

Lecture 3: Block Ciphers and the Data Encryption Standard

Lecture Notes on “Computer and Network Security”

by Avi Kak (kak@purdue.edu)

April 25, 2009

©2009 Avinash Kak, Purdue University

Goals:

- To introduce the notion of a block cipher in the modern context.
- To talk about the infeasibility of **ideal block ciphers**
- To introduce the notion of the **Feistel Cipher Structure**
- To go over **DES**, the Data Encryption Standard

3.1: Ideal Block Cipher

- In a modern block cipher (but still using a classical encryption method), we replace a block of N bits from the plaintext with a block of N bits from the ciphertext. This general idea is illustrated in the figure on the next page.
- One typically wants to use 64-bit blocks.
- In an ideal block cipher, the relationship between the input blocks and the output block is completely random. But it must be invertible for decryption to work. Therefore, it has to be one-to-one, meaning that each input block is mapped to a unique output block.
- The encryption key for the ideal block cipher is the codebook itself, meaning the table that shows the relationship between the input blocks and the output blocks.
- Figure 1 depicts an ideal block cipher that uses blocks of size 4. Each block of 4 bits in the plaintext is transformed into a block of 4 ciphertext bits.

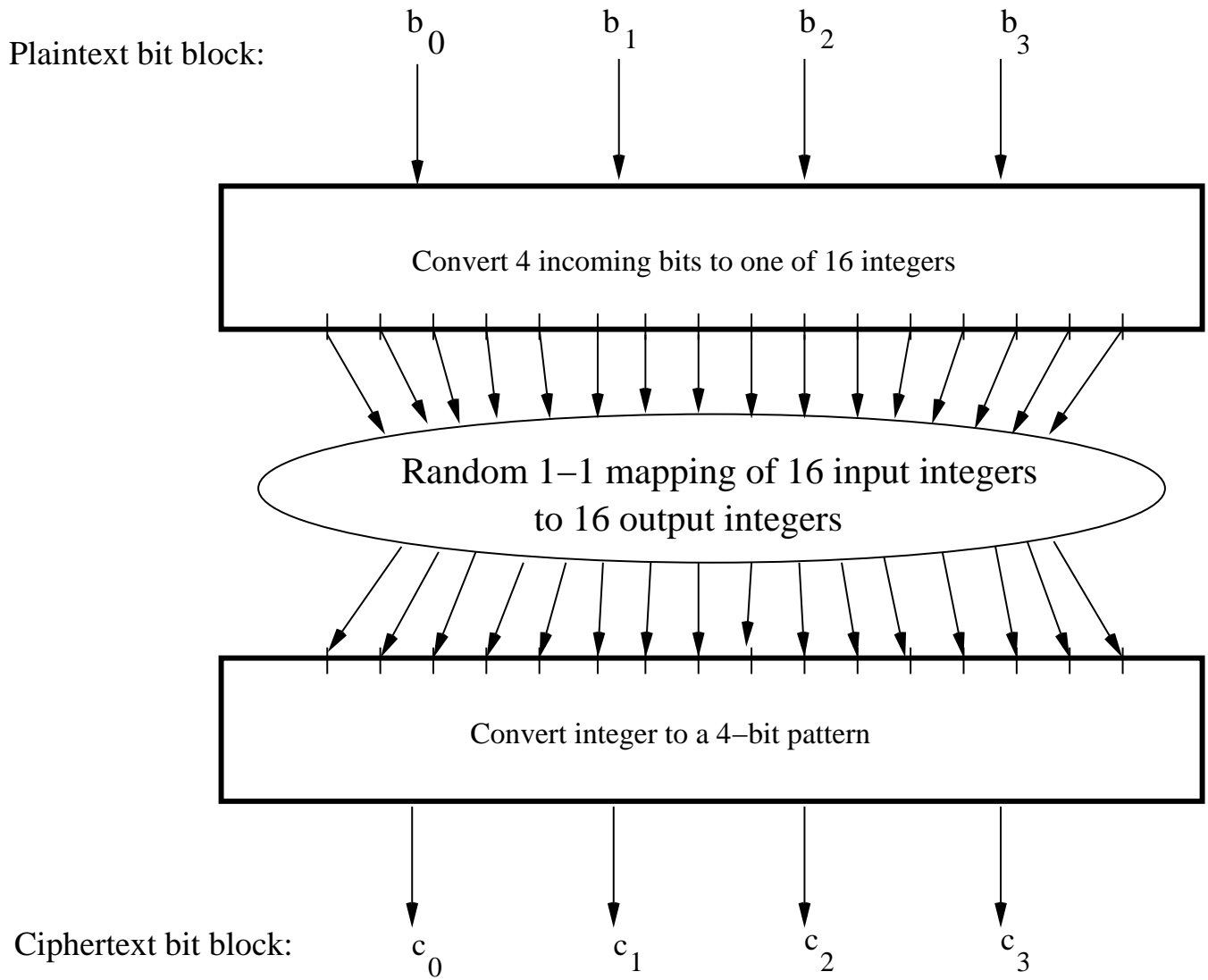


Figure 1: *This figure is from Lecture 3 of “Lecture Notes on Computer and Network Security” by Avi Kak*

3.2: What's the Size of the Encryption Key for the Ideal Block Cipher?

- With a 64-bit block, we can think of each possible input block as one of 2^{64} integers and for each such integer we can specify an output 64-bit block. We can construct the codebook by displaying just the output blocks in the order of the integers corresponding to the input blocks. Such a code book will be of size $64 \times 2^{64} \approx 10^{21}$.
- That implies that the encryption key for the ideal block cipher using 64-bit blocks will be of size 10^{21} .
- The size of the encryption key would make the ideal block cipher an impractical idea. Think of the logistical issues related to the transmission, storage, and processing of such large keys.

3.3: The Feistel Structure for Block Ciphers

- Named after the IBM cryptographer Horst Feistel and first implemented in the Lucifer cipher by Horst Feistel and Don Coppersmith.
- A cryptographic system based on Feistel structure uses the same basic algorithm for both encryption and decryption.
- As shown in Figure 2, the Feistel structure consists of multiple rounds of processing of the plaintext, with each round consisting of a “substitution” step followed by a permutation step.
- The input block to each round is divided into two halves that we can denote **L** and **R** for the left half and the right half.
- In each round, the right half of the block, **R**, goes through unchanged. But the left half, **L**, goes through an operation that depends on **R** and the encryption key.
- The permutation step at the end of each round consists of swapping the modified **L** and **R**. *Therefore, the **L** for the next round would be **R** of the current round. And **R** for the next round be the output **L** of the current round.*

