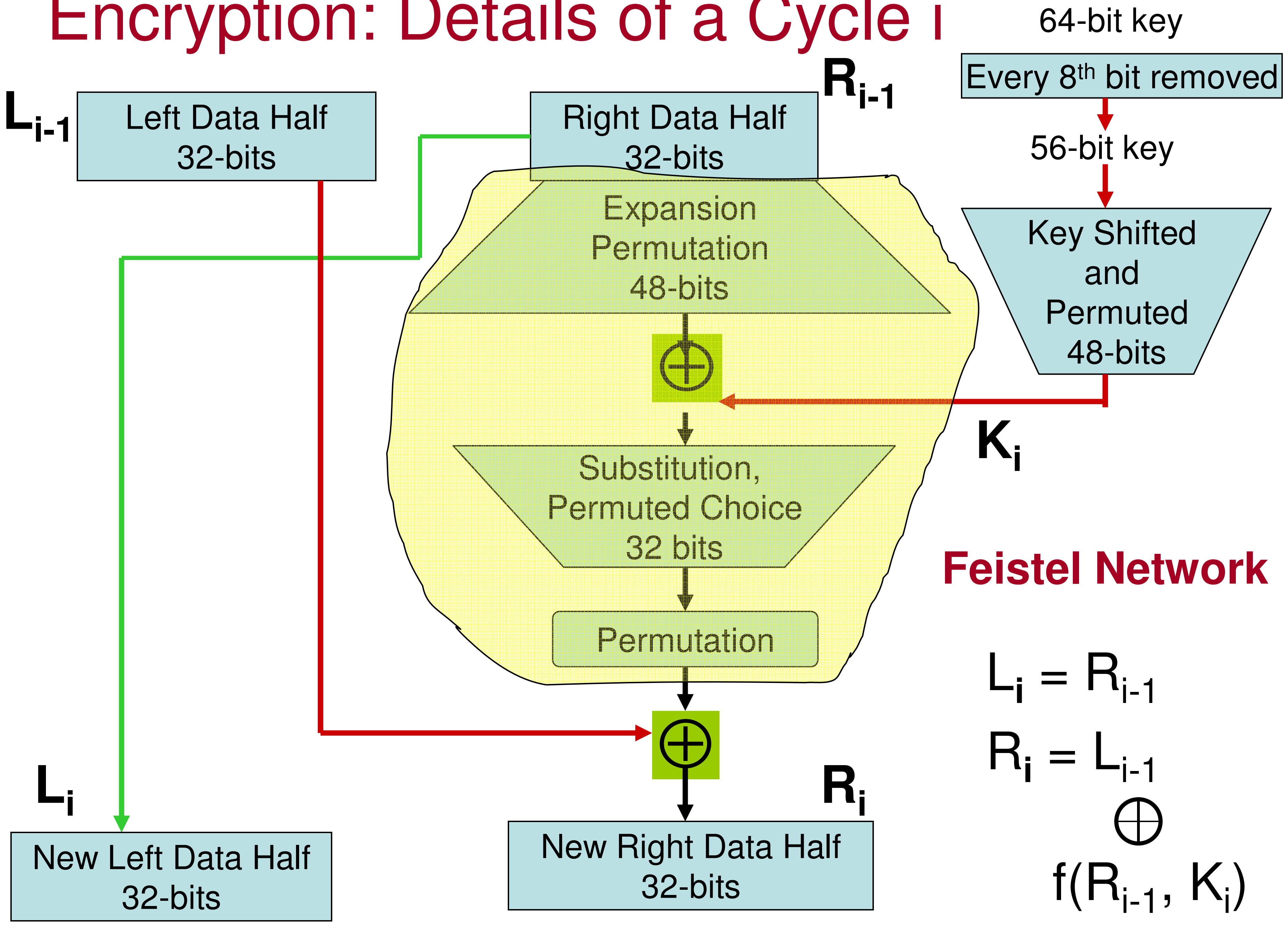


Details of a Cycle


64-bit key
Every 8th bit removed
56-bit key
Key Shifted and Permuted 48-bits
Left Data Half 32-bits
Right Data Half 32-bits
Expansion Permutation 48-bits
Substitution, Permuted Choice 32 bits
Permutation
New Left Data Half 32-bits
New Right Data Half 32-bits

Feistel Network

Encryption: Details of a Cycle i


64-bit key
Every 8th bit removed
56-bit key
Key Shifted and Permuted 48-bits
K_i
L_{i-1}
Left Data Half 32-bits
R_{i-1}
Right Data Half 32-bits
Expansion Permutation 48-bits
Substitution, Permuted Choice 32 bits
Permutation
L_i
New Left Data Half 32-bits
R_i
New Right Data Half 32-bits

Feistel Network

$$L_i = R_{i-1}$$

$$R_i = L_{i-1} \oplus f(R_{i-1}, K_i)$$

Initial and Final 64-bit Permutations

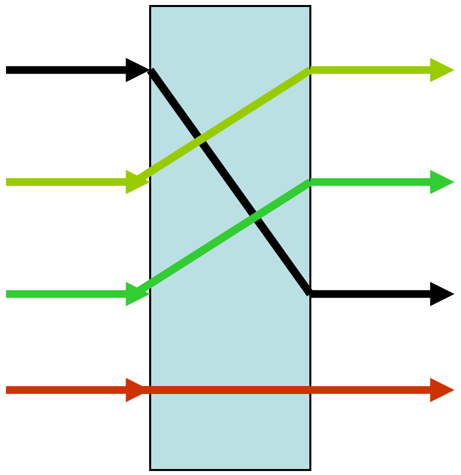
Bit	Goes to Position							
1 – 8	40	8	48	16	56	24	64	32
9 – 16	39	7	47	15	55	23	63	31
17 – 24	38	6	46	14	54	22	62	30
25 – 32	37	5	45	13	53	21	61	29
33 – 40	36	4	44	12	52	20	60	28
41 – 48	35	3	43	11	51	19	59	27
49 – 56	34	2	42	10	50	18	58	26
57 – 64	33	1	41	9	49	17	57	25

Initial Permutation

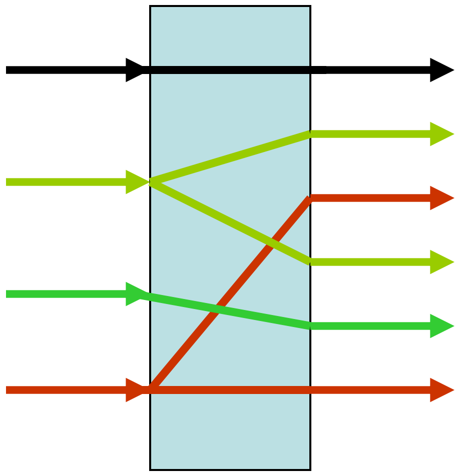
Bit	Goes to Position							
1 – 8	58	50	42	34	26	18	10	2
9 – 16	60	52	44	36	28	20	12	4
17 – 24	62	54	46	38	30	22	14	6
25 – 32	64	56	48	40	32	24	16	8
33 – 40	57	49	41	33	25	17	9	1
41 – 48	59	51	43	35	27	19	11	3
49 – 56	61	53	45	37	29	21	13	5
57 – 64	63	55	47	39	31	23	15	7

Final Permutation (reverse of the initial)

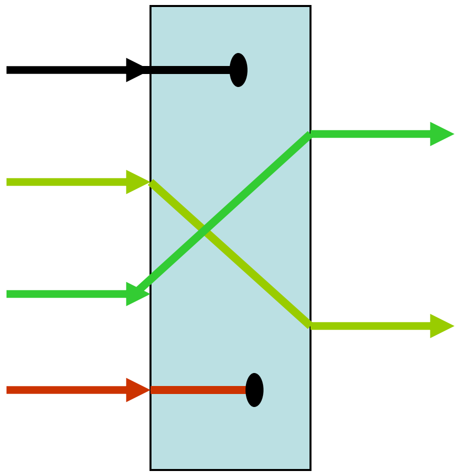
Types of Permutations



Permutation



Expansion
Permutation



Permuted Choice

Various Permutations

Bit	1	2	3	4	5	6	7	8
Moves to Position	2, 48	3	4	5,7	6,8	9	10	11,13
Bit	9	10	11	12	13	14	15	16
Moves to Position	12,14	15	16	17,19	18,20	21	22	23,25
Bit	17	18	19	20	21	22	23	24
Moves to Position	24,26	27	28	29,31	30,32	33	34	35,37
Bit	25	26	27	28	29	30	31	32
Moves to Position	36,38	39	40	41,43	42,44	45	46	47,1

Expansion Permutation: 32-bits to 48-bits

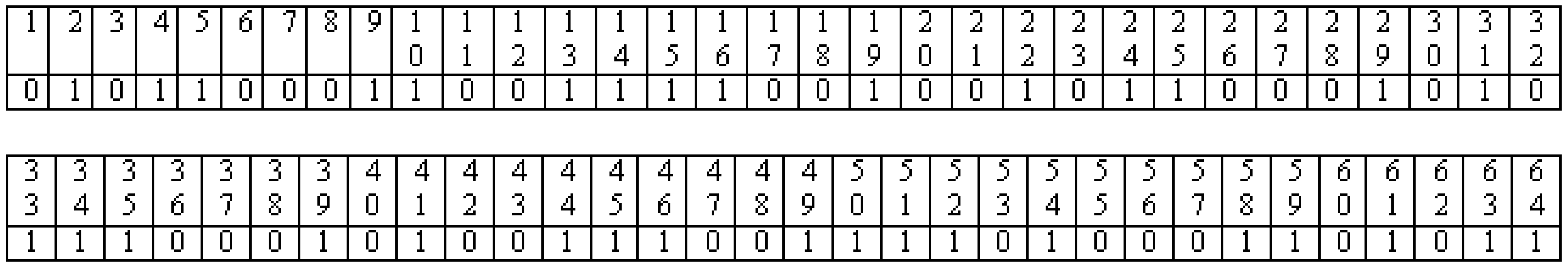
Bit	Goes to Position							
1 – 8	9	17	23	31	13	28	2	18
9 – 16	24	16	30	6	26	20	10	1
17 – 24	8	14	25	3	4	29	11	19
25 – 32	32	12	22	7	5	27	15	21

Permutation Box, P-Box

Key Transformation

- The 64-bit key immediately becomes a 56-bit key by deletion of every eighth bit.
- At each step in the cycle, the key is split into two 28-bit halves, the halves are shifted left by a specified number of digits, the halves are then merged together again, and 48 of these 56 bits are permuted to be fed to the cycle

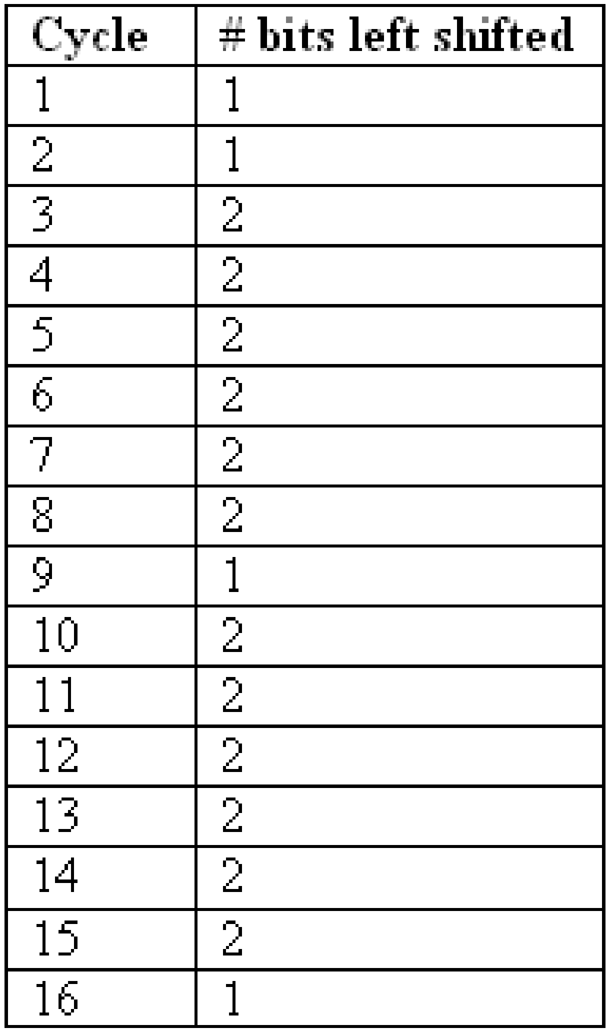
64-bit Key



1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32
0	1	0	1	1	0	0	0	1	1	0	0	1	1	1	1	0	0	1	0	0	1	0	1	1	0	0	0	1	0	1	0

