situations. A workaround suggested by Tuchman uses just two keys for triple DES. Here, the algorithm works as follows:

1. Encrypt the plain text with key $K1$. Thus, we have $E_{K1}(P)$.
2. Decrypt the output of step 1 above with key $K2$. Thus, we have $D_{K2}(E_{K1}(P))$.
3. Finally, encrypt the output of step 2 again with key $K1$. Thus, we have $E_{K1}(D_{K2}(E_{K1}(P)))$.

This is shown in Fig. 3.44.

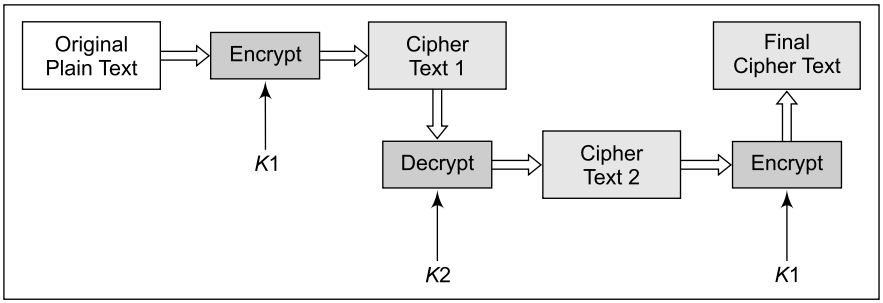

Fig. 3.44 Triple DES with two keys

To decrypt the cipher text C and obtain the original plain text P, we need to perform the operation $P = D_{K1}(E_{K2}(D_{K1}(C)))$.

There is no special meaning attached to the second step of decryption. Its only significance is that it allows triple DES to work with two, rather than three keys. This is also called **Encrypt-Decrypt-Encrypt (EDE)** mode. Triple DES with two keys is not susceptible to the *meet-in-the-middle* attack, unlike double DES as $K1$ and $K2$ alternate here.

3.5 INTERNATIONAL DATA ENCRYPTION ALGORITHM (IDEA)

3.5.1 Background and History

The **International Data Encryption Algorithm (IDEA)** is perceived as one of the strongest cryptographic algorithms. It was launched in 1990, and underwent certain changes in names and capabilities as shown in Table 3.3.

Table 3.3 Progress of IDEA

Year	Name	Description
1990	Proposed Encryption Standard (PES).	Developed by Xuejia Lai and James Massey at the Swiss Federal Institute of Technology.
1991	Improved Proposed Encryption Standard (IPES).	Improvements in the algorithm as a result of cryptanalysts finding some areas of weakness.
1992	International Data Encryption Algorithm (IDEA).	No major changes, simply renamed.

Although it is quite *strong*, IDEA is not as popular as DES for two primary reasons. Firstly, it is patented unlike DES, and therefore, must be licensed before it can be used in commercial applications. Secondly, DES has a long history and track record as compared to IDEA. However one popular email privacy technology known as **Pretty Good Privacy (PGP)** is based on IDEA.

3.5.2 How IDEA Works

1. Basic Principles

Technically, IDEA is a block cipher. Like DES, it also works on 64-bit plain-text blocks. The key is longer, however, and consists of 128 bits. IDEA is reversible like DES, that is, the same algorithm is used for encryption and decryption. Also, IDEA uses both *diffusion* and *confusion* for encryption.

The working of IDEA can be visualized at a broad level as shown in Fig. 3.45. The 64-bit input plain-text block is divided into four portions of plain text (each of size 16 bits), say $P1$ to $P4$. Thus, $P1$ to $P4$ are the inputs to the first *round* of the algorithm. There are eight such *rounds*. As we mentioned, the key consists of 128 bits. In each *round*, six subkeys are generated from the original key. Each of the subkeys consists of 16 bits. (Do not worry if you do not understand this. We shall discuss this in great detail soon). These six subkeys are applied to the four input blocks $P1$ to $P4$. Thus, for the first *round*, we will have the six keys $K1$ to $K6$. For the second *round*, we will have keys $K7$ to $K12$. Finally, for the eighth round, we will have keys $K43$ to $K48$. The final step consists of an **output transformation**, which uses just four subkeys ($K49$ to $K52$). The final output produced is the output produced by the output transformation step, which is four blocks of cipher text named $C1$ to $C4$ (each consisting of 16 bits). These are combined to form the final 64-bit cipher-text block.