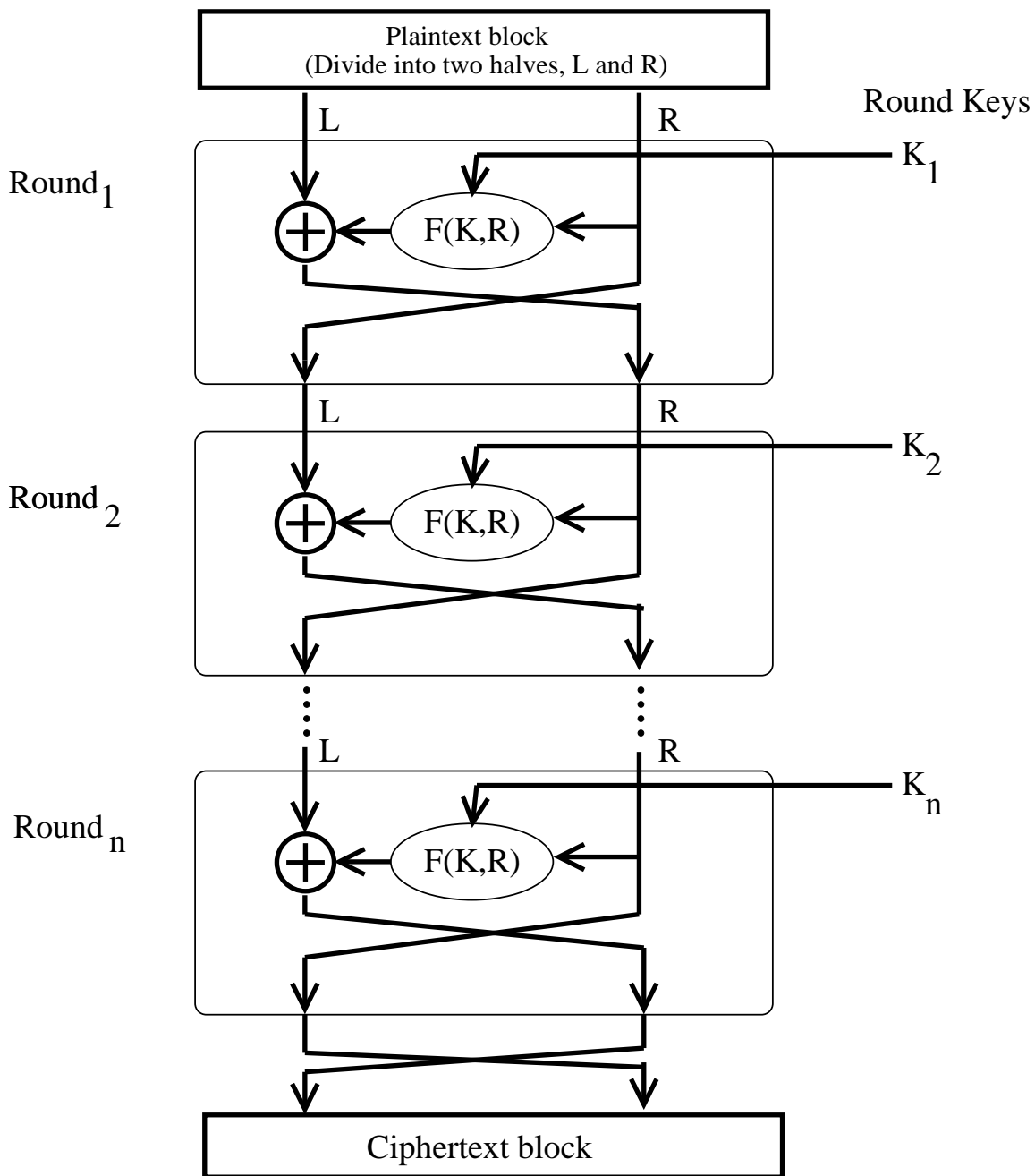


Figure 2: *This figure is from Lecture 3 of “Lecture Notes on Computer and Network Security” by Avi Kak*

3.4: Mathematical Description of Each Round in the Feistel Structure

- Let LE_i and RE_i denote the output half-blocks at the end of the i^{th} round of processing. The letter 'E' denotes encryption.

- We obviously have

$$\begin{aligned}LE_i &= RE_{i-1} \\RE_i &= LE_{i-1} \oplus F(RE_{i-1}, K_i)\end{aligned}$$

where \oplus denotes the bitwise EXCLUSIVE OR operation. The symbol F denotes the operation that “scrambles” RE_{i-1} of the previous round with the current round key K_i . Note that the round key K_i is derived from the main encryption key as we will explain later.

- F is referred to as the Feistel function, after Horst Feistel naturally.

- Assuming 16 rounds of processing (which is typical), the output of the last round of processing is given by

$$\begin{aligned}LE_{16} &= RE_{15} \\RE_{16} &= LE_{15} \oplus F(RE_{15}, K_{16})\end{aligned}$$

3.5: Decryption in Ciphers Based on the Feistel Structure

- As shown in Figure 3, the decryption algorithm is exactly the same as the encryption algorithm with the only difference that the round keys are used in the reverse order.
- **The output of each round during decryption is the input to the corresponding round during encryption. This property holds true regardless of the choice of the Feistel function F .**