33	34	35	36	37	38	39	40	41	42	43	44	45	46	47	48	49	50	51	52	53	54	55	56	57	58	59	60	61	62	63	64
1	1	1	0	0	0	1	0	1	0	0	1	1	1	0	0	1	1	1	1	0	1	0	0	0	1	1	0	1	0	1	1

Key Shift Table



Cycle	# bits left shifted
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

Key Shift and Permutation (Cycle 1)

After every 8th bit removed and split into two 28-bit halves

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32
0	1	0	1	1	0	0	0	1	1	0	0	1	1	1	1	0	0	1	0	0	1	0	1	1	0	0	0	1	0	1	0

33	34	35	36	37	38	39	40	41	42	43	44	45	46	47	48	49	50	51	52	53	54	55	56	57	58	59	60	61	62	63	64
1	1	1	0	0	0	1	0	1	0	0	1	1	1	0	0	1	1	1	1	0	1	0	0	0	1	1	0	1	0	1	1

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28
0	1	0	1	1	0	0	1	1	0	0	1	1	1	0	0	1	0	0	1	0	1	0	0	0	1	0	1

29	30	31	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47	48	49	50	51	52	53	54	55	56
1	1	1	0	0	0	1	1	0	0	1	1	1	0	1	1	1	1	0	1	0	0	1	1	0	1	0	1

Left shifting the two 28-bit halves by 1 bit and putting them together (Cycle 1)

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28
1	0	1	1	0	0	1	1	0	0	1	1	1	0	0	1	0	0	1	0	1	0	0	0	1	0	1	0

29	30	31	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47	48	49	50	51	52	53	54	55	56
1	1	0	0	0	1	1	0	0	1	1	1	0	1	1	1	1	0	1	0	0	1	1	0	1	0	1	1

Permuted to 48-bits

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24
0	0	1	0	1	0	1	0	0	0	1	0	0	1	1	1	0	1	1	1	1	0	1	0

25	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47	48
0	0	0	0	1	1	1	1	1	1	0	0	1	0	1	1	1	1	0	1	1	0	1	0

Choice Permutation to Select 48 Key Bits

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

56 bits to 48 bits

Substitution Boxes S-Boxes

		Column															
Box	**Row**	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**
S_1																	
	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
S_2																	
	0	15	1	8	14	6	11	3	4	9	7	2	13	12	0	5	10
	1	3	13	4	7	15	2	8	14	12	0	1	10	6	9	11	5
	2	0	14	7	11	10	4	13	1	5	8	12	6	9	3	2	15
	3	13	8	10	1	3	15	4	2	11	6	7	12	0	5	14	9
S_3																	
	0	10	0	9	14	6	3	15	5	1	13	12	7	11	4	2	8
	1	13	7	0	9	3	4	6	10	2	8	5	14	12	11	15	1
	2	13	6	4	9	8	15	3	0	11	1	2	12	5	10	14	7
	3	1	10	13	0	6	9	8	7	4	15	14	3	11	5	2	12
S_4																	
	0	7	13	14	3	0	6	9	10	1	2	8	5	11	12	4	15
	1	13	8	11	5	6	15	0	3	4	7	2	12	1	10	14	9
	2	10	6	9	0	12	11	7	13	15	1	3	14	5	2	8	4
	3	3	15	0	6	10	1	13	8	9	4	5	11	12	7	2	14
S_5																	
	0	2	12	4	1	7	10	11	6	8	5	3	15	13	0	14	9
	1	14	11	2	12	4	7	13	1	5	0	15	10	3	9	8	6
	2	4	2	1	11	10	13	7	8	15	9	12	5	6	3	0	14
	3	11	8	12	7	1	14	2	13	6	15	0	9	10	4	5	3
S_6																	
	0	12	1	10	15	9	2	6	8	0	13	3	4	14	7	5	11
	1	10	15	4	2	7	12	9	5	6	1	13	14	0	11	3	8
	2	9	14	15	5	2	8	12	3	7	0	4	10	1	13	11	6
	3	4	3	2	12	9	5	15	10	11	14	1	7	6	0	8	13
S_7																	
	0	4	11	2	14	15	0	8	13	3	12	9	7	5	10	6	1
	1	13	0	11	7	4	9	1	10	14	3	5	12	2	15	8	6
	2	1	4	11	13	12	3	7	14	10	15	6	8	0	5	9	2
	3	6	11	13	8	1	4	10	7	9	5	0	15	14	2	3	12
S_8																	
	0	13	2	8	4	6	15	11	1	10	9	3	14	5	0	12	7
	1	1	15	13	8	10	3	7	4	12	5	6	11	0	14	9	2
	2	7	11	4	1	9	12	14	2	0	6	10	13	15	3	5	8
	3	2	1	14	7	4	10	8	13	15	12	9	0	3	5	6	11

S-Boxes

- An S-box is a permuted choice function by which six bits are replaced by four bits.
- The 48-bit input is divided into eight 6-bit blocks, identified as $B_1B_2 \ldots B_8$; block B_j is operated on by S-box S_j.
- The S-Boxes are substitutions based on a table of 4 rows and 16 columns.
- Suppose that block B_j is the six bits $b_1b_2b_3b_4b_5b_6$.
- Bits b_1 and b_6 taken together form a two-bit binary number $b_1 b_6$ having a decimal value from 0 to 3. Call this value $\underline{r}$.
- Bits b_2, b_3, b_4 and b_5 taken together form a 4-bit binary number $b_2b_3b_4 b_5$, having a decimal value from 0 to 15. Call this value $\underline{c}$.
- The substitutions from the S-boxes transform each 6-bit block B_j into 4-bit result shown in row *r* and column *c* of S-box S_j.

Example to Illustrate Use of S-Boxes

48-bit Input

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24
0	1	1	1	0	1	0	1	0	0	1	0	1	1	0	1	0	1	0	1	1	0	1	1

25	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47	48
0	0	1	1	1	0	1	1	0	1	0	0	1	1	0	1	0	0	1	1	1	0	0	0

Substitution with Permutation

	6-bit input	Row value	Column value	S-box result	4-bit output
S-box S1	011101	1 (01)	14 (1110)	3	0011
S-box S2	010010	0 (00)	9 (1001)	7	0111
S-box S3	110101	3 (11)	10 (1010)	14	1110
S-box S4	011011	1 (01)	13 (1101)	10	1010
S-box S5	001110	0 (00)	7 (0111)	6	0110
S-box S6	110100	2 (10)	10 (1010)	4	0100
S-box S7	110100	2 (10)	10 (1010)	6	0110
S-box S8	111000	2 (10)	12 (1100)	15	1111

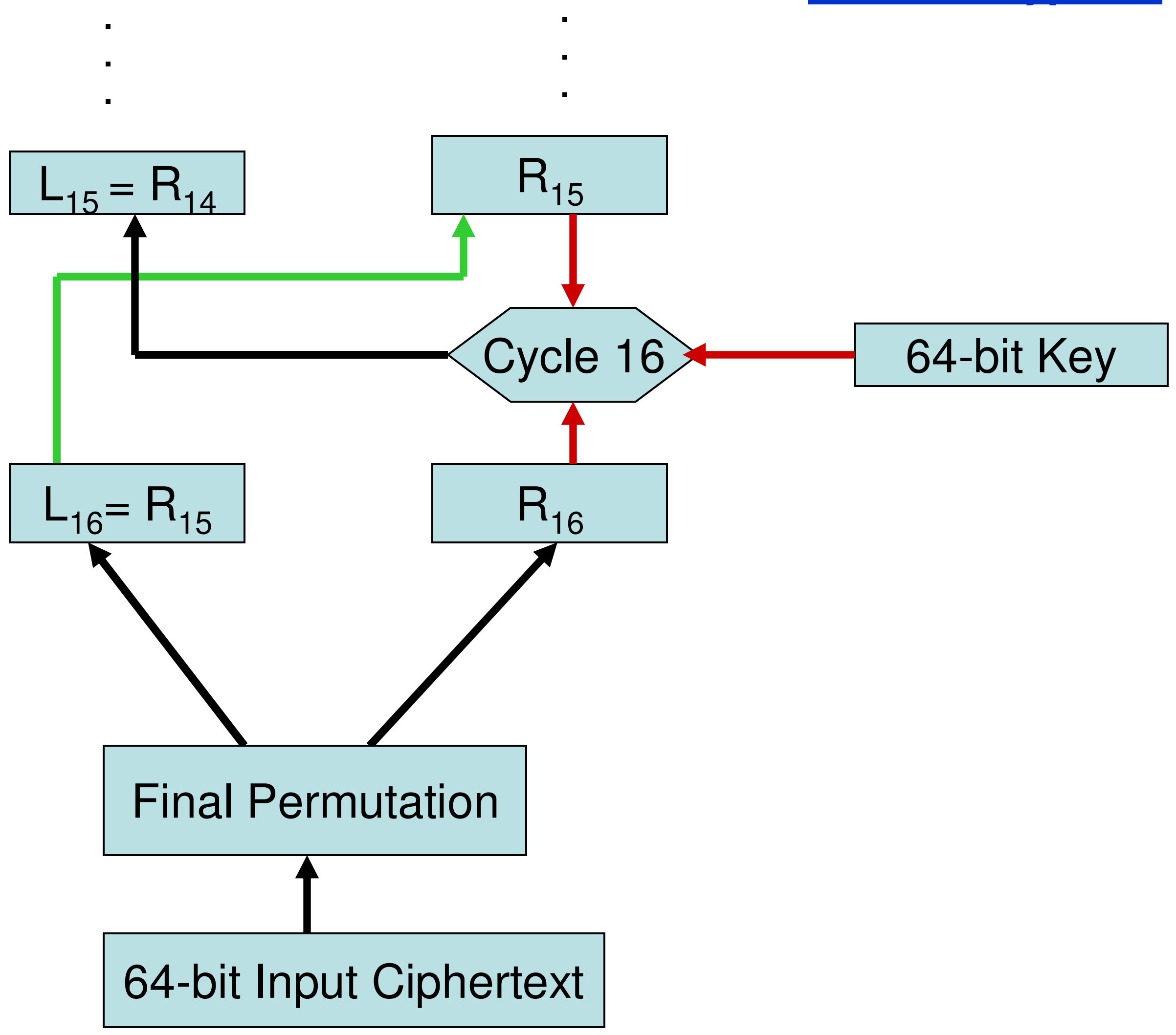
32-bit output

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
0	0	1	1	0	1	1	1	1	1	1	0	1	0	1	0

17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32
0	1	1	0	0	1	0	0	0	1	0	0	1	1	1	1

