Chapter 4

Block Ciphers and Stream Ciphers

This chapter discusses various aspects of block and stream ciphers. We introduce the substitution-permutation network as a design technique for block ciphers and we discuss some standard attacks. We look at standards such as the *Data Encryption Standard* and *Advanced Encryption Standard*. Modes of operation are discussed and we also provide a brief treatment of stream ciphers.

4.1 Introduction

Most modern-day block ciphers incorporate a sequence of permutation and substitution operations. A commonly used design is that of an ***iterated cipher***. An iterated cipher requires the specification of a ***round function*** and a ***key schedule***, and the encryption of a plaintext will proceed through $\mathcal{N}$ similar ***rounds***.

Let K be a random binary key of some specified length. K is used to construct $\mathcal{N}$ ***round keys*** (also called ***subkeys***), which are denoted $K^1, \ldots, K^{\mathcal{N}}$. The list of round keys, $(K^1, \ldots, K^{\mathcal{N}})$, is the key schedule. The key schedule is constructed from K using a fixed, public algorithm.

The round function, say g, takes two inputs: a round key (K^r) and a current ***state*** (which we denote w^{r-1}). The next state is defined as $w^r = g(w^{r-1}, K^r)$. The initial state, w^0, is defined to be the plaintext, x. The ciphertext, y, is defined to be the state after all $\mathcal{N}$ rounds have been performed. Therefore, the encryption operation is carried out as follows:

$$\begin{aligned} w^0 &\leftarrow x \\ w^1 &\leftarrow g(w^0, K^1) \\ w^2 &\leftarrow g(w^1, K^2) \\ &\vdots \quad \vdots \quad \vdots \\ w^{\mathcal{N}-1} &\leftarrow g(w^{\mathcal{N}-2}, K^{\mathcal{N}-1}) \\ w^{\mathcal{N}} &\leftarrow g(w^{\mathcal{N}-1}, K^{\mathcal{N}}) \\ y &\leftarrow w^{\mathcal{N}}. \end{aligned}$$

In order for decryption to be possible, the function g must have the property that it is injective (i.e., one-to-one) if its second argument is fixed. This is equivalent

to saying that there exists a function g^{-1} with the property that

$$g^{-1}(g(w,y),y) = w$$

for all w and y. Then decryption can be accomplished as follows:

$$\begin{aligned} w^{\mathcal{N}} &\leftarrow y \\ w^{\mathcal{N}-1} &\leftarrow g^{-1}(w^{\mathcal{N}}, K^{\mathcal{N}}) \\ &\vdots \quad \vdots \quad \vdots \\ w^1 &\leftarrow g^{-1}(w^2, K^2) \\ w^0 &\leftarrow g^{-1}(w^1, K^1) \\ x &\leftarrow w^0. \end{aligned}$$

In Section 4.2, we describe a simple type of iterated cipher, the substitution-permutation network, which illustrates many of the main principles used in the design of practical block ciphers. Linear and differential attacks on substitution-permutation networks are described in Sections 4.3 and 4.4, respectively. In Section 4.5, we discuss Feistel-type ciphers and the *Data Encryption Standard*. In Section 4.6, we present the *Advanced Encryption Standard*. Finally, modes of operation of block ciphers are the topic of Section 4.7 and stream ciphers are discussed in Section 4.8.

4.2 Substitution-Permutation Networks

We begin by defining a ***substitution-permutation network***, or ***SPN***. (An SPN is a special type of iterated cipher with a couple of small changes that we will indicate.) Suppose that ℓ and m are positive integers. A plaintext and ciphertext will both be binary vectors of length ℓm (i.e., ℓm is the ***block length*** of the cipher). An SPN is built from two components, which are denoted π_S and π_P.

$$\pi_S : \{0,1\}^\ell \to \{0,1\}^\ell$$

is a permutation of the 2^ℓ bitstrings of length ℓ, and

$$\pi_P : \{1,\ldots,\ell m\} \to \{1,\ldots,\ell m\}$$

is also a permutation, of the integers $1,\ldots,\ell m$. The permutation π_S is called an ***S-box*** (the letter "S" denotes "substitution"). It is used to replace ℓ bits with a different set of ℓ bits. π_P, on the other hand, is used to permute ℓm bits by changing their order.

Given an ℓm-bit binary string, say $x = (x_1,\ldots,x_{\ell m})$, we can regard x as the concatenation of m ℓ-bit substrings, which we denote $x_{<1>},\ldots,x_{<m>}$. Thus

$$x = x_{<1>} \| \ldots \| x_{<m>}$$

Cryptosystem 4.1: *Substitution-Permutation Network*

Let ℓ, m, and $\mathcal{N}$ be positive integers, let $\pi_S : \{0,1\}^\ell \to \{0,1\}^\ell$ be a permutation, and let $\pi_P : \{1,\ldots,\ell m\} \to \{1,\ldots,\ell m\}$ be a permutation. Let $\mathcal{P} = \mathcal{C} = \{0,1\}^{\ell m}$, and let $\mathcal{K} \subseteq (\{0,1\}^{\ell m})^{\mathcal{N}+1}$ consist of all possible key schedules that could be derived from an initial key K using the key scheduling algorithm. For a key schedule $(K^1,\ldots,K^{\mathcal{N}+1})$, we encrypt the plaintext x using Algorithm 4.1.

and for $1 \le i \le m$, we have that

$$x_{<i>} = \left(x_{(i-1)\ell+1},\ldots,x_{i\ell}\right).$$

The SPN will consist of $\mathcal{N}$ rounds. In each round (except for the last round, which is slightly different), we will perform m substitutions using π_S, followed by a permutation using π_P. Prior to each substitution operation, we will incorporate round key bits via a simple exclusive-or operation. We now present an SPN, based on π_S and π_P, as Cryptosystem 4.1.