- To prove the above claim, let LD_i and RD_i denote the left half and the right half of the output of the i^{th} round.
- That means that the output of the first decryption round consists of LD_1 and RD_1 . So we can denote the input to the first decryption round by LD_0 and RD_0 . The relationship between the two halves that are input to the first decryption round and what is output by the encryption algorithm is

$$\begin{aligned}LD_0 &= RE_{16} \\RD_0 &= LE_{16}\end{aligned}$$

- We can write the following equations for the output of the first decryption round

$$\begin{aligned}
 LD_1 &= RD_0 \\
 &= LE_{16} \\
 &= RE_{15}
 \end{aligned}$$

$$\begin{aligned}
 RD_1 &= LD_0 \oplus F(RD_0, K_{16}) \\
 &= RE_{16} \oplus F(LE_{16}, K_{16}) \\
 &= [LE_{15} \oplus F(RE_{15}, K_{16})] \oplus F(RE_{15}, K_{16}) \\
 &= LE_{15}
 \end{aligned}$$

This shows that the output of the first round of decryption is the same as the input to the last stage of the encryption round since we have $LD_1 = RE_{15}$ and $RD_1 = LE_{15}$

- The following equalities are used in the above derivation. Assume that A , B , and C are bit arrays.

$$\begin{aligned}
 [A \oplus B] \oplus C &= A \oplus [B \oplus C] \\
 A \oplus A &= 0 \\
 A \oplus 0 &= A
 \end{aligned}$$