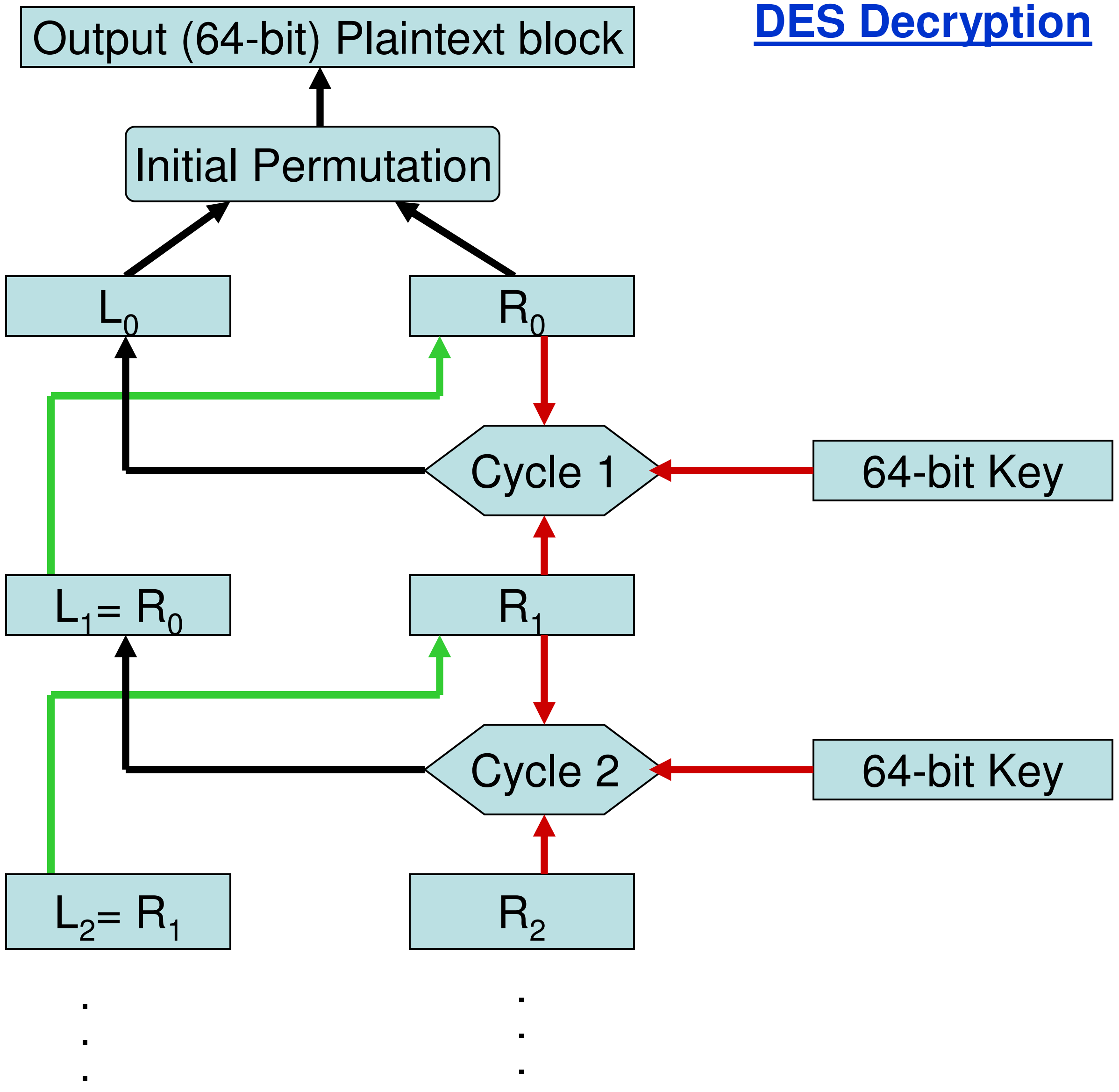
Cycles of Substitution and Permutation

DES Decryption



Cycles of Substitution and Permutation

DES Decryption



Decryption of the DES

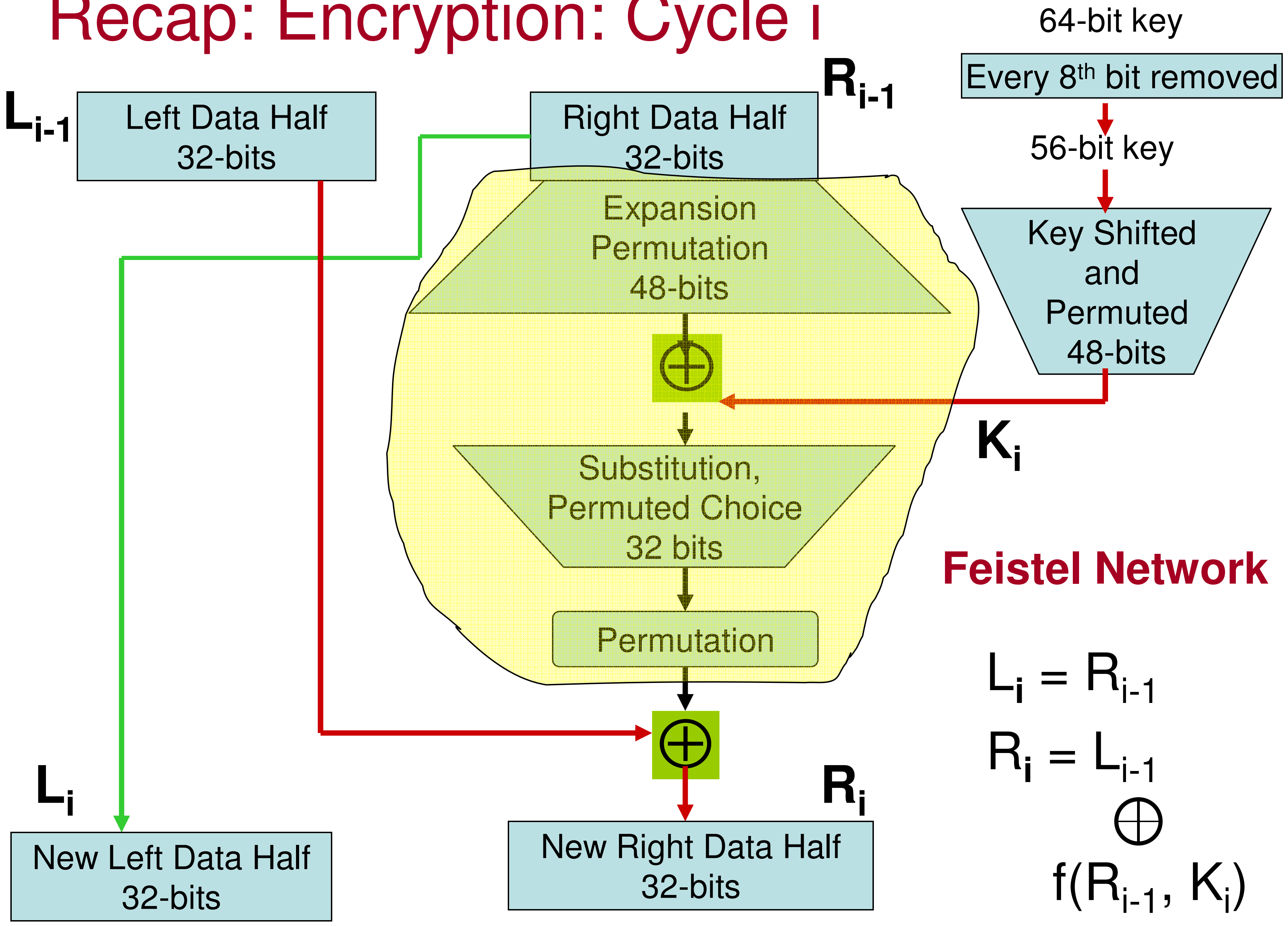
- The same DES algorithm is used for both encryption and decryption
  - Note that cycle *i* derives from cycle (*i*-1) in the following manner:

$$L_i = R_{i-1}$$
$$R_i = L_{i-1} \oplus f(R_{i-1}, K_i)$$

$$R_{i-1} = L_i$$
$$L_{i-1} = R_i \oplus f(L_i, K_i)$$

  - The same function *f* is used forward to encrypt or backward to decrypt.
  - The only change is that the keys must be taken in the reverse order (K_{16}, K_{15}, …, K_3, K_2, K_1)
  - The number of positions shifted for the keys should be considered from the bottom of the table and not top-down.

Recap: Encryption: Cycle i

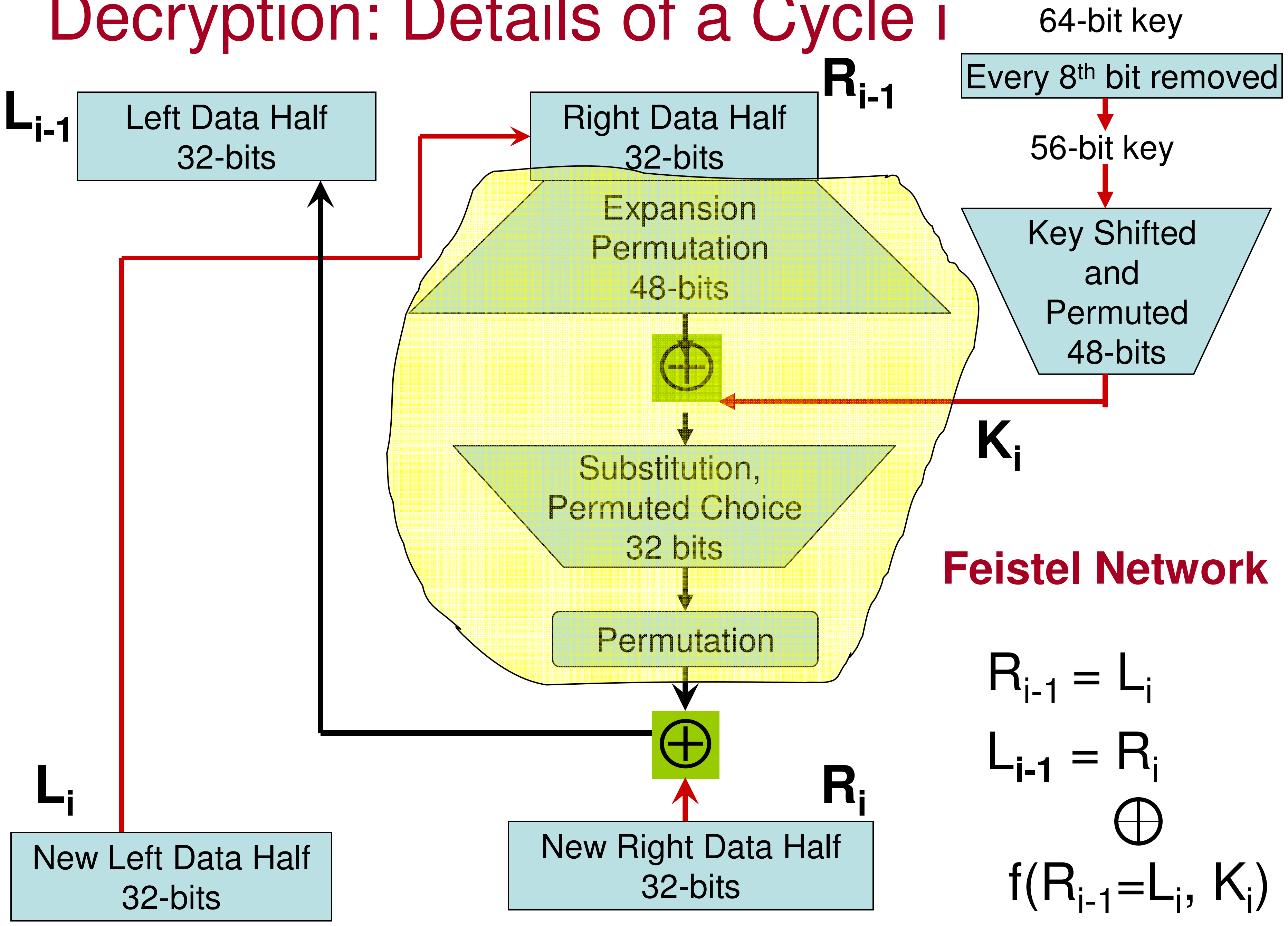

64-bit key
Every 8th bit removed
56-bit key
Key Shifted and Permuted 48-bits
L_{i-1}
Left Data Half 32-bits
R_{i-1}
Right Data Half 32-bits
Expansion Permutation 48-bits
K_i
Substitution, Permuted Choice 32 bits
Permutation
L_i
New Left Data Half 32-bits
R_i
New Right Data Half 32-bits

Feistel Network

$$L_i = R_{i-1}$$

$$R_i = L_{i-1} \oplus f(R_{i-1}, K_i)$$

Decryption: Details of a Cycle i



64-bit key
Every 8th bit removed
56-bit key
Key Shifted and Permuted 48-bits
L_{i-1}
Left Data Half 32-bits
Right Data Half 32-bits
R_{i-1}
Expansion Permutation 48-bits
$\oplus$
K_i
Substitution, Permuted Choice 32 bits
Permutation
$\oplus$
L_i
R_i
New Left Data Half 32-bits
New Right Data Half 32-bits