In Algorithm 4.1, u^r is the input to the S-boxes in round r, and v^r is the output of the S-boxes in round r. w^r is obtained from v^r by applying the permutation π_P, and then u^{r+1} is constructed from w^r by x-or-ing with the round key K^{r+1} (this is called ***round key mixing***). In the last round, the permutation π_P is not applied. As a consequence, the encryption algorithm can also be used for decryption, if appropriate modifications are made to the key schedule and the S-boxes are replaced by their inverses (see the Exercises).

Notice that the very first and last operations performed in this SPN are x-ors with subkeys. This is called ***whitening***, and it is regarded as a useful way to prevent an attacker from even beginning to carry out an encryption or decryption operation if the key is not known.

We illustrate the above general description with a particular SPN.

Example 4.1 Suppose that $\ell = m = \mathcal{N} = 4$. Let π_S be defined as follows, where the input (i.e., z) and the output (i.e., $\pi_S(z)$) are written in hexadecimal notation, ($0 \leftrightarrow (0,0,0,0)$, $1 \leftrightarrow (0,0,0,1)$, $\ldots$, $9 \leftrightarrow (1,0,0,1)$, $A \leftrightarrow (1,0,1,0)$, $\ldots$, $F \leftrightarrow (1,1,1,1)$):

z	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
$\pi_S(z)$	E	4	D	1	2	F	B	8	3	A	6	C	5	9	0	7

Further, let π_P be defined as follows:

z	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
$\pi_P(z)$	1	5	9	13	2	6	10	14	3	7	11	15	4	8	12	16

See Figure 4.1 for a pictorial representation of this particular SPN. (In this diagram,

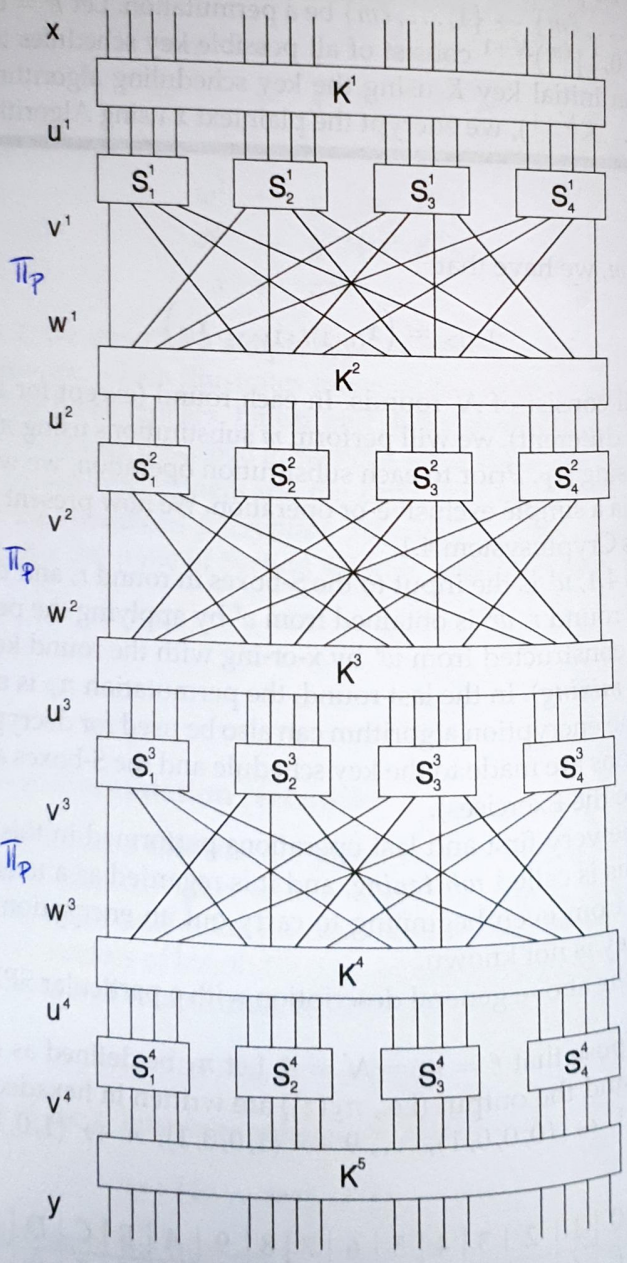

FIGURE 4.1: A substitution-permutation network

Algorithm 4.1: $\mathrm{SPN}(x, \pi_S, \pi_P, (K^1, \ldots, K^{\mathcal{N}+1}))$

$w^0 \leftarrow x$
for $r \leftarrow 1$ **to** $\mathcal{N} - 1$
do
$u^r \leftarrow w^{r-1} \oplus K^r$
for $i \leftarrow 1$ **to** m
do $v^r_{<i>} \leftarrow \pi_S(u^r_{<i>})$
$w^r \leftarrow (v^r_{\pi_P(1)}, \ldots, v^r_{\pi_P(\ell m)})$
$u^{\mathcal{N}} \leftarrow w^{\mathcal{N}-1} \oplus K^{\mathcal{N}}$
for $i \leftarrow 1$ **to** m
do $v^{\mathcal{N}}_{<i>} \leftarrow \pi_S(u^{\mathcal{N}}_{<i>})$
$y \leftarrow v^{\mathcal{N}} \oplus K^{\mathcal{N}+1}$
output (y)

we have named the S-boxes S^r_i $(1 \leq i \leq 4, 1 \leq r \leq 4)$ for ease of later reference. All 16 S-boxes incorporate the same substitution function based on π_S.)

In order to complete the description of the SPN, we need to specify a key scheduling algorithm. Here is a simple possibility: suppose that we begin with a 32-bit key $K = (k_1, \ldots, k_{32}) \in \{0,1\}^{32}$. For $1 \leq r \leq 5$, define K^r to consist of 16 consecutive bits of K, beginning with k_{4r-3}. (This is not a very secure way to define a key schedule; we have just chosen something easy for purposes of illustration.)