- **The above result is independent of the precise nature of F .**

Encryption

Decryption

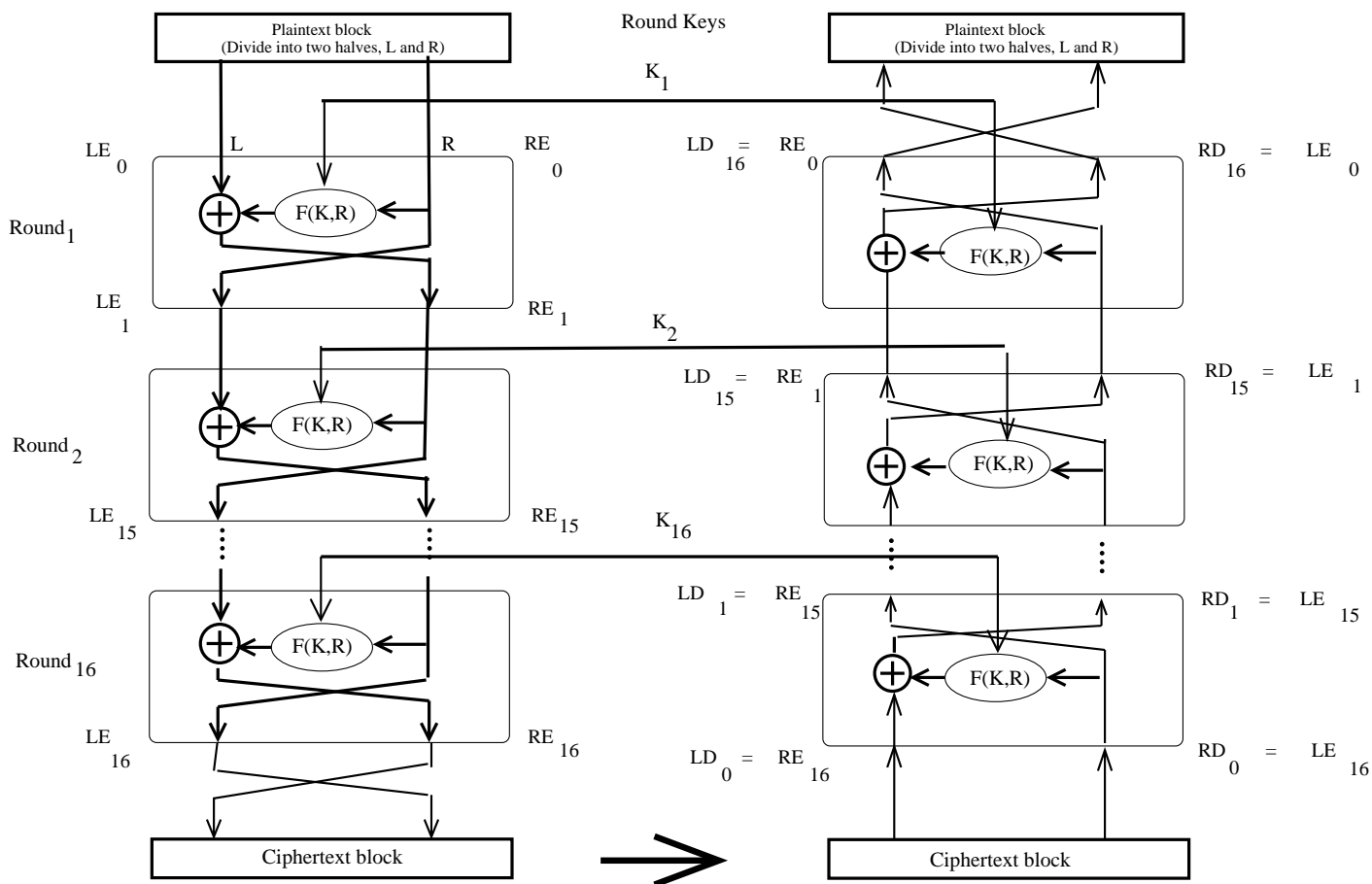


Figure 3: This figure is from Lecture 3 of “Lecture Notes on Computer and Network Security” by Avi Kak

3.6: DES: The Data Encryption Standard

- Adopted by NIST in 1977.
- Based on a cipher (Lucifer) developed earlier by IBM for Lloyd's of London for cash transfer.
- DES uses the Feistel cipher structure with 16 rounds of processing.
- DES uses a 56-bit encryption key. (The key size was apparently dictated by the memory and processing constraints imposed by a single-chip implementation of the algorithm for DES.) The key itself is specified with 8 bytes, but one bit of each byte is used as a parity check.
- DES encryption was broken in 1999 by Electronics Frontiers Organization. This resulted in NIST issuing a new directive that year that required organizations to use **Triple DES**, that is three consecutive applications of **DES**. (That DES was found to be not as strong as originally believed also prompted NIST to

initiate the development of new standards for data encryption. The result is **AES** that we will discuss later.)