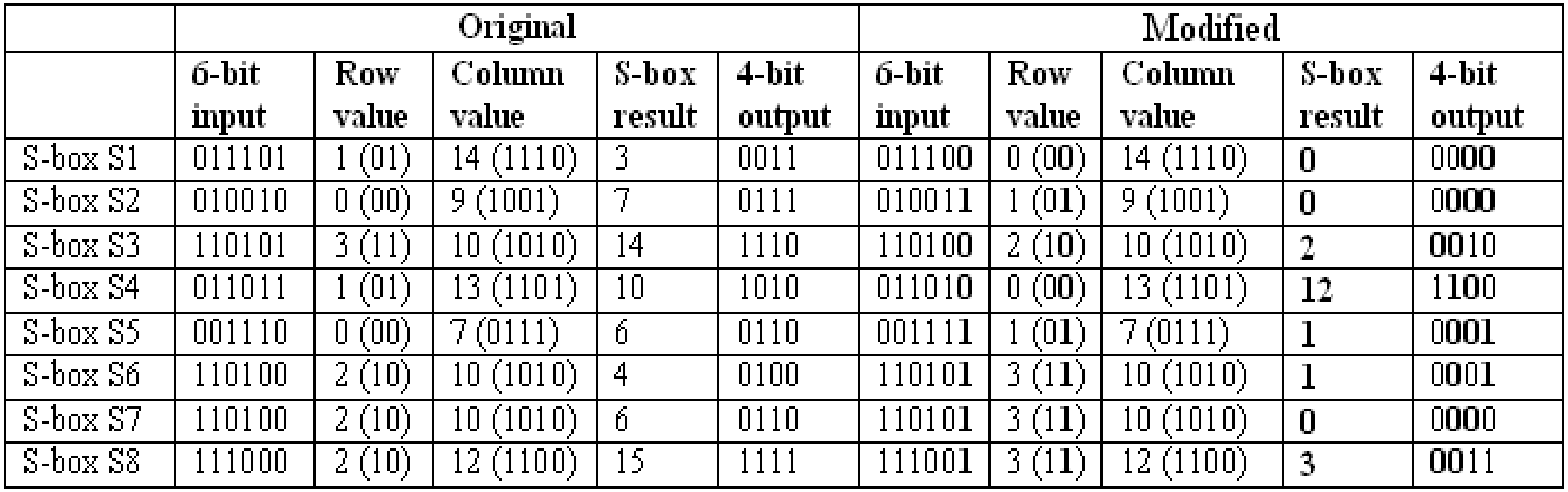
Feistel Network

$$R_{i-1} = L_i$$

$$L_{i-1} = R_i \oplus f(R_{i-1}=L_i, K_i)$$

Property of S-Boxes

- Changing one bit in the input of an S-box results in changing at least two output bits; that is the S-boxes diffuse their information well throughout their outputs.
- Example:



	Original					Modified				
	6-bit input	Row value	Column value	S-box result	4-bit output	6-bit input	Row value	Column value	S-box result	4-bit output
S-box S1	011101	1 (01)	14 (1110)	3	0011	011100	0 (00)	14 (1110)	0	0000
S-box S2	010010	0 (00)	9 (1001)	7	0111	010011	1 (01)	9 (1001)	0	0000
S-box S3	110101	3 (11)	10 (1010)	14	1110	110100	2 (10)	10 (1010)	2	0010
S-box S4	011011	1 (01)	13 (1101)	10	1010	011010	0 (00)	13 (1101)	12	1100
S-box S5	001110	0 (00)	7 (0111)	6	0110	001111	1 (01)	7 (0111)	1	0001
S-box S6	110100	2 (10)	10 (1010)	4	0100	110101	3 (11)	10 (1010)	1	0001
S-box S7	110100	2 (10)	10 (1010)	6	0110	110101	3 (11)	10 (1010)	0	0000
S-box S8	111000	2 (10)	12 (1100)	15	1111	111001	3 (11)	12 (1100)	3	0011

- No S-box is a linear or affine function (a function with a constant slope and may have a non-zero value when the independent variables are zero) of its input; the four output bits cannot be expressed as a system of linear equations of the six input bits

Weaknesses of the DES

- Complements:
  - For a plaintext p and key k, if C = DES(p, k), then $\neg$C = DES($\neg$p, $\neg$k) where $\neg$x is the ones complement (all 0s changed to 1s and vice-versa) of binary string x.
- Weak keys:
  - If the value being shifted in each cycle is all 0s or all 1s, then the key used for encryption is the same across all cycles. Such keys are called weak keys, because for a weak key K, C = DES(P, K) and P = DES(C, K) when proceeded in the forward direction from round 1 to round 16.
  - Keys with all 0s or all 1s or all 0s in the first half and 1s in the second half or vice-versa are also considered weak keys.
- Semi-weak keys:
  - There exists some key pairs k_1 and k_2 such C=DES(p, k_1) = DES(p, k_2). This implies that a message encrypted with key k_1 could be decrypted with key k_2.

Proof of DES Complement Property(1)

- For any plaintext P and key K, if C = DES(P, K), then C' = DES(P', K') where P', K' and C' are the ones complement of P, K and C respectively.

Truth Table

A	B	A'	B'	$A' \oplus B'$	$A \oplus B$	$(A \oplus B)'$	$A' \oplus B$	$A \oplus B'$
0	0	1	1	0	0	1	1	1
0	1	1	0	1	1	0	0	0
1	0	0	1	1	1	0	0	0
1	1	0	0	0	0	1	1	1

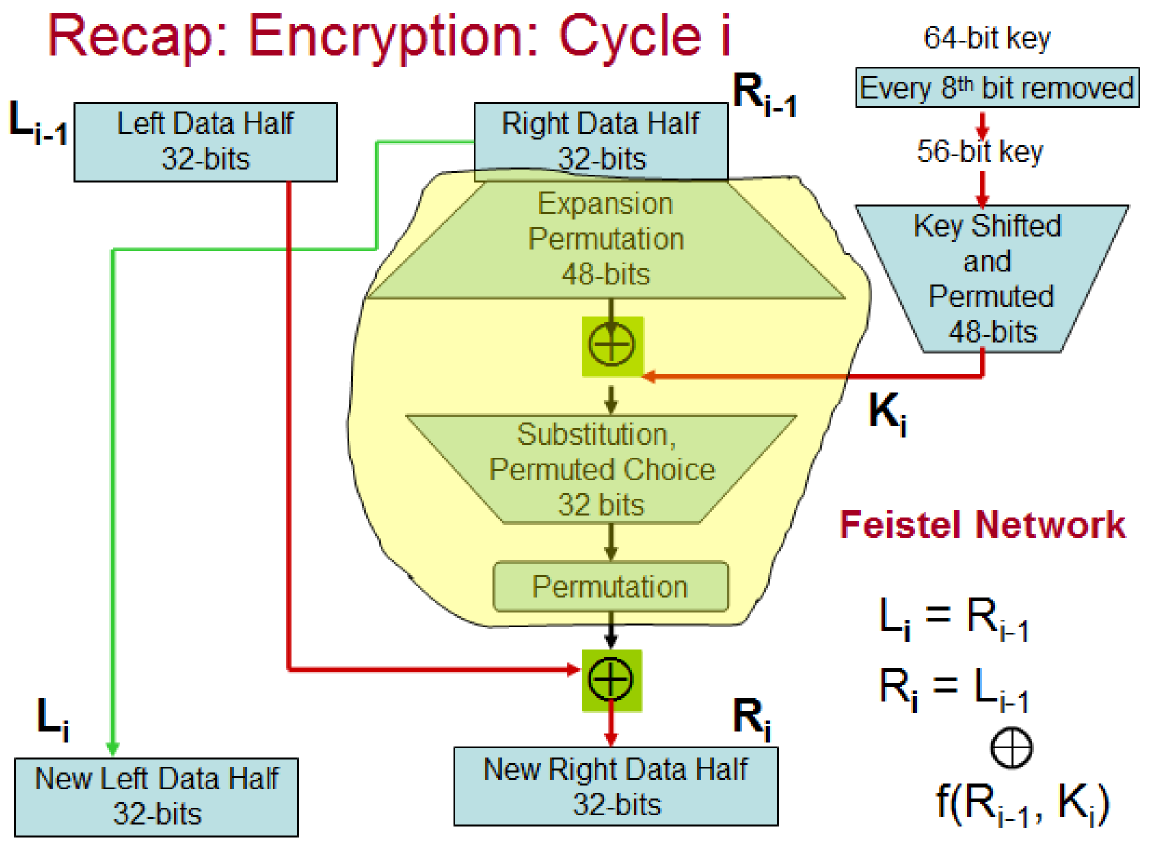
From the truth table,

$A \oplus B = A' \oplus B'$

$(A \oplus B)' = A' \oplus B$

$(A \oplus B)' = A \oplus B'$

Proof of DES Complement Property(2)



Recap: Encryption: Cycle i
L_{i-1}
Left Data Half 32-bits
R_{i-1}
Right Data Half 32-bits
Expansion Permutation 48-bits
Substitution, Permuted Choice 32 bits
Permutation
64-bit key
Every 8th bit removed
56-bit key
Key Shifted and Permuted 48-bits
K_i
Feistel Network
L_i
New Left Data Half 32-bits
R_i
New Right Data Half 32-bits

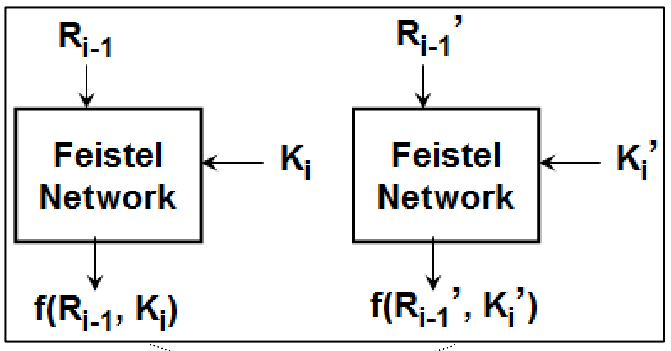
$L_i = R_{i-1}$

$R_i = L_{i-1} \oplus f(R_{i-1}, K_i)$

$R_{i-1}' \oplus K_i' == R_{i-1} \oplus K_i$

Expanded from 32 bits to 48 bits

Reduced from 64 bits to 48 bits



$$f(R_{i-1}, K_i) = f(R_{i-1}', K_i')$$

From the truth table,

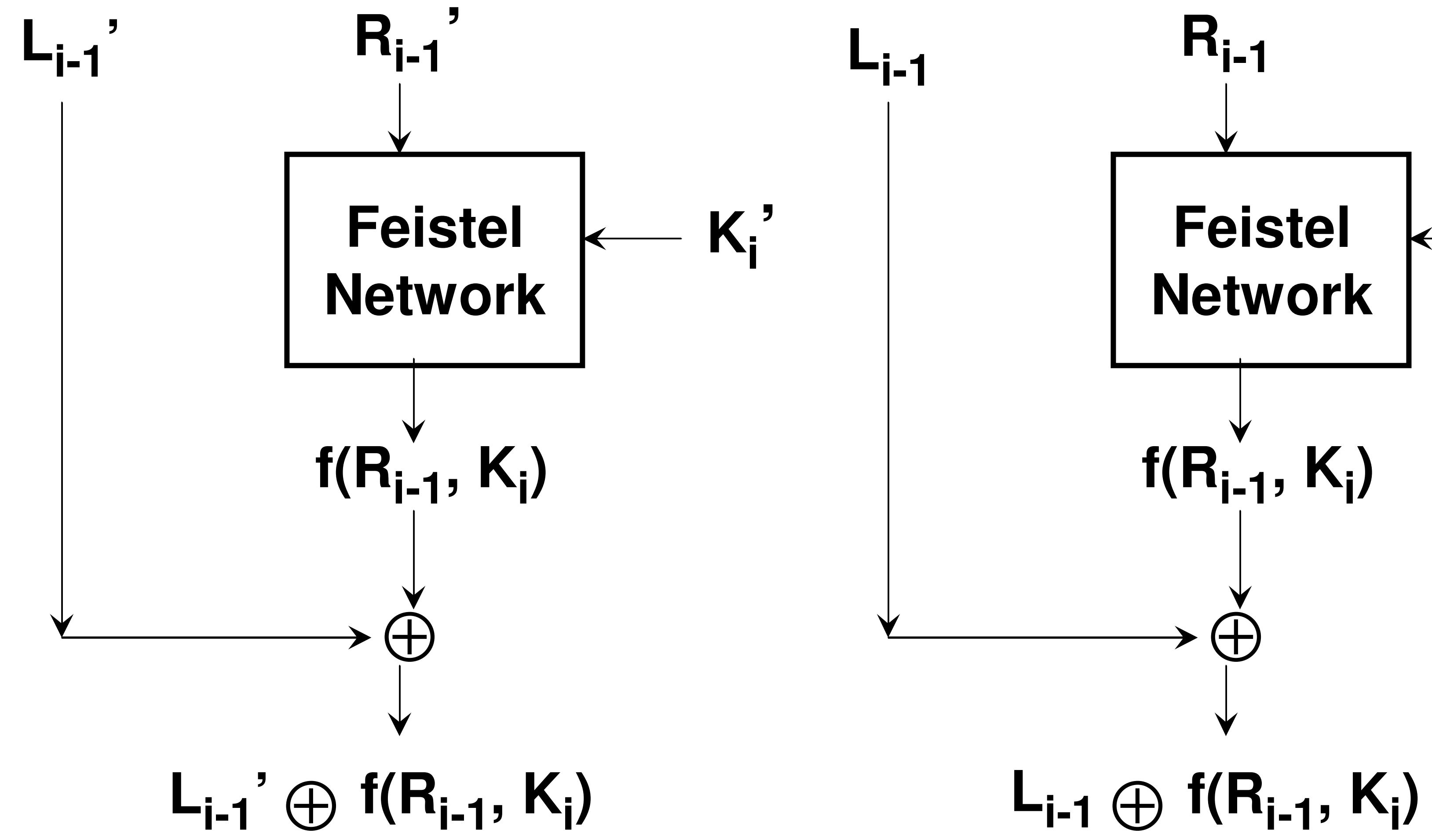
$A \oplus B = A' \oplus B'$

$(A \oplus B)' = A' \oplus B$

$(A \oplus B)' = A \oplus B'$

If Ri-1’ and Ki’ are passed to a Fiestel Network, the the output of the first XOR operator will be the same as the one obtained when one passes Ri-1 and Ki as Inputs. As a result, what comes out of the Feistel network is also the same as that comes out with Ri-1 and Ki.