2. Rounds

We have mentioned that there are 8 rounds in IDEA. Each round involves a series of operations on the four data blocks using six keys. At a broad level, these steps can be described as shown in Fig. 3.46. As we can see, these steps perform a lot of mathematical actions. There are multiplications, additions and XOR operations.

Note that we have put an asterisk in front of the words *add* and *multiply*, causing them to be shown as Add* and Multiply*, respectively. The reason behind this is that these are not mere additions and multiplications. Instead, these are addition modulo 2^{16} (i.e. addition modulo 65536) and multiplication modulo $2^{16} + 1$ (i.e. multiplication modulo 65537), respectively. [For those who are not conversant with modulo arithmetic, if a and b are two integers, then $a \bmod b$ is the remainder of the division a/b. Thus, for example, $5 \bmod 2$ is 1 (because the remainder of 5/2 is 1), and $5 \bmod 3$ is 2 (because the remainder of 5/3 is 2)].

Why is this required in IDEA, and what does this mean? Let us examine the case of the normal binary addition with an example. Suppose that we are in *round* 2 of IDEA. Further, let us assume that $P2$ = 1111111100000000 and $K2$ = 1111111111000001. First, simply add them (*and not add* them!*) and see what happens. The operation would look like as shown in Fig. 3.47.

As we can see, the normal addition will produce a number that consists of 17 bits (i.e. 11111111011000001). However, remember that we have only 16 bit positions available for the output of *round 2*. Therefore, we must reduce this number (which is 130753 in decimal) to a 16-bit number. For this, we take modulo

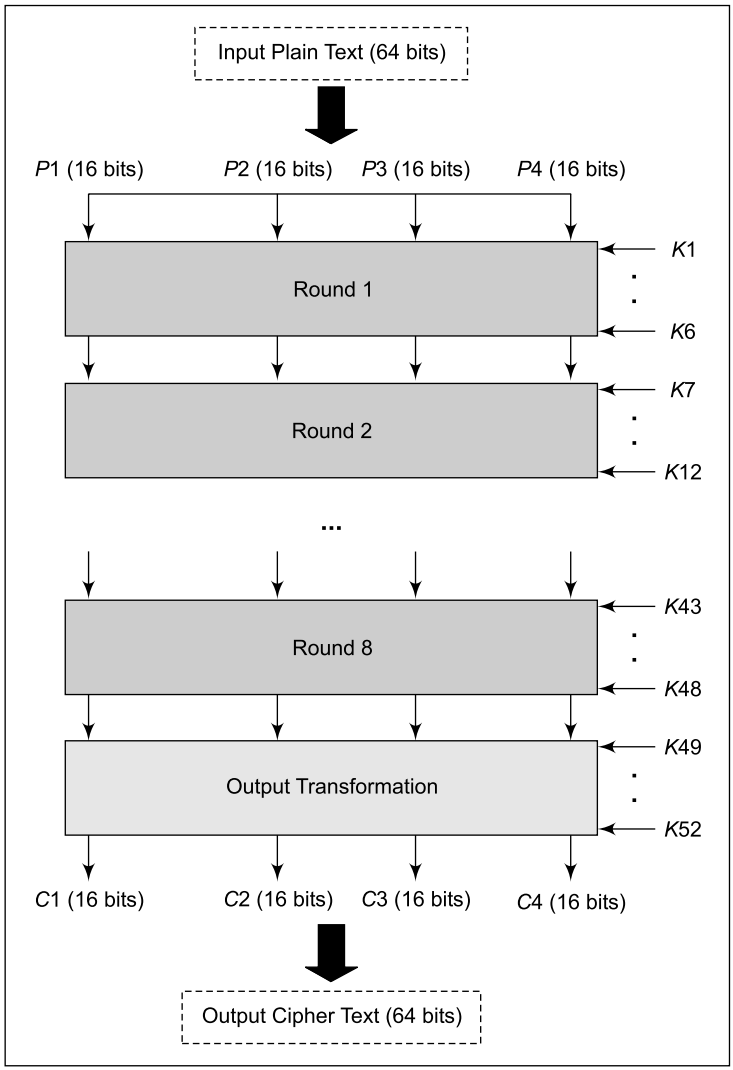

Fig. 3.45 Broad level steps in IDEA

65536 of this. *130753 modulo 65536* yields 65217, which is 1111111011000001 in binary, and is a 16-bit number, which fits well in our scheme.