Now let's work out a sample encryption using this SPN. We represent all data in binary notation. Suppose the key is

$$K = 0011\ 1010\ 1001\ 0100\ 1101\ 0110\ 0011\ 1111.$$

Then the round keys are as follows:

$$\begin{aligned} K^1 &= 0011\ 1010\ 1001\ 0100 \\ K^2 &= 1010\ 1001\ 0100\ 1101 \\ K^3 &= 1001\ 0100\ 1101\ 0110 \\ K^4 &= 0100\ 1101\ 0110\ 0011 \\ K^5 &= 1101\ 0110\ 0011\ 1111. \end{aligned}$$

Suppose that the plaintext is

$$x = 0010\ 0110\ 1011\ 0111.$$

Then the encryption of x proceeds as shown in Figure 4.2, yielding the ciphertext y.

□

$$
\begin{aligned}
w^0 &= 0010\ 0110\ 1011\ 0111 \\
K^1 &= 0011\ 1010\ 1001\ 0100 \\
u^1 &= 0001\ 1100\ 0010\ 0011 \\
v^1 &= 0100\ 0101\ 1101\ 0001 \\
w^1 &= 0010\ 1110\ 0000\ 0111 \\
K^2 &= 1010\ 1001\ 0100\ 1101 \\
u^2 &= 1000\ 0111\ 0100\ 1010 \\
v^2 &= 0011\ 1000\ 0010\ 0110 \\
w^2 &= 0100\ 0001\ 1011\ 1000 \\
K^3 &= 1001\ 0100\ 1101\ 0110 \\
u^3 &= 1101\ 0101\ 0110\ 1110 \\
v^3 &= 1001\ 1111\ 1011\ 0000 \\
w^3 &= 1110\ 0100\ 0110\ 1110 \\
K^4 &= 0100\ 1101\ 0110\ 0011 \\
u^4 &= 1010\ 1001\ 0000\ 1101 \\
v^4 &= 0110\ 1010\ 1110\ 1001 \\
K^5 &= 1101\ 0110\ 0011\ 1111 \\
y &= 1011\ 1100\ 1101\ 0110
\end{aligned}
$$

FIGURE 4.2: Encryption using a substitution-permutation network

SPNs have several attractive features. First, the design is simple and very efficient, in both hardware and software. In software, an S-box is usually implemented in the form of a look-up table. Observe that the memory requirement of the S-box $\pi_S : \{0,1\}^\ell \rightarrow \{0,1\}^\ell$ is $\ell 2^\ell$ bits, since we have to store 2^ℓ values, each of which needs ℓ bits of storage. Hardware implementations, in particular, necessitate the use of relatively small S-boxes.

In Example 4.1, we used four identical S-boxes in each round. The memory requirement of the S-box is $2^4 \times 4 = 2^6$ bits. If we instead used one S-box that mapped 16 bits to 16 bits, the memory requirement would be increased to 2^{20} bits, which would be prohibitively high for some applications. The S-box $16 = 2^{20}$ used in the *Advanced Encryption Standard* (to be discussed in Section 4.6) maps eight bits to eight bits.

The SPN in Example 4.1 is not secure, if for no other reason than the key length (32 bits) is small enough that an exhaustive key search is feasible. However, "larger" SPNs can be designed that are secure against all known attacks. A practical, secure SPN would have a larger key size and block length than Example 4.1, would most likely use larger S-boxes, and would have more rounds. Rijndael, which was chosen to be the *Advanced Encryption Standard*, is an example of an

SPN that is similar to Example 4.1 in many respects. *Rijndael* has a minimum key size of 128 bits, a block length of 128, a minimum of 10 rounds; and its S-box maps eight bits to eight bits (see Section 4.6 for a complete description).

Many variations of SPNs are possible. One common modification would be to use more than one S-box. In Example, 4.1, we could use four different S-boxes in each round if we so desired, instead of using the same S-box four times. This feature can be found in the *Data Encryption Standard*, which employs eight different S-boxes in each round (see Section 4.5.1). Another popular design strategy is to include an invertible linear transformation in each round, either as a replacement for, or in addition to, the permutation operation. This is done in the *Advanced Encryption Standard* (see Section 4.6.1).

4.3 Linear Cryptanalysis

We begin by informally describing the strategy behind linear cryptanalysis. The idea can be applied, in principle, to any iterated cipher. Suppose that it is possible to find a probabilistic linear relationship between a subset of plaintext bits and a subset of state bits immediately preceding the substitutions performed in the last round. In other words, there exists a subset of bits whose exclusive-or behaves in a non-random fashion (it takes on the value 0, say, with probability bounded away from $1/2$). Now assume that an attacker has a large number of plaintext-ciphertext pairs, all of which are encrypted using the same unknown key K (i.e., we consider a known-plaintext attack). For each of the plaintext-ciphertext pairs, we will begin to decrypt the ciphertext, using all possible candidate keys for the last round of the cipher. For each candidate key, we compute the values of the relevant state bits involved in the linear relationship, and determine if the above-mentioned linear relationship holds. Whenever it does, we increment a counter corresponding to the particular candidate key. At the end of this process, we hope that the candidate key that has a frequency count furthest from $1/2$ times the number of plaintext-ciphertext pairs contains the correct values for these key bits.

We will illustrate the above description with a detailed example later in this section. First, we need to establish some results from probability theory to provide a (non-rigorous) justification for the techniques involved in the attack.