- **Triple DES** continues to enjoy wide usage in commercial applications. To understand Triple DES, you must first understand the basic **DES** encryption.
- As mentioned, DES uses the Feistel structure with 16 rounds.
- What is specific to DES is the implementation of the F function in the algorithm and how the round keys are derived from the main encryption key.
- The round keys are generated from the main key by a sequence of permutations. Each round key is 48 bits in length.

3.7: One Round of Processing in DEA

- The algorithmic implementation of DES is known as **DEA** for **Data Encryption Algorithm**. Figure 4 shows a single round of processing in DEA. The dotted rectangle constitutes the F function.
- The 32-bit right half of the 64-bit input data block is expanded by into a 48-bit block. This is referred to as the **expansion permutation** step, or the **E-step**.
- The above-mentioned E-step entails the following:
 - first divide the 32-bit block into eight 4-bit words
 - attach an additional bit on the left to each 4-bit word that is the last bit of the previous 4-bit word
 - attach an additional bit to the right of each 4-bit word that is the beginning bit of the next 4-bit word.

Note that what gets prefixed to the first 4-bit block is the last bit of the last 4-bit block. By the same token, what gets appended to the last 4-bit block is the first bit of the first 4-bit block. The

reason for why we expand each 4-bit block into a 6-bit block in the manner explained will become clear shortly.

- The 56-bit key is divided into two halves, each half shifted separately, and the combined 56-bit key **permuted/contracted** to yield a 48-bit **round** key. How this is done will be explained later.
- The 48 bits of the expanded output produced by the E-step are XORed with the round key. This is referred to as **key mixing**.
- The output produced by the previous step is broken into eight six-bit words. Each six-bit word goes through a substitution step; its replacement is a 4-bit word. The substitution is carried out with an **S-box**, as we explain in greater detail on page 17.
- So after all the substitutions, we again end up with a 32-bit word.
- The 32-bits of the previous step then go through a P-box based permutation, as shown in Figure 4.

- What comes out of the P-box is then XORed with the left half of the 64-bit block that we started out with. The output of this XORing operation gives us the right half block for the next round.
- Note that the goal of the substitution step implemented by the **S-box** is to introduce **diffusion** in the generation of the output from the input. *Diffusion means that each plaintext bit must affect as many ciphertext bits as possible.*
- The strategy used for creating the different round keys from the main key is meant to introduce **confusion** into the encryption process. *Confusion in this context means that the relationship between the encryption key and the ciphertext must be as complex as possible.*
- Diffusion and confusion are the two cornerstones of block cipher design.