This should explain why modulo arithmetic is required in IDEA. It simply ensures that even if the result of an addition or multiplication of two 16-bit numbers contains more than 17 bits, we bring it back to 16 bits.

Let us now re-look at the details of one round in a more symbolic fashion, as shown in Fig. 3.48. It conveys the same meaning as the earlier diagram, but is more symbolic than verbose in nature. We have depicted the same steps as earlier. The input blocks are shown as $P1$ to $P4$, the subkeys are denoted by $K1$ to $K6$, and the output of this step is denoted by $R1$ to $R4$ (and *not* $C1$ to $C4$, because this is *not* the

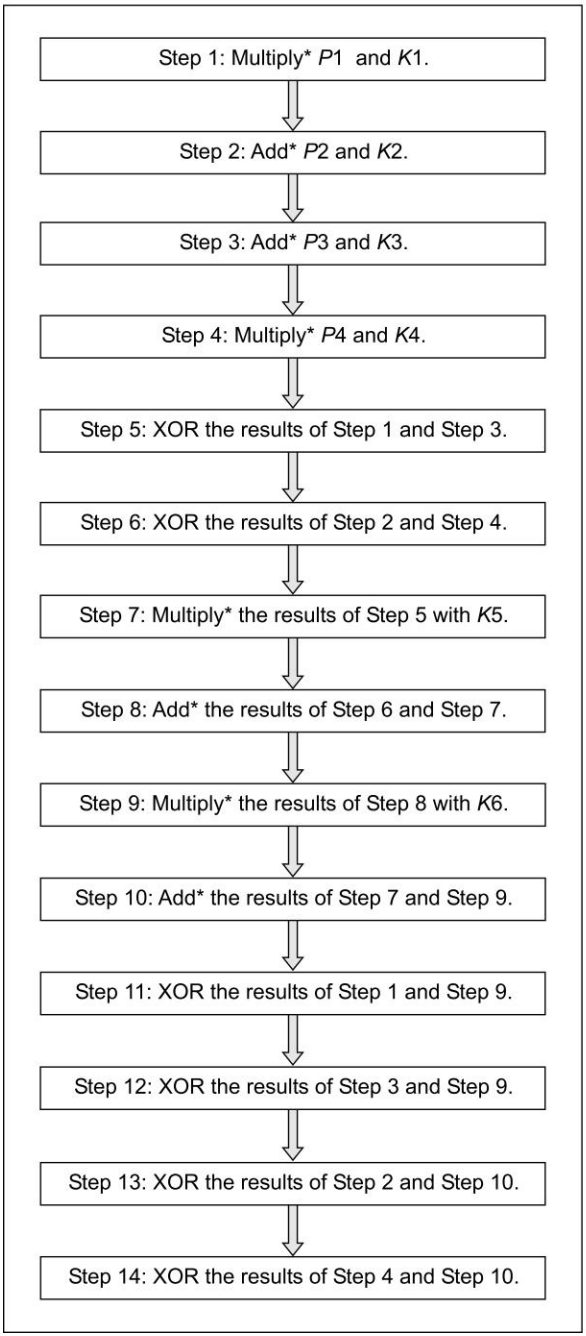

Fig. 3.46 Details of one *round* in IDEA

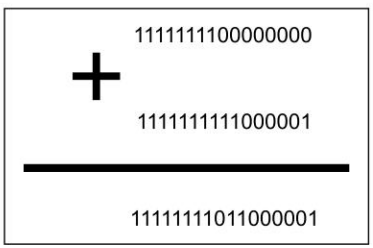

Fig. 3.47 Binary addition of two 16-bit numbers

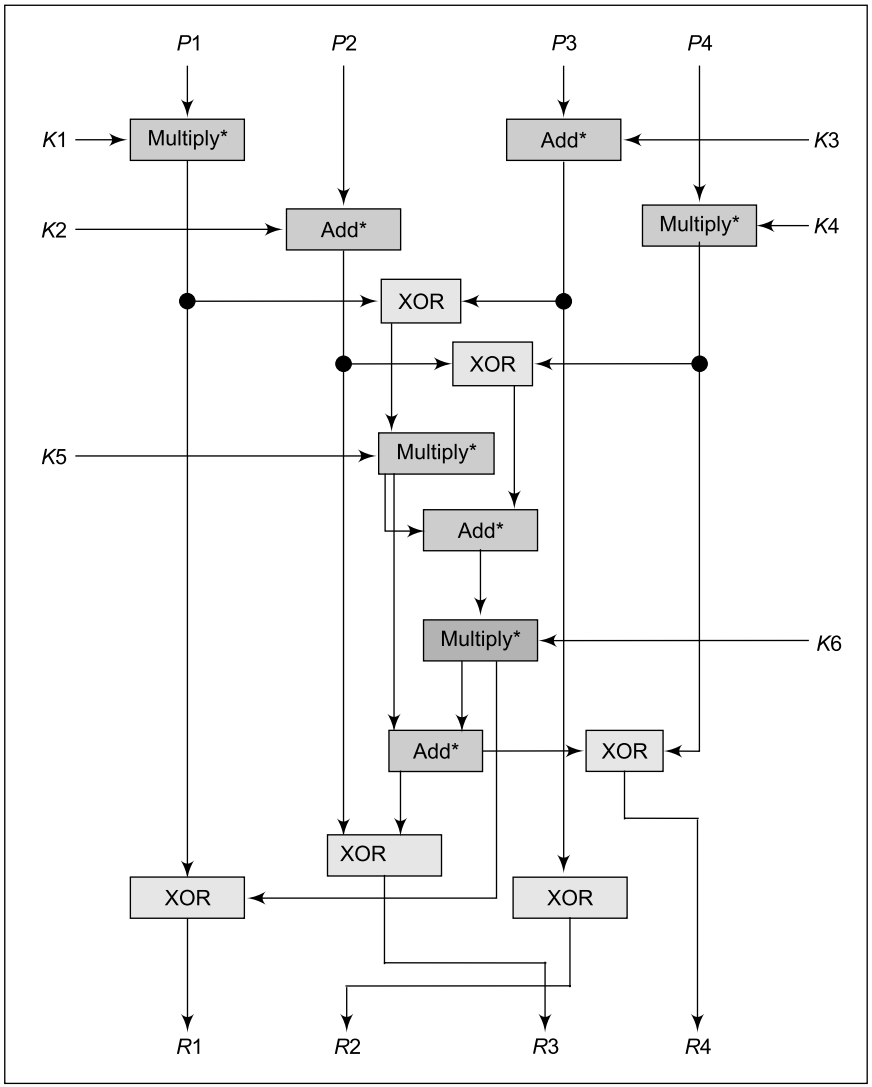

Fig. 3.48 One *round* of IDEA

final cipher text—it is an intermediate output, which will be processed in further *rounds* as well as in the *output transformation* step.

3. Subkey Generation for a *Round*

We have been talking about subkeys in our discussion quite frequently. As we mentioned, each of the eight *rounds* makes use of six subkeys (so, $8 \times 6 = 48$ subkeys are required for the *rounds*), and the final output transformation uses four subkeys (making a total of $48 + 4 = 52$ subkeys overall). From an input key of 128 bits, how are these 52 subkeys generated? Let us understand this with the explanation