4.3.1 The Piling-up Lemma

We use terminology and concepts introduced in Section 3.2. Suppose that $X_1, X_2, \ldots$ are independent random variables taking on values from the set $\{0, 1\}$. Let $p_1, p_2, \ldots$ be real numbers such that $0 \leq p_i \leq 1$ for all i, and suppose that

$$\mathbf{Pr}[X_i = 0] = p_i,$$

$i = 1, 2, \ldots$. Hence,

$$\mathbf{Pr}[X_i = 1] = 1 - p_i,$$

$i = 1, 2, \ldots$.

Suppose that $i \neq j$. The independence of $\mathbf{X_i}$ and $\mathbf{X_j}$ implies that

$$\begin{aligned}
\mathbf{Pr}[\mathbf{X_i} = 0, \mathbf{X_j} = 0] &= p_i p_j \\
\mathbf{Pr}[\mathbf{X_i} = 0, \mathbf{X_j} = 1] &= p_i(1 - p_j) \\
\mathbf{Pr}[\mathbf{X_i} = 1, \mathbf{X_j} = 0] &= (1 - p_i)p_j, \quad \text{and} \\
\mathbf{Pr}[\mathbf{X_i} = 1, \mathbf{X_j} = 1] &= (1 - p_i)(1 - p_j).
\end{aligned}$$

Now consider the discrete random variable $\mathbf{X_i} \oplus \mathbf{X_j}$ (this is the same thing as $\mathbf{X_i} + \mathbf{X_j} \bmod 2$). It is easy to see that $\mathbf{X_i} \oplus \mathbf{X_j}$ has the following probability distribution:

$$\begin{aligned}
\mathbf{Pr}[\mathbf{X_i} \oplus \mathbf{X_j} = 0] &= p_i p_j + (1 - p_i)(1 - p_j) \\
\mathbf{Pr}[\mathbf{X_i} \oplus \mathbf{X_j} = 1] &= p_i(1 - p_j) + (1 - p_i)p_j.
\end{aligned}$$

It is often convenient to express a probability distribution of a random variable taking on the values 0 and 1 in terms of a quantity called the bias of the distribution. The ***bias*** of $\mathbf{X_i}$ is defined to be the quantity

$$\epsilon_i = p_i - \frac{1}{2}.$$

Observe the following facts:

$$-\frac{1}{2} \leq \epsilon_i \leq \frac{1}{2},$$

$$\mathbf{Pr}[\mathbf{X_i} = 0] = \frac{1}{2} + \epsilon_i, \text{and}$$

$$\mathbf{Pr}[\mathbf{X_i} = 1] = \frac{1}{2} - \epsilon_i,$$

for $i = 1, 2, \ldots$.

The following result, which gives a formula for the bias of the random variable $\mathbf{X_{i_1}} \oplus \cdots \oplus \mathbf{X_{i_k}}$, is known as the ***piling-up lemma***.

LEMMA 4.1 (Piling-up lemma) *Let $\epsilon_{i_1,i_2,\ldots,i_k}$ denote the bias of the random variable $\mathbf{X_{i_1}} \oplus \cdots \oplus \mathbf{X_{i_k}}$. Then*

$$\epsilon_{i_1,i_2,\ldots,i_k} = 2^{k-1} \prod_{j=1}^{k} \epsilon_{i_j}.$$

PROOF The proof is by induction on k. Clearly the result is true when $k = 1$. We next prove the result for $k = 2$, where we want to determine the bias of $\mathbf{X_{i_1}} \oplus \mathbf{X_{i_2}}$. Using the equations presented above, we have that

$$\begin{aligned}
\mathbf{Pr}[\mathbf{X_{i_1}} \oplus \mathbf{X_{i_2}} = 0] &= \left(\frac{1}{2} + \epsilon_{i_1}\right)\left(\frac{1}{2} + \epsilon_{i_2}\right) + \left(\frac{1}{2} - \epsilon_{i_1}\right)\left(\frac{1}{2} - \epsilon_{i_2}\right) \\
&= \frac{1}{2} + 2\epsilon_{i_1}\epsilon_{i_2}.
\end{aligned}$$

Hence, the bias of $\mathbf{X}_{i_1} \oplus \mathbf{X}_{i_2}$ is $2\epsilon_{i_1}\epsilon_{i_2}$, as claimed.

Now, as an induction hypothesis, assume that the result is true for $k = \ell$, for some positive integer $\ell \geq 2$. We will prove that the formula is true for $k = \ell + 1$.

We want to determine the bias of $\mathbf{X}_{i_1} \oplus \cdots \oplus \mathbf{X}_{i_{\ell+1}}$. We split this random variable into two parts, as follows:

$$\mathbf{X}_{i_1} \oplus \cdots \oplus \mathbf{X}_{i_{\ell+1}} = (\mathbf{X}_{i_1} \oplus \cdots \oplus \mathbf{X}_{i_\ell}) \oplus \mathbf{X}_{i_{\ell+1}}.$$

The bias of $\mathbf{X}_{i_1} \oplus \cdots \oplus \mathbf{X}_{i_\ell}$ is $2^{\ell-1}\prod_{j=1}^{\ell}\epsilon_{i_j}$ (by induction) and the bias of $\mathbf{X}_{i_{\ell+1}}$ is $\epsilon_{i_{\ell+1}}$. Then, by induction (more specifically, using the formula for $k = 2$), the bias of $\mathbf{X}_{i_1} \oplus \cdots \oplus \mathbf{X}_{i_{\ell+1}}$ is

$$2 \times \left(2^{\ell-1}\prod_{j=1}^{\ell}\epsilon_{i_j}\right) \times \epsilon_{i_{\ell+1}} = 2^{\ell}\prod_{j=1}^{\ell+1}\epsilon_{i_j},$$

as desired. By induction, the proof is complete. ∎

COROLLARY 4.2 *Let $\epsilon_{i_1,i_2,\ldots,i_k}$ denote the bias of the random variable $\mathbf{X}_{i_1} \oplus \cdots \oplus \mathbf{X}_{i_k}$. Suppose that $\epsilon_{i_j} = 0$ for some j. Then $\epsilon_{i_1,i_2,\ldots,i_k} = 0$.*