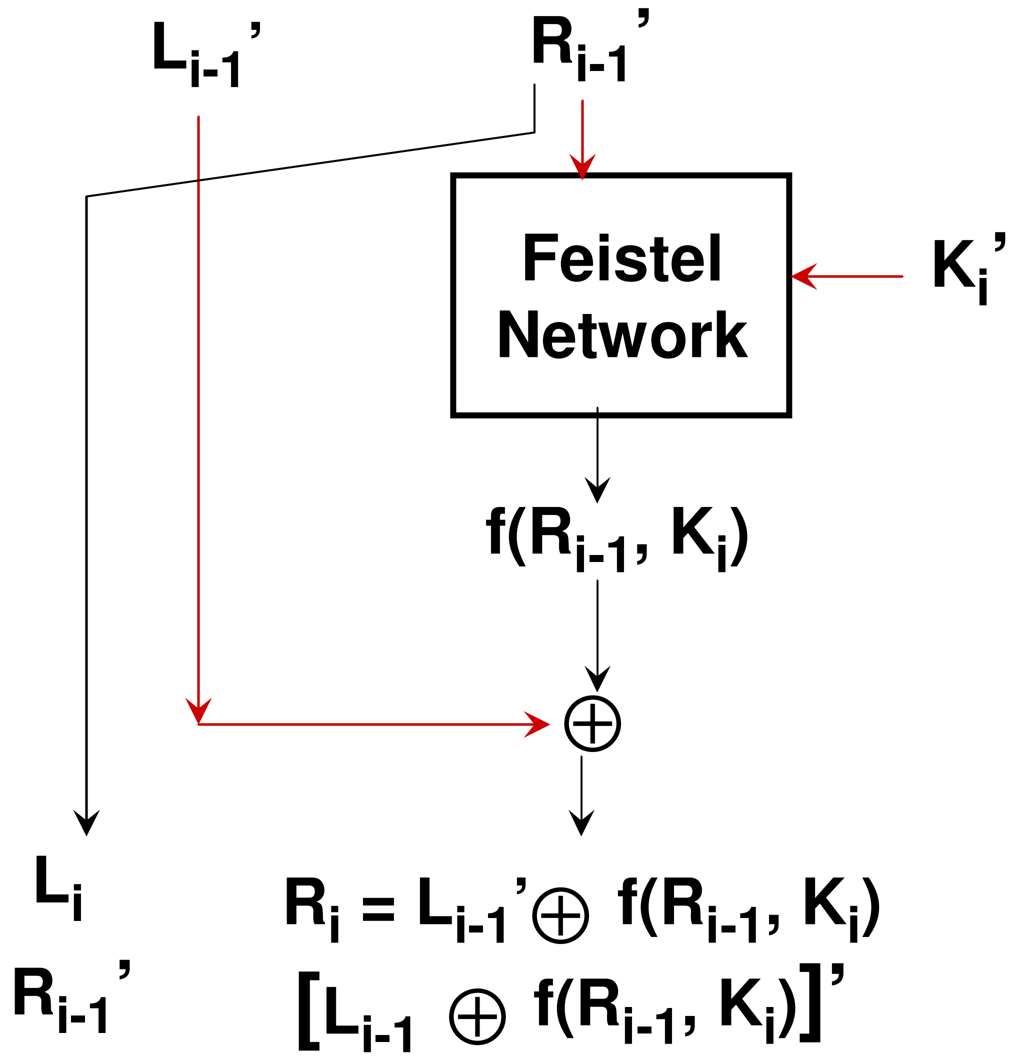
Proof of DES Complement Property(3)



$$(A \oplus B)' = A' \oplus B$$

$$L_{i-1}' \oplus f(R_{i-1}, K_i) = [L_{i-1} \oplus f(R_{i-1}, K_i)]'$$

Proof of DES Complement Property(4)



So, for every DES encryption cycle i, if we input the complement of Li-1, Ri-1 and Ki, then the output is the complement of the cycle is the complement of what we would get if the inputs Li-1, Ri-1 and Ki.

The observation holds good for each cycle.

Chosen Plaintext Attack on DES (1)

- Let P be the plaintext that is chosen by an attacker. The attacker knows both P and P'
- Let C1 = DES(P, K).
  - The attacker knows C1 (due to this being a chosen plaintext attack) and hence also deduce C1'.
- Due to the complement property C1' = DES(P', K').
- Let C2 = DES(P', K)
  - Due to the chosen plaintext attack, the attacker can pass P' to the DES routine and know C2.
- Also, due to the complement property,
- C2' = DES(P, K') and C2' is known to the attacker without actually running DES.
- The objective would be to determine the key K.

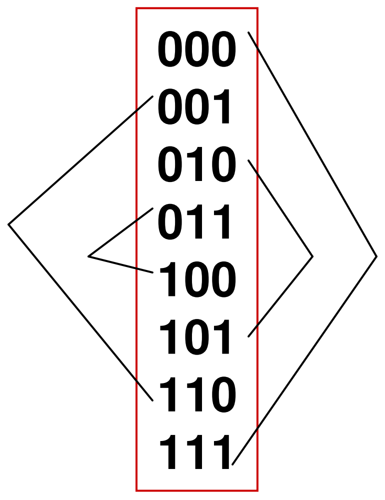
Chosen Plaintext Attack on DES (2)

- Let T be a key chosen from the search space of 56 bits (there are 2^{56} possible combinations of 1s and 0s).
- Let CT = DES(P, T).
- If CT = C1, then T = K
- If CT = C2', then T = K'.
- If CT ≠ C1 and CT ≠ C2', then:
  - T ≠ K as well as T ≠ K'.
- Half of the search space are keys that are complement to the other half.
- With just one key T, we are now able to decide on two keys in the search space.
- Hence, the overall search space is only $O(2^{55})$ and not $O(2^{56})$.

Known

C1 = DES(P, K)
C1' = DES(P', K')
C2 = DES(P', K)
C2' = DES(P, K')

Search Space



Complementary Key pairs

Double DES and Triple DES

- The DES algorithm is fixed for a 56-bit key.
- As the computing power has increased rapidly these days and hopefully will continue in the near future too, it may not be that time consuming to do an exhaustive search of all the 2^{56} keys, when an attacker gets a plaintext and the corresponding ciphertext.
  Double DES:
  - To encrypt: C = E(K2, E(P, K1))
  - To decrypt: P = D(K1, D(K2, C))
  - The encryption/ decryption algorithm used is DES.
- Triple DES:
  - To encrypt: C = E(K3, D(K2, E(K1, P)))
  - To decrypt: P = D(K1, E(K2, D(K3, C)))
  - The encryption/ decryption algorithm used is DES.
  - With 3 keys, 3DES uses 168-bits and is more robust; but, also slow.
  - 3DES has also been adopted for Internet applications like PGP, S/MIME.
  - Note: Triple DES can also be run with two keys such that K1=K3 and K2.

Meet-in-the-Middle Attack with Double DES

- It is a known-plaintext attack where the <plaintext, ciphertext> pair and the encryption algorithm (DES) is known and the key(s) need to be determined.
  - $C = E_{K2}(E_{K1}(P))$
- Since $X = E_{K1}(P) = D_{K2}(C)$, the attack consists of encrypting P with all possible values of 56-bit keys (K_1) and storing the resulting X values. Similarly, we decrypt C with all possible values of 56-bit keys (K_2) and compare the resulting values for a match with the set obtained based on K_1. The 56-bit key values (K_1 and K_2) for which $E_{K1}(P) = D_{K2}(C)$, constitute the 112-bit key K_1 K_2.
- The time complexity for cryptanalysis is thus $O(2^{56})$ and not $O(2^{112})$.

Advanced Encryption Standard (AES)

- AES is a block cipher with a block length of 128 bits
- The key length could be 128, 192 or 256 bits
- The number of rounds for AES varies with the key length:
  - 128 bits: 10 rounds
  - 192 bits: 12 rounds
  - 256 bits: 14 rounds
- In each case: all the rounds are identical, except the last round.
- Each round of AES consists of the following:
  - Single-byte based substitution (byte level)
  - Row-wise permutation (word level)
  - Column-wise mixing (word level)
  - XOR (addition) with the round key

AES Input Block

- 4x4 matrix of bytes, arranged in a column-major fashion.

Referred to as the State array for each round

$$\begin{array}{cccc} \left[\begin{matrix} byte_0 & byte_4 & byte_8 & byte_{12} \\ byte_1 & byte_5 & byte_9 & byte_{13} \\ byte_2 & byte_6 & byte_{10} & byte_{14} \\ byte_3 & byte_7 & byte_{11} & byte_{15} \end{matrix}\right] \\ \mathbf{W0} \quad \mathbf{W1} \quad \mathbf{W2} \quad \mathbf{W3} \end{array}$$

- A word consists of 4 bytes (32 bits). Each column of the state array is a word.
- Each AES round processes the input state array and produces an output state array.