for the first two rounds. Based on the understanding of the subkey generation process for the first two rounds, we will later tabulate the subkey generation for all the rounds.

(a) First *Round* We know that the initial key consists of 128 bits, from which 6 subkeys $K1$ to $K6$ are generated for the first *round*. Since $K1$ to $K6$ consist of 16 bits each, out of the original 128 bits, the first 96 bits (6 subkeys x 16 bits per subkey) are used for the first round. Thus, at the end of the first round, bits 97-–128 of the original key are unused. This is shown in Fig. 3.49.

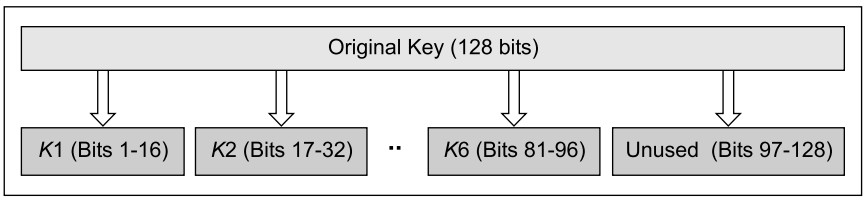

Fig. 3.49 Subkey generation for *round* 1

(b) Second *Round* As we can see, bits 1–96 are adequate to produce subkeys $K1$ to $K6$ for round 1. For the second round, we can utilize the 32 unused key bits at positions 97 to 128 (which would give us two subkeys, each of 16 bits). How do we now get the remaining 64 bits for the second round? For this, IDEA employs the technique of **key shifting**. At this stage, the original key is *shifted left circularly* by 25 bits. That is, the 26th bit of the original key moves to the first position, and becomes the first bit after the shift, and the 25th bit of the original key moves to the last position, and becomes the 128th bit after the shift. The whole process is shown in Fig. 3.50.