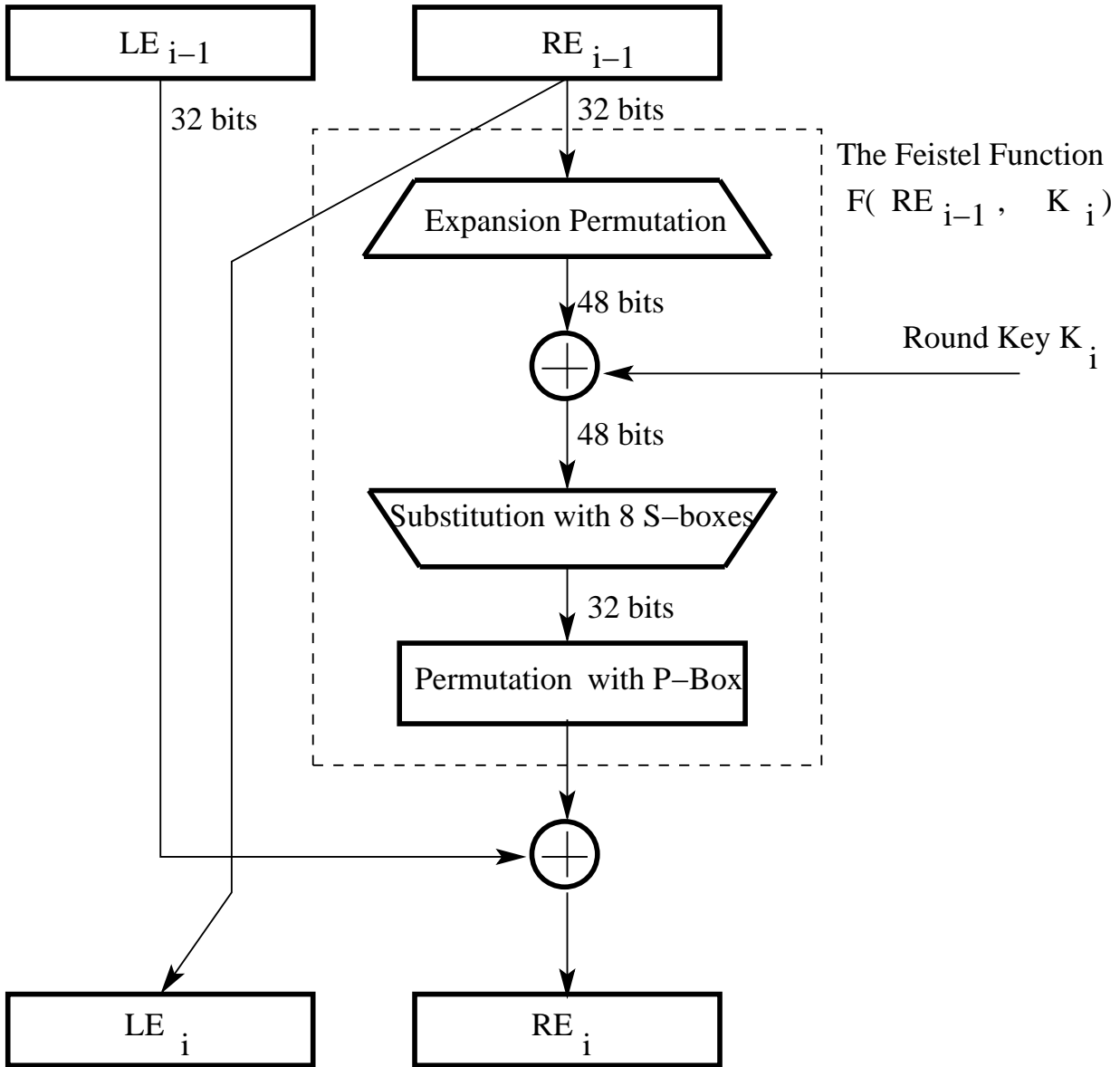


Figure 4: *This figure is from Lecture 3 of “Lecture Notes on Computer and Network Security” by Avi Kak*

3.8: The S-Box for the Substitution Step in Each Round

- As shown in Figure 5, the 48-bit input word is divided into eight 6-bit words and each 6-bit word fed into a separate S-box. Each S-box produces a 4-bit output. Therefore, the 8 S-boxes together generate a 32-bit output. As you can see, the overall substitution step takes the 48-bit input back to a 32-bit output.
- Each of the eight S-boxes consists of a 4×16 table lookup for an output 4-bit word. The first and the last bit of the 6-bit input word are decoded into one of our rows and the middle 4 bits into one of 16 columns for the table lookup.
- The goal of the substitution carried out by an S-box is to enhance **diffusion**. As you will recall from the E-step described on page 13, the expansion-permutation step (the E-step) expands a 32-bit block into a 48-bit block by attaching a bit at the beginning and a bit at the end of each 4-bit sub-block, the two bits needed for these attachments belong to the adjacent blocks.
- Thus, the row lookup for each of the eight S-boxes becomes a function of the input bits for the previous S-box and the next

S-box.