It is important to realize that Lemma 4.1 holds, in general, only when the relevant random variables are independent. We illustrate this by considering an example. Suppose that $\epsilon_1 = \epsilon_2 = \epsilon_3 = 1/4$. Applying Lemma 4.1, we see that $\epsilon_{1,2} = \epsilon_{2,3} = \epsilon_{1,3} = 1/8$. Now, consider the random variable $\mathbf{X_1} \oplus \mathbf{X_3}$. It is clear that

$$\mathbf{X_1} \oplus \mathbf{X_3} = (\mathbf{X_1} \oplus \mathbf{X_2}) \oplus (\mathbf{X_2} \oplus \mathbf{X_3}).$$

If the two random variables $\mathbf{X_1} \oplus \mathbf{X_2}$ and $\mathbf{X_2} \oplus \mathbf{X_3}$ were independent, then Lemma 4.1 would say that $\epsilon_{1,3} = 2(1/8)^2 = 1/32$. However, we already know that this is not the case: $\epsilon_{1,3} = 1/8$. Lemma 4.1 does not yield the correct value of $\epsilon_{1,3}$ because $\mathbf{X_1} \oplus \mathbf{X_2}$ and $\mathbf{X_2} \oplus \mathbf{X_3}$ are not independent.

4.3.2 Linear Approximations of S-boxes

Consider an S-box $\pi_S : \{0,1\}^m \to \{0,1\}^n$. (We do not assume that π_S is a permutation, or even that $m = n$.) Let us write an input m-tuple as $X = (x_1, \ldots, x_m)$. This m-tuple is chosen uniformly at random from $\{0,1\}^m$, which means that each co-ordinate x_i defines a random variable $\mathbf{X_i}$ taking on values 0 and 1, having bias $\epsilon_i = 0$. Further, these m random variables are independent.

Now write an output n-tuple as $Y = (y_1, \ldots, y_n)$. Each co-ordinate y_j defines a random variable $\mathbf{Y_j}$ taking on values 0 and 1. These n random variables are, in general, not independent from each other or from the $\mathbf{X_i}$'s. In fact, it is not hard to see that the following formula holds:

$$\mathbf{Pr}[\mathbf{X_1} = x_1, \ldots, \mathbf{X_m} = x_m, \mathbf{Y_1} = y_1, \ldots, \mathbf{Y_n} = y_n] = 0$$

TABLE 4.1: Random variables defined by an S-box

X_1	X_2	X_3	X_4	Y_1	Y_2	Y_3	Y_4
0	0	0	0	1	1	1	0
0	0	0	1	0	1	0	0
0	0	1	0	1	1	0	1
0	0	1	1	0	0	0	1
0	1	0	0	0	0	1	0
0	1	0	1	1	1	1	1
0	1	1	0	1	0	1	1
0	1	1	1	1	0	0	0
1	0	0	0	0	0	1	1
1	0	0	1	1	0	1	0
1	0	1	0	0	1	1	0
1	0	1	1	1	1	0	0
1	1	0	0	0	1	0	1
1	1	0	1	1	0	0	1
1	1	1	0	0	0	0	0
1	1	1	1	0	1	1	1

if $(y_1, \ldots, y_n) \neq \pi_S(x_1, \ldots, x_m)$; and

$$\mathbf{Pr}[\mathbf{X_1} = x_1, \ldots, \mathbf{X_m} = x_m, \mathbf{Y_1} = y_1, \ldots, \mathbf{Y_n} = y_n] = 2^{-m}$$

if $(y_1, \ldots, y_n) = \pi_S(x_1, \ldots, x_m)$. (The last formula holds because

$$\mathbf{Pr}[\mathbf{X_1} = x_1, \ldots, \mathbf{X_m} = x_m] = 2^{-m}$$

and

$$\mathbf{Pr}[\mathbf{Y_1} = y_1, \ldots, \mathbf{Y_n} = y_n | \mathbf{X_1} = x_1, \ldots, \mathbf{X_m} = x_m] = 1$$

if $(y_1, \ldots, y_n) = \pi_S(x_1, \ldots, x_m)$.)

It is now relatively straightforward to compute the bias of a random variable of the form

$$\mathbf{X}_{i_1} \oplus \cdots \oplus \mathbf{X}_{i_\ell} \oplus \mathbf{Y}_{j_1} \oplus \cdots \oplus \mathbf{Y}_{j_\ell}$$

using the formulas stated above. (A linear cryptanalytic attack can potentially be mounted when a random variable of this form has a bias that is bounded away from zero.)

Let's consider a small example.

Example 4.2 We use the S-box from Example 4.1, which is defined by a permutation $\pi_S : \{0,1\}^4 \to \{0,1\}^4$. We record the possible values taken on by the eight random variables $\mathbf{X_1}, \ldots, \mathbf{X_4}, \mathbf{Y_1}, \ldots, \mathbf{Y_4}$ in the rows of Table 4.1.

Now, consider the random variable $\mathbf{X_1} \oplus \mathbf{X_4} \oplus \mathbf{Y_2}$. The probability that this

random variable takes on the value 0 can be determined by counting the number of rows in the above table in which $\mathbf{X_1} \oplus \mathbf{X_4} \oplus \mathbf{Y_2} = 0$, and then dividing by 16 ($16 = 2^4$ is the total number of rows in the table). It is seen that

$$\mathbf{Pr}[\mathbf{X_1} \oplus \mathbf{X_4} \oplus \mathbf{Y_2} = 0] = \frac{1}{2}$$

(and therefore

$$\mathbf{Pr}[\mathbf{X_1} \oplus \mathbf{X_4} \oplus \mathbf{Y_2} = 1] = \frac{1}{2},$$

as well.) Hence, the bias of this random variable is 0. □

If we instead analyzed the random variable $\mathbf{X_3} \oplus \mathbf{X_4} \oplus \mathbf{Y_1} \oplus \mathbf{Y_4}$, we would find that the bias is $-3/8$. (We suggest that the reader verify this computation.) Indeed, it is not difficult to compute the biases of all $2^8 = 256$ possible random variables of this form.