We can now imagine that the unused bits of the second *round* (i.e. bit positions 65-128) will firstly be used in *round* 3, and then a circular-left shift of 25 bits will be performed once again. From the resulting shifted key, we would extract the first 32 bits, which covers the shortfall in the key bits for this round. This process would go on for the remaining rounds on similar line. These procedures for all the 8 rounds can be tabulated as shown in Table 3.4.

4. Output Transformation

The **output transformation** is a one-time operation. It takes place at the end of the 8th *round*. The input to the output transformation is, of course, the output of the 8th *round*. This is, as usual, a 64-bit value divided into four sub-blocks (say $R1$ to $R4$, each consisting of 16 bits). Also, four subkeys are applied here, and not six. We will describe the key-generation process for the output transformation later. For now, we shall assume that four 16-bit subkeys $K1$ to $K4$ are available to the output transformation. The process of the output transformation is described in Fig. 3.51.

This process can be shown diagrammatically as illustrated by Fig. 3.52. The output of this process is the final 64-bit cipher text, which is a combination of the four cipher-text sub-blocks $C1$ to $C4$.

5. Subkey Generation for the Output Transformation

The process for the subkey generation for the output transformation is exactly similar to the subkey generation process for the eight *rounds*. Recall that at the end of the eighth and the final *round*, the key is exhausted and shifted. Therefore, in this round, the first 64 bits make up subkeys $K1$ to $K4$, which are used as the four subkeys for this round.

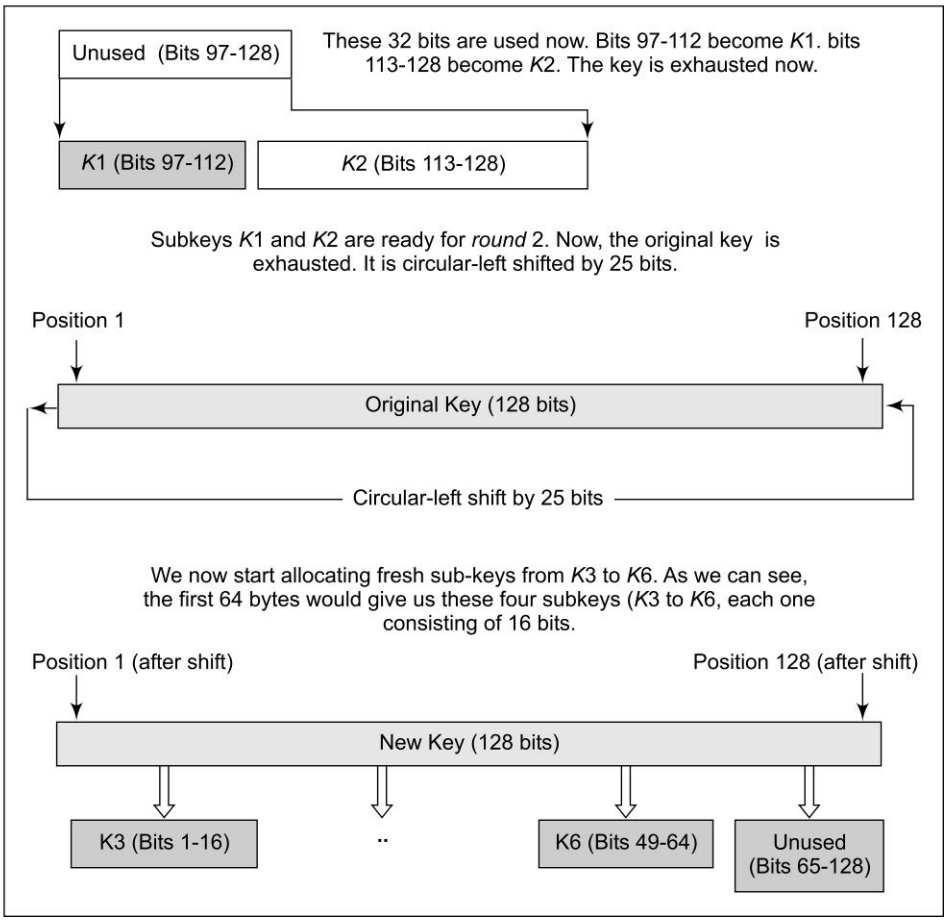

Fig. 3.50 Circular-left key shift and its use in subkey generation for *round* 2

6. *IDEA Decryption*

The decryption process is exactly the same as the encryption process. There are some alterations in the generation and pattern of subkeys. The decryption subkeys are actually an inverse of the encryption subkeys. We shall not discuss this any further, as the basic concepts remain the same.