- In the design of the DES, the S-boxes were tuned to enhance the resistance of DES to what is known as the **differential cryptanalysis** attack. Even a slight modification of the S-boxes can weaken the DES to differential cryptanalysis attack. [For block ciphers, differential cryptanalysis consists of presenting to the encryption algorithm pairs of plaintext bit patterns with known differences between them and examining the differences between the corresponding outputs. The outputs are usually recorded at the input to last round of the cipher. Let's represent one plaintext bit block by $X = [X_1, X_2, \dots, X_n]$ where X_i denotes the i^{th} bit in the block, and let's represent the corresponding output bit block by $Y = [Y_1, Y_2, \dots, Y_n]$. By the difference between two plaintext bit blocks X' and X'' we mean $\Delta X = X' \oplus X''$. The difference between the corresponding outputs Y' and Y'' is given by $\Delta Y = Y' \oplus Y''$. The pair $(\Delta X, \Delta Y)$ is known as a **differential**. In an ideally randomizing block cipher, the probability of ΔY being a particular value for a given ΔX is $1/2^n$ for an n -bit block cipher. What is interesting is that the probabilities of ΔY taking on different values for a given ΔX can be shown to be independent of the encryption key because of the properties of the XOR operator, but these probabilities are strongly dependent on the S-box tables. By feeding into a cipher pairs of plaintext blocks with known ΔX and observing the corresponding ΔY , it is possible to establish constraints on the round key bits encountered along the different paths in the encryption processing chain. (By constraints I mean the following: Speaking hypothetically for the purpose of illustrating a point and focussing on just one round of DES, suppose we can show that the following condition can be expected to be obeyed with high probability: $\Delta X \oplus \Delta Y \oplus K_i = 0$ for some bit K_i of the encryption key, then it must be the case that $K_i = \Delta X \oplus \Delta Y$.) Note that differential cryptanalysis is a **chosen plaintext attack**, meaning that the attacker will feed known plaintext bit patterns into the cipher and analyze the corresponding outputs in order to figure out the encryption key. In a theoretical analysis of an attack based on differential cryptanalysis, it was shown by Eli Biham and Adi Shamir in 1990 that the DES's encryption key could be figured out provided one

could feed known 2^{47} plaintext blocks into the cipher. For a tutorial by Howard Heys on differential cryptanalysis, see http://www.engr.mun.ca/~howard/PAPERS/ldc_tutorial.pdf.]

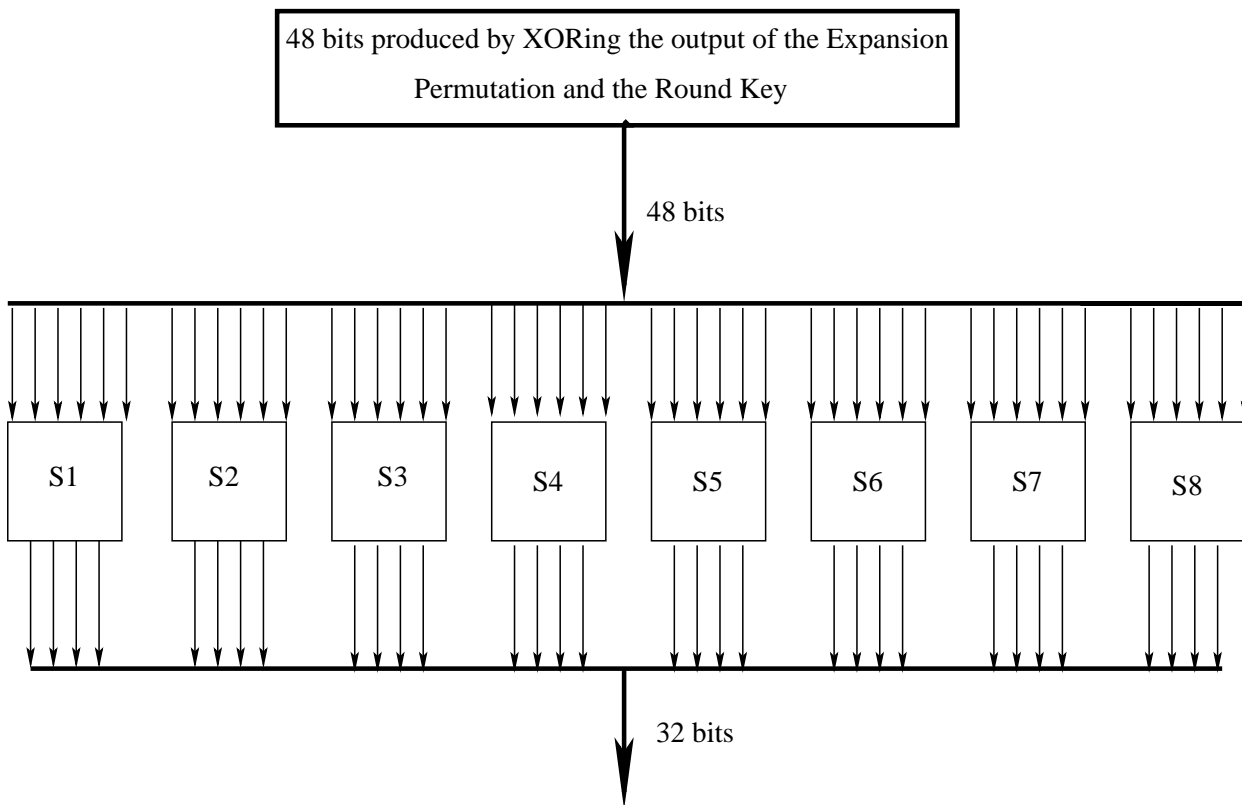


Figure 5: *This figure is from Lecture 3 of “Lecture Notes on Computer and Network Security” by Avi Kak*