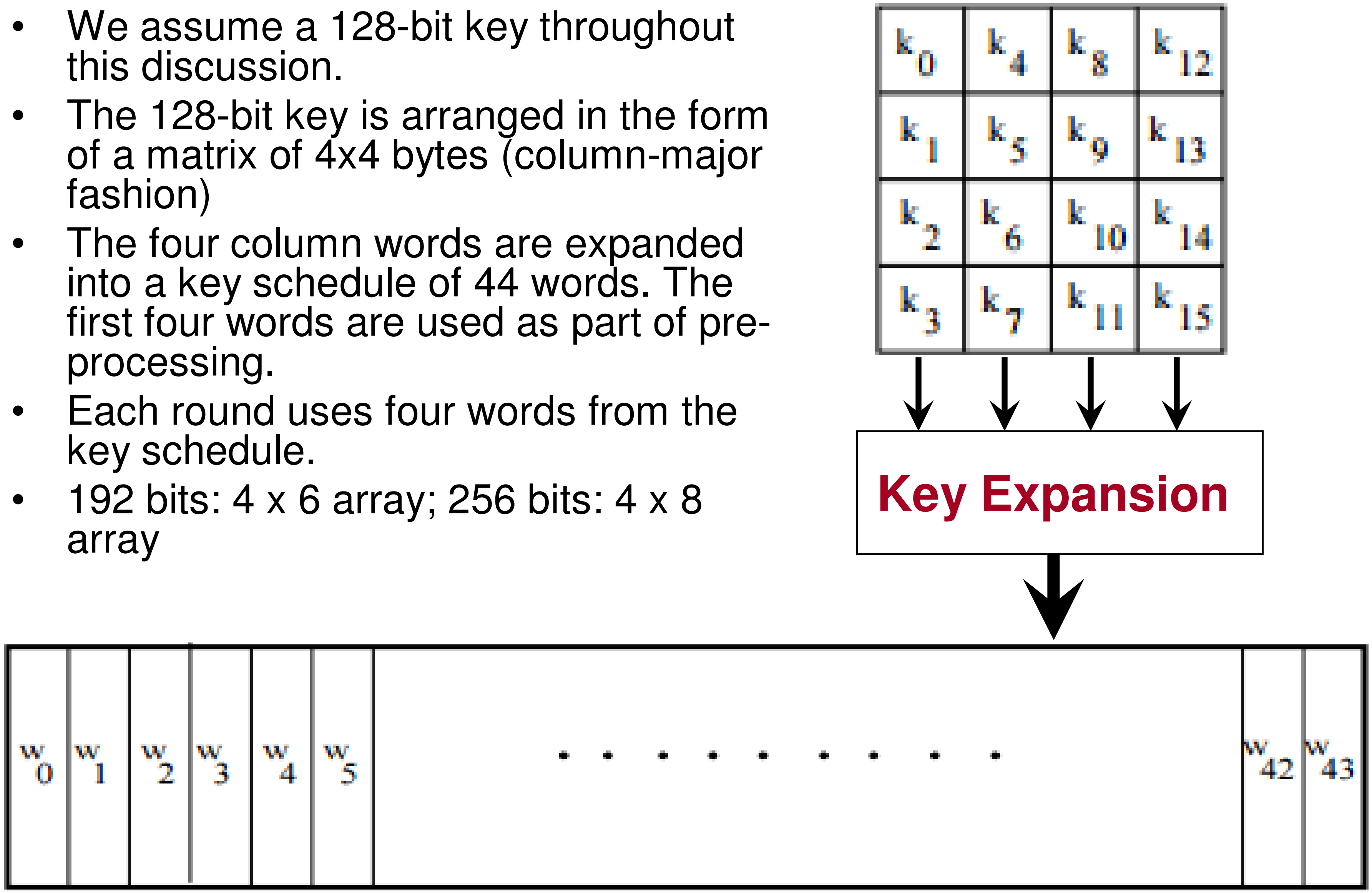
Hexa decimal Basics

0 – 9: 0 – 9

10	A	13	D
11	B	14	E
12	C	15	F

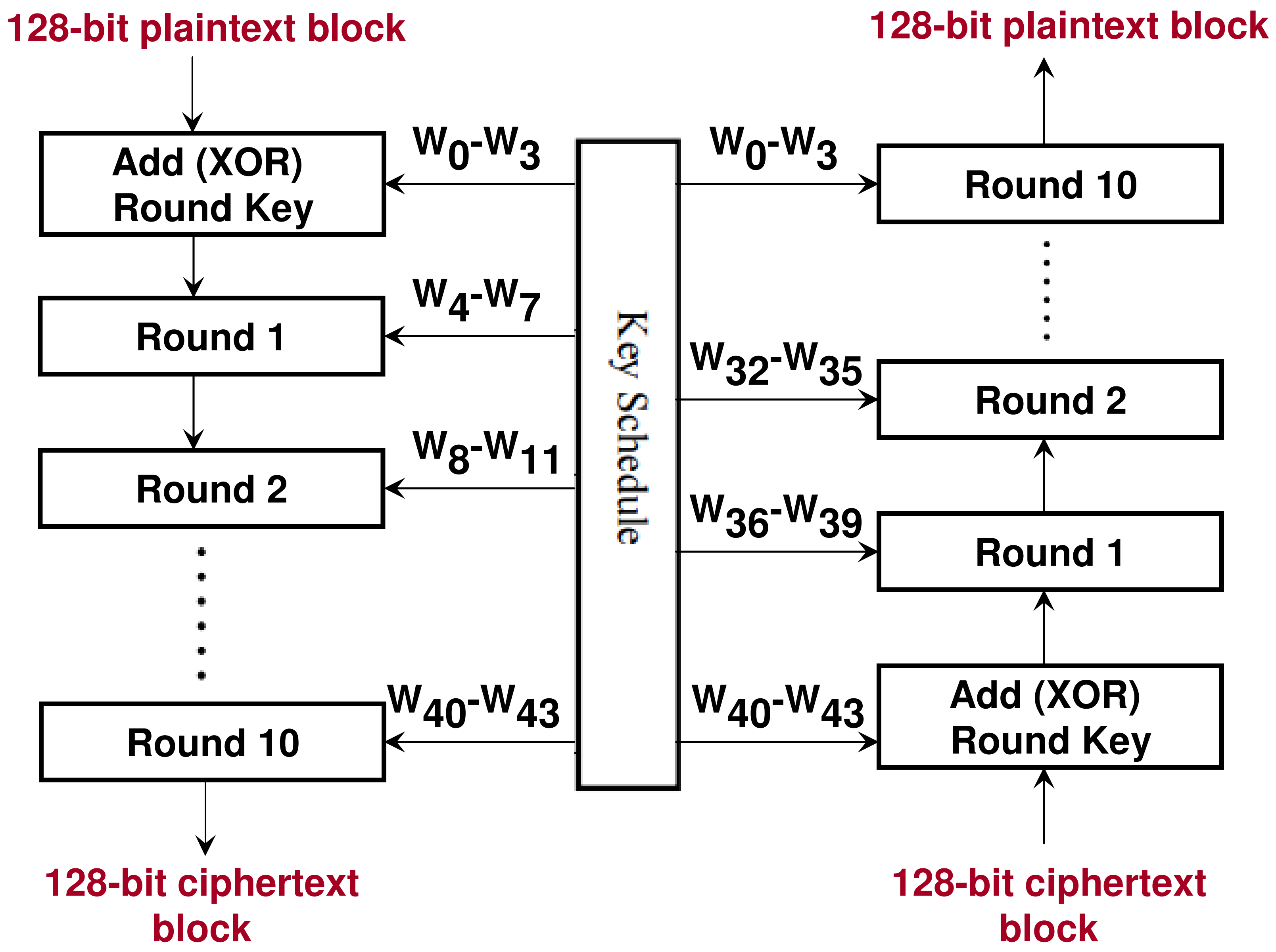
AES Encryption Key and its Expansion

- We assume a 128-bit key throughout this discussion.
- The 128-bit key is arranged in the form of a matrix of 4x4 bytes (column-major fashion)
- The four column words are expanded into a key schedule of 44 words. The first four words are used as part of pre-processing.
- Each round uses four words from the key schedule.
- 192 bits: 4 x 6 array; 256 bits: 4 x 8 array

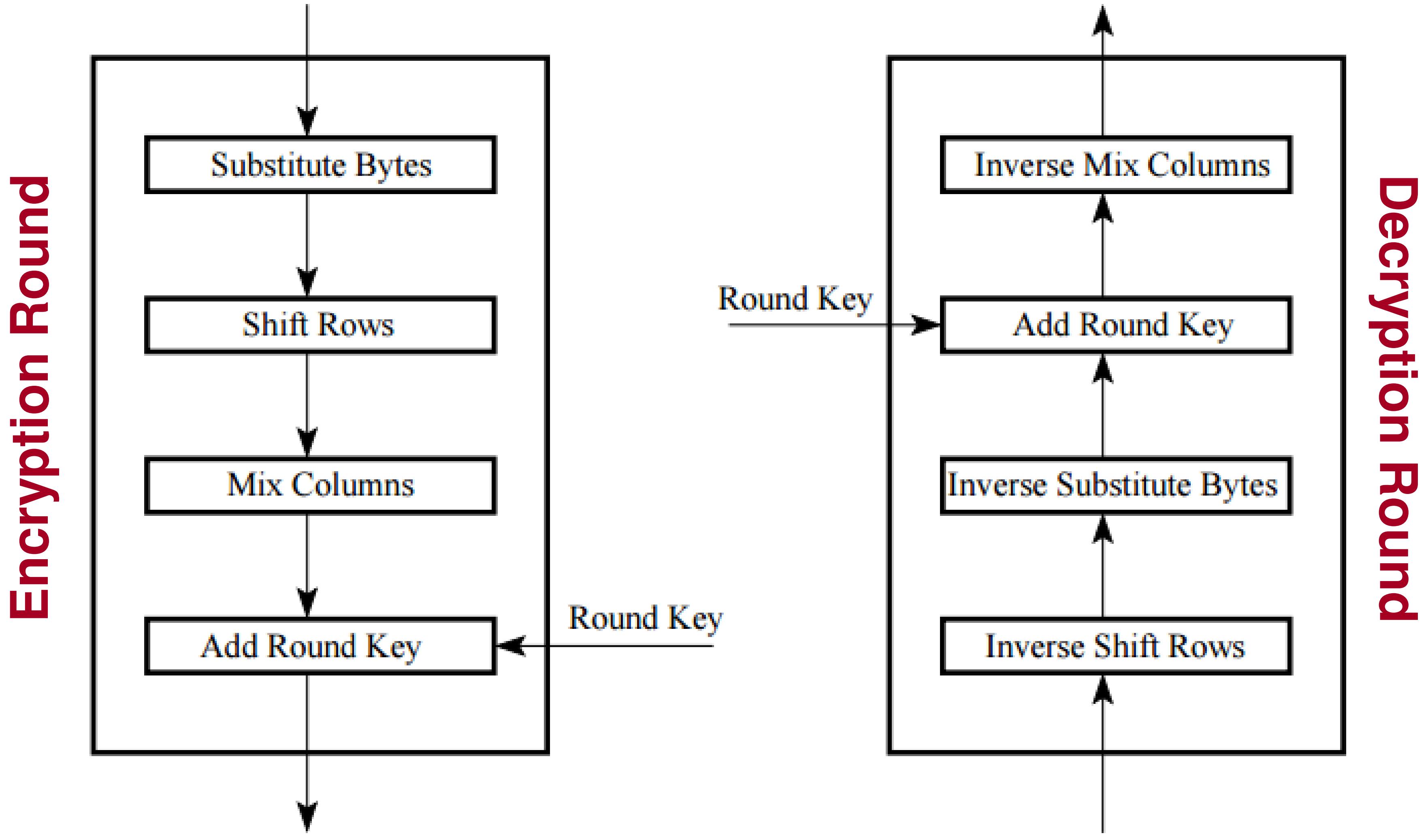


k_0 k_4 k_8 k_{12}
k_1 k_5 k_9 k_{13}
k_2 k_6 k_{10} k_{14}
k_3 k_7 k_{11} k_{15}
Key Expansion
w_0 w_1 w_2 w_3 w_4 w_5 . . . . w_{42} w_{43}

Overall Structure of AES



Encryption and Decryption Round



Note: The last round of encryption does not involve the “Mix Columns” step.
The last round of decryption does not involve the “Inverse Mix Columns” step.

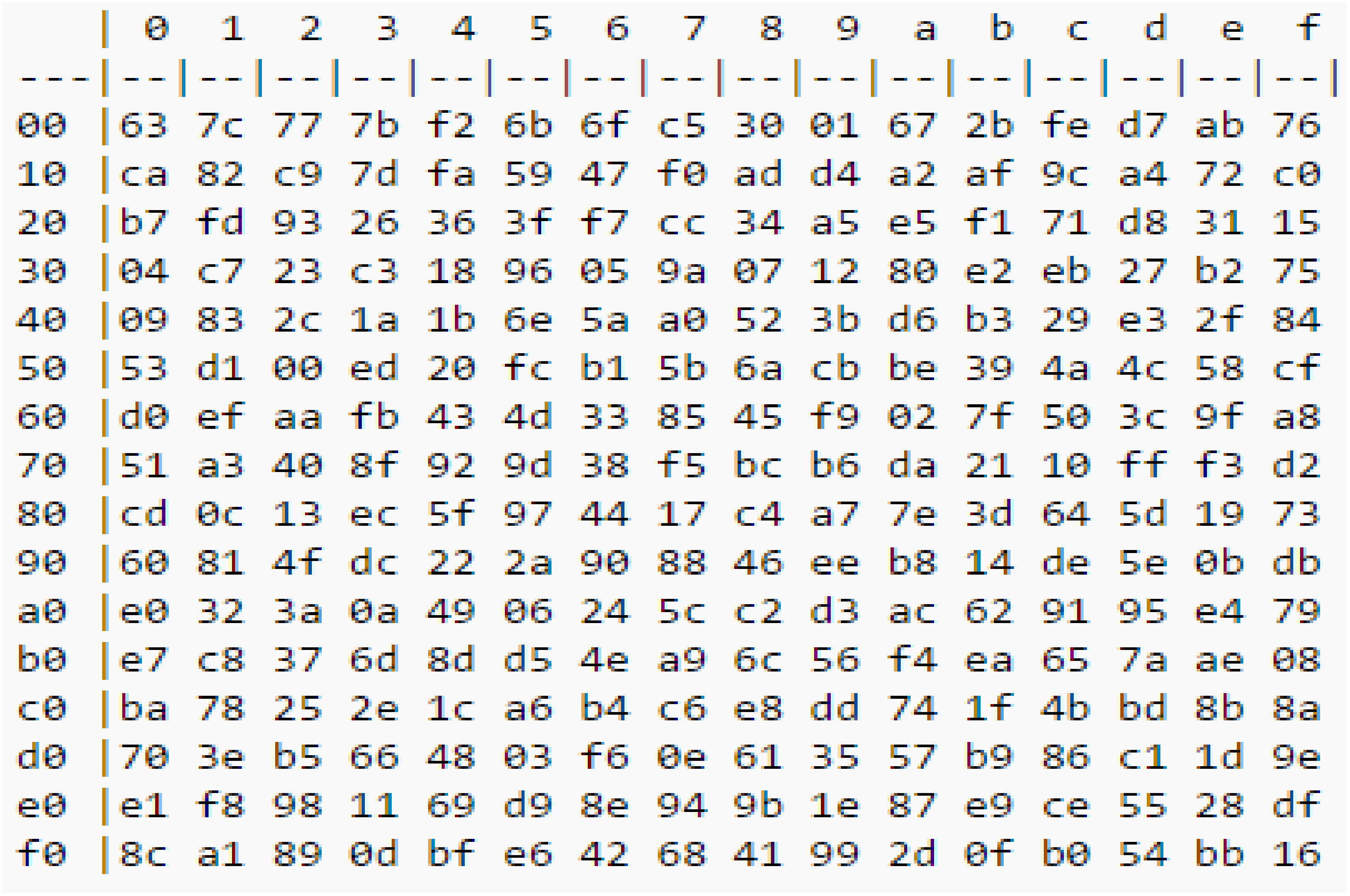
SUBSTITUTE BYTES STEP (SubBytes and Inverse SubBytes)

- This is a byte-by-byte substitution step using a 16 x 16 lookup table (whose entry values range from 0 to 255: a byte each).
- The same lookup table is used for each byte in all the rounds
  - One lookup table for SubBytes: encryption
  - A different (but related) lookup table for InvSubBytes: decryption
- The substitution lookup tables are developed based on bit scrambling (a kind of randomization) to reduce the correlation between the input bits and the output bits at the byte level.
- To find the substitute for an input byte, we break the byte into two four-bit units (nibble); use the first 4-bit nibble as the row index and the second 4-bit nibble as the column index to the lookup table.