7. *The Strength of IDEA*

IDEA uses a 128-bit key, which is double than the key size of DES. Thus, to break into IDEA, 2^{128} (i.e. 10^{38}) encryption operations would be required. As before, even if we assume that to obtain the correct key, only half of the possible keys (i.e. half of the *key space*) needs to be examined and tried out, a single computer performing one IDEA encryption per microsecond would require more than 5400000000000000000000000 years to break IDEA!

Table 3.4 Subkey generation process for each *round*

Round	Details of the subkey generation and use
1	Bit positions 1-96 of the initial 128-bit key would be used. This would give us 6 subkeys *K*1 to *K*6 for round 1. Key bits 97 to 128 are available for the next round.
2	Key bits 97 to 128 make up subkeys *K*1 and *K*2 for this round. A 25-bit shift on the original key happens, as explained. Post this shifting; the first 64 bits are used as subkeys *K*3 to *K*6 for this round. This leaves bits 65 to 128 unused for the next round.
3	Unused key bits 65 to 128 are used as subkeys *K*1 to *K*4 of this round. Upon key exhaustion, another 25-bit shift happens, and bits 1 to 32 of the shifted key are used as subkeys *K*5 and *K*6. This leaves bits 33 to 128 unused for the next round.
4	Bits 33 to 128 are used for this round, which is perfectly adequate. No bits are unused at this stage. After this, the current key is again shifted.
5	This is similar to round 1. Bit positions 1-96 of the current 128-bit key would be used. This would give us 6 subkeys *K*1 to *K*6 for round 1. Key bits 97 to 128 are available for the next round.
6	Key bits 97 to 128 make up subkeys *K*1 and *K*2 for this round. A 25-bit shift on the original key happens, as explained. Post this shifting; the first 64 bits are used as subkeys *K*3 to *K*6 for this round. This leaves bits 65 to 128 unused for the next round.
7	Unused key bits 65 to 128 are used as subkeys *K*1 to *K*4 of this round. Upon key exhaustion, another 25-bit shift happens, and bits 1 to 32 of the shifted key are used as subkeys *K*5 and *K*6. This leaves bits 33 to 128 unused for the next round.
8	Bits 33 to 128 are used for this round, which is perfectly adequate. No bits are unused at this stage. After this, the current key is again shifted for the *output transformation* round.

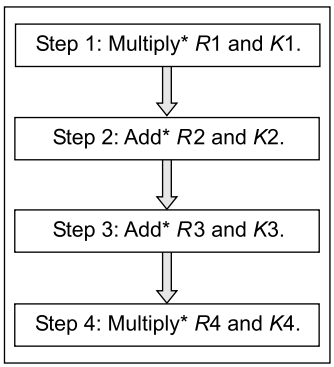

Fig. 3.51 Details of the output transformation

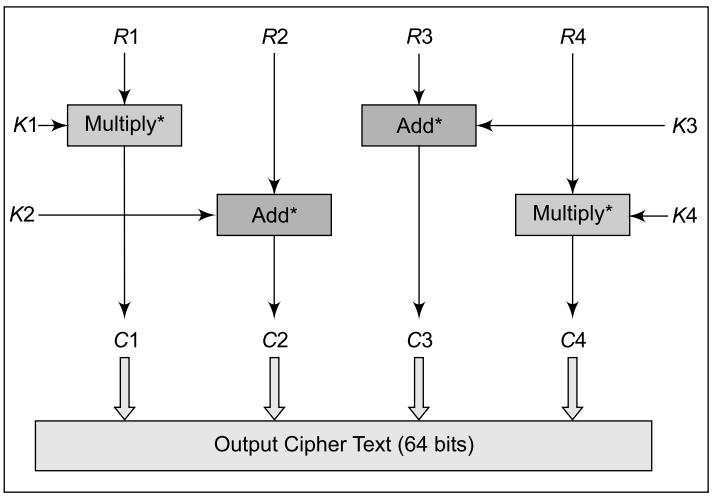

Fig. 3.52 Output transformation process