3.9: The Substitution Tables

The 4×16 substitution table for S_1															
14	4	13	1	2	15	11	8	3	10	6	12	5	9	0	7
0	15	7	4	14	2	13	1	10	6	12	11	9	5	3	8
4	1	14	8	13	6	2	11	15	12	9	7	3	10	5	0
15	12	8	2	4	9	1	7	5	11	3	14	10	0	6	13

The 4×16 substitution table for S_2															
15	1	8	14	6	11	3	4	9	7	2	13	12	0	5	10
3	13	4	7	15	2	8	14	12	0	1	10	6	9	11	5
0	14	7	11	10	4	13	1	5	8	12	6	9	3	2	15
13	8	10	1	3	15	4	2	11	6	7	12	0	5	14	9

One can similarly specify tables for the other six substitution boxes.

As mentioned earlier, each row of a substitution table is indexed by the two outermost bits of a six-bit block and each column by the remaining inner 4 bit.

3.10: The P-Box Permutation in the Feistel Function

The last step in the Feistel function shown in Figure 4 is labeled “Permutation with P-Box”. The permutation table is shown below.

P-Box Permutation							
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

As with all permutation tables, this permutation table simply means that the first output bit will be the 16th bit of the input, the second output bit the 7th bit of the input, and so on, for all of the 32 bits of the output that are obtained from the 32 bits of the input. *Note that bit indexing starts with 1 and not with 0.*

3.11: Round Key Generation

- The initial 56-bit key may be represented as 8 bytes, with the last bit of each byte used as a parity bit.
- The relevant 56 bits are subject to a permutation at the beginning before any round keys are generated. This is our Permutation Choice 1.
- At the beginning of each round, we divide the 56 relevant key bits into two 28 bit halves and circularly shift each half by one or two bits.
- For generating round key, we join together the two halves and apply a 56 bit to 48 bit contracting permutation (Permutation Choice 2) to the joined bit pattern. The resulting 48 bits constitute our round key.
- The two halves generated in each round are fed as the two halves going into the next round.

3.12: Initial Permutation of the Encryption Key

Permutation Choice 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

Note that the bit positions assume that the key bits are addressed 1 through 64 in an 8-byte bit pattern. But note that the last bit of each byte is used as a parity bit. Also note that the permutation shown is not a table, in the sense that the rows and the columns do not carry any special and separate meanings. The permutation order for the bits is given by reading the entries shown from the upper left corner to the lower right corner.

3.13: Contraction-Permutation that Generates the 48-bit Round Key from the 56 Key

Permutation Choice 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

As with the permutation shown on the previous page, note that the bit positions assume that the key bits are addressed 1 through 64 in an 8-byte bit pattern. But note that the last bit of each byte is used as a parity bit. Also note that the permutation shown is not a table, in the sense that the rows and the columns do not carry any special and separate meanings. The permutation order for the bits is given by reading the entries shown from the upper left corner to the lower right corner.

3.14: What Makes DES Strong (to the Extent it is Strong)

- The substitution step is very effective as far as **diffusion** is concerned. It has been shown that if you change just one bit of the 64-bit input data block, on the average that alters 34 bits of the ciphertext block.
- The manner in which the round keys are generated from the encryption key is also very effective as far as **confusion** is concerned. It has been shown that if you change just one bit of the encryption key, on the average that changes 35 bits of the ciphertext.
- Both effects mentioned above are referred to as the **avalanche effect**.
- And, of course, the 56-bit encryption key means a key space of size $2^{56} \approx 7.2 \times 10^{16}$. Assuming that, on the average, you'd need to try half the keys in a brute-force attack, a machine trying one key per microsecond would take 1142 years to break the code. However, a parallel-processing machine trying 1 million keys simultaneously would need only about 10 hours. (EFF took three days on a specially architected machine to break the code.)

Acknowledgement

Dustin Mitchell caught typos in the two permutation tables presented in Sections 3.12 and 3.13. Thanks Dustin!