SubBytes Lookup Table (dec.)

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	99	124	119	123	242	107	111	197	48	1	103	43	254	215	171	118
1	202	130	201	125	250	89	71	240	173	212	162	175	156	164	114	192
2	183	253	147	38	54	63	247	204	52	165	229	241	113	216	49	21
3	4	199	35	195	24	150	5	154	7	18	128	226	235	39	178	117
4	9	131	44	26	27	110	90	160	82	59	214	179	41	227	47	132
5	83	209	0	237	32	252	177	91	106	203	190	57	74	76	88	207
6	208	239	170	251	67	77	51	133	69	249	2	127	80	60	159	168
7	81	163	64	143	146	157	56	245	188	182	218	33	16	255	243	210
8	205	12	19	236	95	151	68	23	196	167	126	61	100	93	25	115
9	96	129	79	220	34	42	144	136	70	238	184	20	222	94	11	219
10	224	50	58	10	73	6	36	92	194	211	172	98	145	149	228	121
11	231	200	55	109	141	213	78	169	108	86	244	234	101	122	174	8
12	186	120	37	46	28	166	180	198	232	221	116	31	75	189	139	138
13	112	62	181	102	72	3	246	14	97	53	87	185	134	193	29	158
14	225	248	152	17	105	217	142	148	155	30	135	233	206	85	40	223
15	140	161	137	13	191	230	66	104	65	153	45	15	176	84	187	22

SubBytes Lookup Table (hex.)

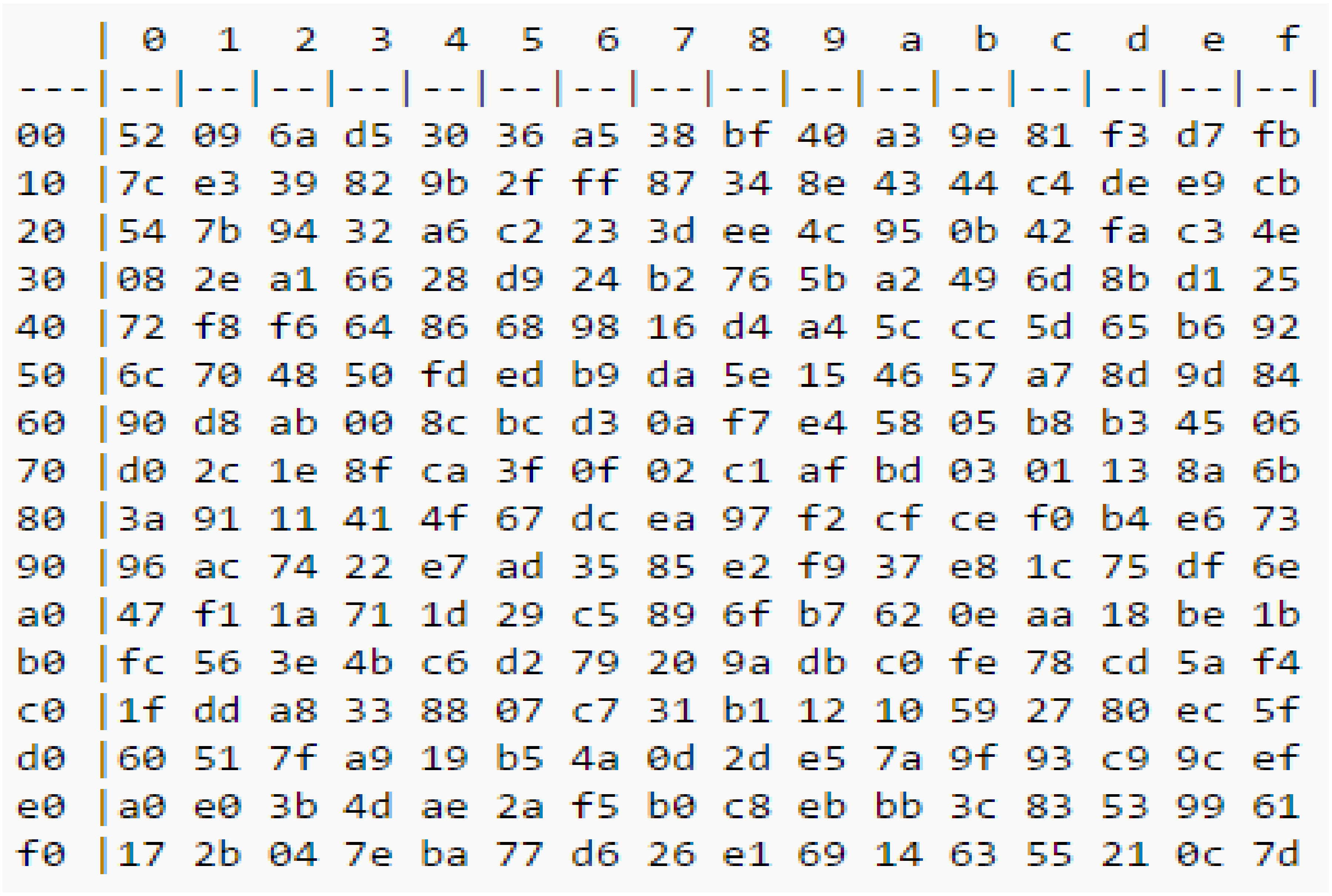


	0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
00	63	7c	77	7b	f2	6b	6f	c5	30	01	67	2b	fe	d7	ab	76
10	ca	82	c9	7d	fa	59	47	f0	ad	d4	a2	af	9c	a4	72	c0
20	b7	fd	93	26	36	3f	f7	cc	34	a5	e5	f1	71	d8	31	15
30	04	c7	23	c3	18	96	05	9a	07	12	80	e2	eb	27	b2	75
40	09	83	2c	1a	1b	6e	5a	a0	52	3b	d6	b3	29	e3	2f	84
50	53	d1	00	ed	20	fc	b1	5b	6a	cb	be	39	4a	4c	58	cf
60	d0	ef	aa	fb	43	4d	33	85	45	f9	02	7f	50	3c	9f	a8
70	51	a3	40	8f	92	9d	38	f5	bc	b6	da	21	10	ff	f3	d2
80	cd	0c	13	ec	5f	97	44	17	c4	a7	7e	3d	64	5d	19	73
90	60	81	4f	dc	22	2a	90	88	46	ee	b8	14	de	5e	0b	db
a0	e0	32	3a	0a	49	06	24	5c	c2	d3	ac	62	91	95	e4	79
b0	e7	c8	37	6d	8d	d5	4e	a9	6c	56	f4	ea	65	7a	ae	08
c0	ba	78	25	2e	1c	a6	b4	c6	e8	dd	74	1f	4b	bd	8b	8a
d0	70	3e	b5	66	48	03	f6	0e	61	35	57	b9	86	c1	1d	9e
e0	e1	f8	98	11	69	d9	8e	94	9b	1e	87	e9	ce	55	28	df
f0	8c	a1	89	0d	bf	e6	42	68	41	99	2d	0f	b0	54	bb	16

Inverse SubBytes Lookup Table (dec)

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	82	9	106	213	48	54	165	56	191	64	163	158	129	243	215	251
1	124	227	57	130	155	47	255	135	52	142	67	68	196	222	233	203
2	84	123	148	50	166	194	35	61	238	76	149	11	66	250	195	78
3	8	46	161	102	40	217	36	178	118	91	162	73	109	139	209	37
4	114	248	246	100	134	104	152	22	212	164	92	204	93	101	182	146
5	108	112	72	80	253	237	185	218	94	21	70	87	167	141	157	132
6	144	216	171	0	140	188	211	10	247	228	88	5	184	179	69	6
7	208	44	30	143	202	63	15	2	193	175	189	3	1	19	138	107
8	58	145	17	65	79	103	220	234	151	242	207	206	240	180	230	115
9	150	172	116	34	231	173	53	133	226	249	55	232	28	117	223	110
10	71	241	26	113	29	41	197	137	111	183	98	14	170	24	190	27
11	252	86	62	75	198	210	121	32	154	219	192	254	120	205	90	244
12	31	221	168	51	136	7	199	49	177	18	16	89	39	128	236	95
13	96	81	127	169	25	181	74	13	45	229	122	159	147	201	156	239
14	160	224	59	77	174	42	245	176	200	235	187	60	131	83	153	97
15	23	43	4	126	186	119	214	38	225	105	20	99	85	33	12	125

Inverse SubBytes Lookup Table (hex)



	0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
00	52	09	6a	d5	30	36	a5	38	bf	40	a3	9e	81	f3	d7	fb
10	7c	e3	39	82	9b	2f	ff	87	34	8e	43	44	c4	de	e9	cb
20	54	7b	94	32	a6	c2	23	3d	ee	4c	95	0b	42	fa	c3	4e
30	08	2e	a1	66	28	d9	24	b2	76	5b	a2	49	6d	8b	d1	25
40	72	f8	f6	64	86	68	98	16	d4	a4	5c	cc	5d	65	b6	92
50	6c	70	48	50	fd	ed	b9	da	5e	15	46	57	a7	8d	9d	84
60	90	d8	ab	00	8c	bc	d3	0a	f7	e4	58	05	b8	b3	45	06
70	d0	2c	1e	8f	ca	3f	0f	02	c1	af	bd	03	01	13	8a	6b
80	3a	91	11	41	4f	67	dc	ea	97	f2	cf	ce	f0	b4	e6	73
90	96	ac	74	22	e7	ad	35	85	e2	f9	37	e8	1c	75	df	6e
a0	47	f1	1a	71	1d	29	c5	89	6f	b7	62	0e	aa	18	be	1b
b0	fc	56	3e	4b	c6	d2	79	20	9a	db	c0	fe	78	cd	5a	f4
c0	1f	dd	a8	33	88	07	c7	31	b1	12	10	59	27	80	ec	5f
d0	60	51	7f	a9	19	b5	4a	0d	2d	e5	7a	9f	93	c9	9c	ef
e0	a0	e0	3b	4d	ae	2a	f5	b0	c8	eb	bb	3c	83	53	99	61
f0	17	2b	04	7e	ba	77	d6	26	e1	69	14	63	55	21	0c	7d

Shift Rows Step

- Shift Rows transformation:
  - The first row is NOT shifted
  - The second row is shifted <u>one byte to the left</u>
  - The third row is shifted <u>two bytes to the left</u>
  - The fourth row is shifted <u>three bytes to the left</u>
- <u>Scrambling</u>: As the bytes of the state array are filled column-wise, shifting the rows in the manner indicated above scrambles the byte order of the state array and promotes diffusion.

$$\begin{bmatrix} s_{0.0} & s_{0,1} & s_{0,2} & s_{0,3} \\ s_{1.0} & s_{1,1} & s_{1,2} & s_{1,3} \\ s_{2.0} & s_{2,1} & s_{2,2} & s_{2,3} \\ s_{3.0} & s_{3,1} & s_{3,2} & s_{3,3} \end{bmatrix} \quad ===> \quad \begin{bmatrix} s_{0.0} & s_{0,1} & s_{0,2} & s_{0,3} \\ s_{1.1} & s_{1,2} & s_{1,3} & s_{1,0} \\ s_{2.2} & s_{2,3} & s_{2,0} & s_{2,1} \\ s_{3.3} & s_{3,0} & s_{3,1} & s_{3,2} \end{bmatrix}$$