We record this information using the following notation. We represent each of the relevant random variables in the form

$$\left(\bigoplus_{i=1}^{4} a_i \mathbf{X_i}\right) \oplus \left(\bigoplus_{i=1}^{4} b_i \mathbf{Y_i}\right),$$

where $a_i \in \{0,1\}$, $b_i \in \{0,1\}$, $i = 1,2,3,4$. Then, in order to have a compact notation, we treat each of the binary vectors (a_1, a_2, a_3, a_4) and (b_1, b_2, b_3, b_4) as a hexadecimal digit (these are called the ***input sum*** and ***output sum***, respectively). In this way, each of the 256 random variables is named by a (unique) pair of hexadecimal digits, representing the input and output sum.

As an example, consider the random variable $\mathbf{X_1} \oplus \mathbf{X_4} \oplus \mathbf{Y_2}$. The input sum is $(1,0,0,1)$, which is 9 in hexadecimal; the output sum is $(0,1,0,0)$, which is 4 in hexadecimal.

Definition 4.1: For a random variable having (hexadecimal) input sum a and output sum b (where $a = (a_1, a_2, a_3, a_4)$ and $b = (b_1, b_2, b_3, b_4)$, in binary), let $N_L(a,b)$ denote the number of binary eight-tuples $(x_1, x_2, x_3, x_4, y_1, y_2, y_3, y_4)$ such that

$$(y_1, y_2, y_3, y_4) = \pi_S(x_1, x_2, x_3, x_4)$$

and

$$\left(\bigoplus_{i=1}^{4} a_i x_i\right) \oplus \left(\bigoplus_{i=1}^{4} b_i y_i\right) = 0.$$

Observe that the bias of the random variable having input sum a and output sum b is computed as $\epsilon(a,b) = (N_L(a,b) - 8)/16$.

We computed $N_L(9,4) = 8$, and hence $\epsilon(9,4) = 0$, in Example 4.2. The table of all values N_L is called the ***linear approximation table***; see Table 4.2.

TABLE 4.2: Linear approximation table: values of $N_L(a,b)$

	b															
a	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	16	8	8	8	8	8	8	8	8	8	8	8	8	8	8	8
1	8	8	6	6	8	8	6	14	10	10	8	8	10	10	8	8
2	8	8	6	6	8	8	6	6	8	8	10	10	8	8	2	10
3	8	8	8	8	8	8	8	8	10	2	6	6	10	10	6	6
4	8	10	8	6	6	4	6	8	8	6	8	10	10	4	10	8
5	8	6	6	8	6	8	12	10	6	8	4	10	8	6	6	8
6	8	10	6	12	10	8	8	10	8	6	10	12	6	8	8	6
7	8	6	8	10	10	4	10	8	6	8	10	8	12	10	8	10
8	8	8	8	8	8	8	8	8	6	10	10	6	10	6	6	2
9	8	8	6	6	8	8	6	6	4	8	6	10	8	12	10	6
A	8	12	6	10	4	8	10	6	10	10	8	8	10	10	8	8
B	8	12	8	4	12	8	12	8	8	8	8	8	8	8	8	8
C	8	6	12	6	6	8	10	8	10	8	10	12	8	10	8	6
D	8	10	10	8	6	12	8	10	4	6	10	8	10	8	8	10
E	8	10	10	8	6	4	8	10	6	8	8	6	4	10	6	8
F	8	6	4	6	6	8	10	8	8	6	12	6	6	8	10	8

4.3.3 A Linear Attack on an SPN

Linear cryptanalysis requires finding a set of linear approximations of S-boxes that can be used to derive a linear approximation of the entire SPN (excluding the last round). We will illustrate the procedure using the SPN from Example 4.1. The diagram in Figure 4.3 illustrates the structure of the approximation we will use. This diagram can be interpreted as follows: Lines with arrows correspond to random variables that will be involved in linear approximations. The labeled S-boxes are the ones used in these approximations (they are called the *active S-boxes* in the approximation).

This approximation incorporates four active S-boxes:

- In S_2^1, the random variable $\mathbf{T_1} = \mathbf{U_5^1} \oplus \mathbf{U_7^1} \oplus \mathbf{U_8^1} \oplus \mathbf{V_6^1}$ has bias $1/4$
- In S_2^2, the random variable $\mathbf{T_2} = \mathbf{U_6^2} \oplus \mathbf{V_6^2} \oplus \mathbf{V_8^2}$ has bias $-1/4$
- In S_2^3, the random variable $\mathbf{T_3} = \mathbf{U_6^3} \oplus \mathbf{V_6^3} \oplus \mathbf{V_8^3}$ has bias $-1/4$
- In S_4^3, the random variable $\mathbf{T_4} = \mathbf{U_{14}^3} \oplus \mathbf{V_{14}^3} \oplus \mathbf{V_{16}^3}$ has bias $-1/4$

Note that $\mathbf{U_i^r}$ $(1 \leq i \leq 4)$ are random variables corresponding to the inputs to the S-boxes in round r, and $\mathbf{V_i^r}$ are random variables corresponding to the outputs of the same S-boxes. The four random variables $\mathbf{T_1}, \mathbf{T_2}, \mathbf{T_3}, \mathbf{T_4}$ have biases that are high in absolute value.

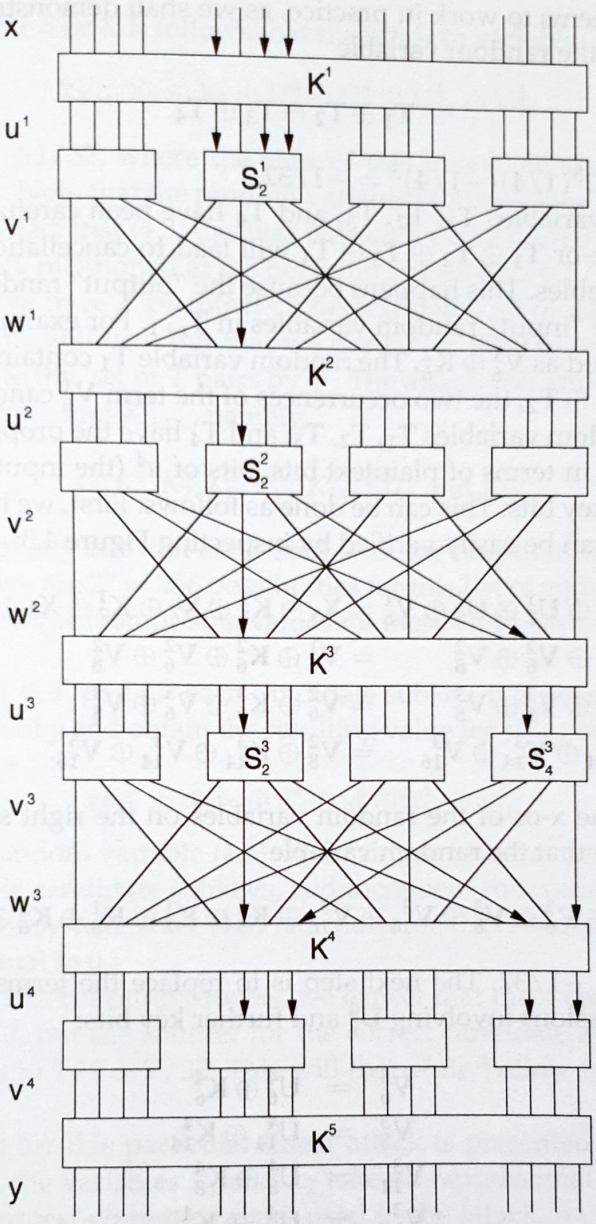

FIGURE 4.3: A linear approximation of a substitution-permutation network

If we make the assumption that these four random variables are independent, then we can compute the bias of their x-or using the piling-up lemma (Lemma 4.1). (The random variables are in fact not independent, which means that we cannot provide a mathematical justification of this approximation. Nevertheless, the approximation seems to work in practice, as we shall demonstrate.) We therefore hypothesize that the random variable

Pilling-up-lemma

$$\mathbf{T}_1 \oplus \mathbf{T}_2 \oplus \mathbf{T}_3 \oplus \mathbf{T}_4$$

has bias equal to $2^3(1/4)(-1/4)^3 = -1/32$.

The random variables $\mathbf{T}_1$, $\mathbf{T}_2$, $\mathbf{T}_3$, and $\mathbf{T}_4$ have been carefully constructed so that the exclusive-or $\mathbf{T}_1 \oplus \mathbf{T}_2 \oplus \mathbf{T}_3 \oplus \mathbf{T}_4$ will lead to cancellations of "intermediate" random variables. This happens because the "output" random variables in $\mathbf{T}_r$ correspond to the "input" random variables in $\mathbf{T}_{r+1}$. For example, the term $\mathbf{U}_6^2$ in $\mathbf{T}_2$ can be expressed as $\mathbf{V}_6^1 \oplus \mathbf{K}_6^2$. The random variable $\mathbf{T}_1$ contains a term $\mathbf{V}_6^1$. Thus, if we compute $\mathbf{T}_1 \oplus \mathbf{T}_2$, the two occurrences of the term $\mathbf{V}_6^1$ cancel each other out.

Thus, the random variables $\mathbf{T}_1$, $\mathbf{T}_2$, $\mathbf{T}_3$, and $\mathbf{T}_4$ have the property that their x-or can be expressed in terms of plaintext bits, bits of u^4 (the input to the last round of S-boxes), and key bits. This can be done as follows: First, we have the following relations, which can be easily verified by inspecting Figure 4.3:

$$\begin{aligned}
\mathbf{T}_1 &= \mathbf{U}_5^1 \oplus \mathbf{U}_7^1 \oplus \mathbf{U}_8^1 \oplus \mathbf{V}_6^1 = \mathbf{X}_5 \oplus \mathbf{K}_5^1 \oplus \mathbf{X}_7 \oplus \mathbf{K}_7^1 \oplus \mathbf{X}_8 \oplus \mathbf{K}_8^1 \oplus \mathbf{V}_6^1 \\
\mathbf{T}_2 &= \mathbf{U}_6^2 \oplus \mathbf{V}_6^2 \oplus \mathbf{V}_8^2 \qquad = \mathbf{V}_6^1 \oplus \mathbf{K}_6^2 \oplus \mathbf{V}_6^2 \oplus \mathbf{V}_8^2 \\
\mathbf{T}_3 &= \mathbf{U}_6^3 \oplus \mathbf{V}_6^3 \oplus \mathbf{V}_8^3 \qquad = \mathbf{V}_6^2 \oplus \mathbf{K}_6^3 \oplus \mathbf{V}_6^3 \oplus \mathbf{V}_8^3 \\
\mathbf{T}_4 &= \mathbf{U}_{14}^3 \oplus \mathbf{V}_{14}^3 \oplus \mathbf{V}_{16}^3 \qquad = \mathbf{V}_8^2 \oplus \mathbf{K}_{14}^3 \oplus \mathbf{V}_{14}^3 \oplus \mathbf{V}_{16}^3.
\end{aligned}$$