Inverse Shift Rows Step

- Inverse Shift Rows transformation:
  - The first row is NOT shifted
  - The second row is shifted <u>one byte to the right</u>
  - The third row is shifted <u>two bytes to the right</u>
  - The fourth row is shifted <u>three bytes to the right</u>

$$\begin{bmatrix} s_{0.0} & s_{0,1} & s_{0,2} & s_{0,3} \\ s_{1.0} & s_{1,1} & s_{1,2} & s_{1,3} \\ s_{2.0} & s_{2,1} & s_{2,2} & s_{2,3} \\ s_{3.0} & s_{3,1} & s_{3,2} & s_{3,3} \end{bmatrix} ===> \begin{bmatrix} s_{0.0} & s_{0,1} & s_{0,2} & s_{0,3} \\ s_{1.3} & s_{1,0} & s_{1,1} & s_{1,2} \\ s_{2.2} & s_{2,3} & s_{2,0} & s_{2,1} \\ s_{3.1} & s_{3,2} & s_{3,3} & s_{3,0} \end{bmatrix}$$

Mix Columns and Inv. Mix Col. Step

- This step replaces each byte of a column by a function of all the bytes in the same column
- All multiplications are according to the GF(2^8) arithmetic and all additions are XOR operations.
- For Encryption, the state matrix is multiplied with the following matrix

Mix Columns

$$\begin{bmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{bmatrix} \times \begin{bmatrix} s_{0.0} & s_{0,1} & s_{0,2} & s_{0,3} \\ s_{1.0} & s_{1,1} & s_{1,2} & s_{1,3} \\ s_{2.0} & s_{2,1} & s_{2,2} & s_{2,3} \\ s_{3.0} & s_{3,1} & s_{3,2} & s_{3,3} \end{bmatrix} = \begin{bmatrix} s'_{0.0} & s'_{0,1} & s'_{0,2} & s'_{0,3} \\ s'_{1.0} & s'_{1,1} & s'_{1,2} & s'_{1,3} \\ s'_{2.0} & s'_{2,1} & s'_{2,2} & s'_{2,3} \\ s'_{3.0} & s'_{3,1} & s'_{3,2} & s'_{3,3} \end{bmatrix}$$

State Matrix

- For Decryption, the state matrix is multiplied with the following matrix

Inv. Mix Cols

$$\begin{bmatrix} 0E & 0B & 0D & 09 \\ 09 & 0E & 0B & 0D \\ 0D & 09 & 0E & 0B \\ 0B & 0D & 09 & 0E \end{bmatrix} \times \begin{bmatrix} s_{0.0} & s_{0,1} & s_{0,2} & s_{0,3} \\ s_{1.0} & s_{1,1} & s_{1,2} & s_{1,3} \\ s_{2.0} & s_{2,1} & s_{2,2} & s_{2,3} \\ s_{3.0} & s_{3,1} & s_{3,2} & s_{3,3} \end{bmatrix} = \begin{bmatrix} s'_{0.0} & s'_{0,1} & s'_{0,2} & s'_{0,3} \\ s'_{1.0} & s'_{1,1} & s'_{1,2} & s'_{1,3} \\ s'_{2.0} & s'_{2,1} & s'_{2,2} & s'_{2,3} \\ s'_{3.0} & s'_{3,1} & s'_{3,2} & s'_{3,3} \end{bmatrix}$$

AES Columns – Finite Field Arithmetic

- Also called Galois Field (GF) arithmetic
- AES uses GF(2^8) arithmetic: all values are in the range 0 – 255
- We write all values in hex: a byte is written as two hexadecimal values
- A binary string is represented as a polynomial
  - 00110110: $X^5 + X^4 + X^2 + X$
  - 10010011: $X^7 + X^4 + X + 1$
- Addition (XOR): Example

Note: $1 + 1 = 0$
Hence, $X^i + X^i = 0$
for any exponent i.

$$36 + 93 = 00110110 + 10010011$$
$$= (X^5 + X^4 + X^2 + X) + (X^7 + X^4 + X + 1)$$
$$= X^5 + \cancel{X^4} + X^2 + \cancel{X} + X^7 + \cancel{X^4} + \cancel{X} + 1$$
$$= X^5 + X^2 + X^7 + 1 = X^7 + X^5 + X^2 + 1$$
$$= 1010\ 0101 = a5$$

Finite Field Arithmetic Multiplication: Ex 1

- (36)(93) = (0011 0110)(1001 0011)

$= (X^5 + X^4 + X^2 + X)(X^7 + X^4 + X + 1)$

$= X^{12} + \cancel{X^9} + \cancel{X^6} + \cancel{X^5} + X^{11} + \cancel{X^8} + \cancel{X^5} + X^4 + \cancel{X^9} + \cancel{X^6} + X^3 + \cancel{X^2} + \cancel{X^8} + X^5 + \cancel{X^2} + X$

$= X^{12} + X^{11} + X^5 + X^4 + X^3 + X = 1100000111010$

If the degree of the resulting polynomial exceeds 7, we need to do an XOR division with the GF(2^8) reducing polynomial: $X^8 + X^4 + X^3 + X + 1 = 100011011$

```
100011011   | 1100000111010
            | 100011011↓
            | ---------
            |  100110001
            |  100011011↓↓↓
            |  ---------
            |      101010010
                   100011011
                   ---------
                     1001001      = 01001001
                                       4   9
```

Prefix the remainder with sufficient 0s to make it 8-bits long

(36)(93) = 49

Finite Field Arithmetic Multiplication: Ex 2

- (53)(ca) = (0101 0011)(1100 1010)

$= (X^6 + X^4 + X + 1)(X^7 + X^6 + X^3 + X)$

$= X^{13} + X^{12} + X^9 + X^7 + X^{11} + X^{10} + X^7 + X^5 + X^8 + X^7 + X^4 + X^2 + X^7 + X^6 + X^3 + X$

$= X^{13} + X^{12} + X^9 + \cancel{X^7} + X^{11} + X^{10} + \cancel{X^7} + X^5 + X^8 + \cancel{X^7} + X^4 + X^2 + \cancel{X^7} + X^6 + X^3 + X$

$= X^{13} + X^{12} + X^{11} + X^{10} + X^9 + X^8 + X^6 + X^5 + X^4 + X^3 + X^2 + X$

= 11111101111110

We divide the above polynomial by the GF(2^8) reducing polynomial:

$X^8 + X^4 + X^3 + X + 1$ = 100011011

Finite Field Arithmetic Multiplication: Ex 2 (continued…)

```
           _______________
100011011 | 1111110111110
            100011011
            ---------
              111000001
              100011011
              ---------
                110110101
                100011011
                ---------
                  101011101
                  100011011
                  ---------
                      100011010
                      100011011
                      ---------
                              1
```

Prefix with seven 0s to make it: 0000 0001 = 01 (hex)

Hence, (53)(ca) = 01

AES Column Multiplication Example

$$\begin{bmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{bmatrix} \begin{bmatrix} d4 \\ bf \\ 5d \\ 30 \end{bmatrix} = \begin{bmatrix} 04 \\ 66 \\ 81 \\ e5 \end{bmatrix}$$

Note: All values shown here are in hex.

Assume the column used is the first column of the state matrix

<u>Steps to show how the first value in the product vector is **04**</u>

$$\begin{aligned}
&(02*d4) + (03*bf) + (01*5d) + (01*30) \\
= \ &(0000\ 0010 * 1101\ 0100) + \\
&(0000\ 0011 * 1011\ 1111) + \\
&(0000\ 0001 * 0101\ 1101) + \\
&(0000\ 0001 * 0011\ 0000)
\end{aligned}$$

$$\begin{aligned}
= \ &(X)(X^7 + X^6 + X^4 + X^2) + \\
&(X + 1)(X^7 + X^5 + X^4 + X^3 + X^2 + X + 1) + \\
&(1)(X^6 + X^4 + X^3 + X^2 + 1) + \\
&(1)(X^5 + X^4)
\end{aligned}$$

$$\begin{array}{lllllllllll}
= X^8 + & X^7 + & & X^5 + & & X^3 + & & & \\
X^8 + & & X^6 + & X^5 + & X^4 + & X^3 + & X^2 + & X + & \\
 & X^7 + & & X^5 + & X^4 + & X^3 + & X^2 + & X + & 1 + \\
 & & X^6 + & & X^4 + & X^3 + & X^2 & & + 1 + \\
 & & & X^5 + & X^4 & & & &
\end{array}$$

AES Column Multiplication Ex. (cont.)

Steps to show how the first value in the product vector is **04**

$$\begin{array}{lllllllllllllll}
= \cancel{X^8} & + & \cancel{X^7} & + & & & \cancel{X^5} & + & & & \cancel{X^3} & + & & & \\
\cancel{X^8} & + & & & \cancel{X^6} & + & \cancel{X^5} & + & \cancel{X^4} & + & \cancel{X^3} & + & \cancel{X^2} & + & \cancel{X} + \\
& & \cancel{X^7} & + & & & \cancel{X^5} & + & \cancel{X^4} & + & \cancel{X^3} & + & \cancel{X^2} & + & \cancel{X} + \cancel{1} + \\
& & & & \cancel{X^6} & + & & & \cancel{X^4} & + & \cancel{X^3} & + & X^2 & & + \cancel{1} + \\
& & & & & & \cancel{X^5} & + & \cancel{X^4} & & & & & &
\end{array}$$

$= X^2$

$=$ 0000 0100 $=$ **0 4**

AES Column Multiplication Ex. (cont.)

$$\begin{bmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{bmatrix} \begin{bmatrix} d4 \\ bf \\ 5d \\ 30 \end{bmatrix} = \begin{bmatrix} 04 \\ 66 \\ 81 \\ e5 \end{bmatrix}$$

Note: All values shown here are in hex.

<u>Steps to show how the second value in the product vector is **66**</u>

$$\begin{aligned} & (01*d4) + (02*bf) + (03*5d) + (01*30) \\ = \; & (0000\ 0001 * 1101\ 0100) + \\ & (0000\ 0010 * 1011\ 1111) + \\ & (0000\ 0011 * 0101\ 1101) + \\ & (0000\ 0001 * 0011\ 0000) \end{aligned}$$

$$\begin{aligned} = \; & (1)(X^7 + X^6 + X^4 + X^2) + \\ & (X)(X^7 + X^5 + X^4 + X^3 + X^2 + X + 1) + \\ & (X+1)(X^6 + X^4 + X^3 + X^2 + 1) + \\ & (1)(X^5 + X^4) \end{aligned}$$

$$\begin{array}{lllllllllll} = & & X^7 + & X^6 + & & X^4 + & & X^2 + & & \\ & X^8 + & & X^6 + & X^5 + & X^4 + & X^3 + & X^2 + & X + & \\ & & X^7 + & & X^5 + & X^4 + & X^3 + & & X + & \\ & & & X^6 + & & X^4 + & X^3 + & X^2 & & + 1 + \\ & & & & X^5 + & X^4 & & & & \end{array}$$

AES Column Multiplication Ex. (cont.)

Steps to show how the second value in the product vector is **66**

$$
\begin{aligned}
= \quad & \cancel{X^7} + \cancel{X^6} + \cancel{X^4} + \cancel{X^2} + \\
& X^8 + \cancel{X^6} + \cancel{X^5} + \cancel{X^4} + \cancel{X^3} + \cancel{X^2} + \cancel{X} + \\
& \cancel{X^7} + \cancel{X^5} + \cancel{X^4} + \cancel{X^3} + \cancel{X} + \\
& X^6 + \cancel{X^4} + X^3 + X^2 + 1 + \\
& X^5 + X^4
\end{aligned}
$$

$= X^8 + X^6 + X^5 + X^4 + X^3 + X^2 + 1 = 101111101$

We divide the above polynomial by the GF(2^8) reducing polynomial:
$X^8 + X^4 + X^3 + X + 1 = 100011011$

```
100011011 | 101111101
          | 100011011
          |----------
          |   1100110
```

Prefix with sufficient 0s to make the remainder an 8-bit quantity: 0110 0110 = **6 6**

Differences between AES and DES

- **Input Processing**
- With DES the permutations are based on the Feistel network wherein the input block is divided into two halves, processed separately and then the two halves are swapped.
- AES processes the whole input block and make them go through byte-level substitutions followed by word-level permutations.

- **Encryption and Decryption**
- With DES, the encryption and decryption rounds look the same and are based on the Fiestel network.
- With AES, the encryption and decryption rounds are different (the byte-sub, row-shift and column mix, add round key steps are done in a different order).

- **Avalanche Effect**
- On average, with DES, changing one bit of the plaintext affects 31 bit positions in the ciphertext. With AES, changing one bit of the plaintext affects all the 128 bit positions of the ciphertext.