If we compute the x-or of the random variables on the right sides of the above equations, we see that the random variable

$$\mathbf{X}_5 \oplus \mathbf{X}_7 \oplus \mathbf{X}_8 \oplus \mathbf{V}_6^3 \oplus \mathbf{V}_8^3 \oplus \mathbf{V}_{14}^3 \oplus \mathbf{V}_{16}^3 \oplus \mathbf{K}_5^1 \oplus \mathbf{K}_7^1 \oplus \mathbf{K}_8^1 \oplus \mathbf{K}_6^2 \oplus \mathbf{K}_6^3 \oplus \mathbf{K}_{14}^3 \tag{4.1}$$

has bias equal to $-1/32$. The next step is to replace the terms $\mathbf{V}_i^3$ in the above formula by expressions involving $\mathbf{U}_i^4$ and further key bits:

$$\begin{aligned}
\mathbf{V}_6^3 &= \mathbf{U}_6^4 \oplus \mathbf{K}_6^4 \\
\mathbf{V}_8^3 &= \mathbf{U}_{14}^4 \oplus \mathbf{K}_{14}^4 \\
\mathbf{V}_{14}^3 &= \mathbf{U}_8^4 \oplus \mathbf{K}_8^4 \\
\mathbf{V}_{16}^3 &= \mathbf{U}_{16}^4 \oplus \mathbf{K}_{16}^4
\end{aligned}$$

Now we substitute these four expressions into (4.1), to get the following:

$$\begin{aligned}
&\mathbf{X}_5 \oplus \mathbf{X}_7 \oplus \mathbf{X}_8 \oplus \mathbf{U}_6^4 \oplus \mathbf{U}_8^4 \oplus \mathbf{U}_{14}^4 \oplus \mathbf{U}_{16}^4 \\
&\qquad \oplus \mathbf{K}_5^1 \oplus \mathbf{K}_7^1 \oplus \mathbf{K}_8^1 \oplus \mathbf{K}_6^2 \oplus \mathbf{K}_6^3 \oplus \mathbf{K}_{14}^3 \oplus \mathbf{K}_6^4 \oplus \mathbf{K}_8^4 \oplus \mathbf{K}_{14}^4 \oplus \mathbf{K}_{16}^4
\end{aligned} \tag{4.2}$$

This expression only involves plaintext bits, bits of u^4, and key bits. Suppose that the key bits in (4.2) are fixed. Then the random variable

$$\mathbf{K}_5^1 \oplus \mathbf{K}_7^1 \oplus \mathbf{K}_8^1 \oplus \mathbf{K}_6^2 \oplus \mathbf{K}_6^3 \oplus \mathbf{K}_{14}^3 \oplus \mathbf{K}_6^4 \oplus \mathbf{K}_8^4 \oplus \mathbf{K}_{14}^4 \oplus \mathbf{K}_{16}^4$$

has the (fixed) value 0 or 1. It follows that the random variable

$$\mathbf{X}_5 \oplus \mathbf{X}_7 \oplus \mathbf{X}_8 \oplus \mathbf{U}_6^4 \oplus \mathbf{U}_8^4 \oplus \mathbf{U}_{14}^4 \oplus \mathbf{U}_{16}^4 \tag{4.3}$$

has bias equal to $\pm 1/32$, where the sign of this bias depends on the values of unknown key bits. Note that the random variable (4.3) involves only plaintext bits and bits of u^4. The fact that (4.3) has bias bounded away from 0 allows us to carry out the linear attack mentioned at the beginning of Section 4.3.

Suppose that we have T plaintext-ciphertext pairs, all of which use the same unknown key, K. (It will turn out that we need $T \approx 8000$ in order for the attack to succeed.) Denote this set of T pairs by $\mathcal{T}$. The attack will allow us to obtain the eight key bits in $K_{<2>}^5$ and $K_{<4>}^5$, namely,

$$K_5^5, K_6^5, K_7^5, K_8^5, K_{13}^5, K_{14}^5, K_{15}^5, \text{ and } K_{16}^5.$$

These are the eight key bits that are x-ored with the output of the S-boxes S_2^4 and S_4^4. Notice that there are $2^8 = 256$ possibilities for this list of eight key bits. We will refer to a binary 8-tuple (comprising values for these eight key bits) as a ***candidate subkey***.

For each $(x, y) \in \mathcal{T}$ and for each candidate subkey, it is possible to compute a partial decryption of y and obtain the resulting value for $u_{<2>}^4$ and $u_{<4>}^4$. Then we compute the value

$$x_5 \oplus x_7 \oplus x_8 \oplus u_6^4 \oplus u_8^4 \oplus u_{14}^4 \oplus u_{16}^4 \tag{4.4}$$

taken on by the random variable (4.3). We maintain an array of counters indexed by the 256 possible candidate subkeys, and increment the counter corresponding to a particular subkey whenever (4.4) has the value 0. (This array is initialized to have all values equal to 0.)

At the end of this counting process, we expect that most counters will have a value close to $T/2$, but the counter for the correct candidate subkey will have a value that is close to $T/2 \pm T/32$. This will (hopefully) allow us to identify eight subkey bits.

The algorithm for this particular linear attack is presented as Algorithm 4.2. In this algorithm, the variables L_1 and L_2 take on hexadecimal values. The set $\mathcal{T}$ is the set of T plaintext-ciphertext pairs used in the attack. ${\pi_S}^{-1}$ is the permutation corresponding to the inverse of the S-box; this is used to partially decrypt the ciphertexts. The output, *maxkey*, contains the "most likely" eight subkey bits identified in the attack.

Algorithm 4.2 is not very complicated. As mentioned previously, we are just computing (4.4) for every plaintext-ciphertext pair $(x, y) \in \mathcal{T}$ and for every possible candidate subkey (L_1, L_2). In order to do this, we refer to Figure 4.3. First, we compute the exclusive-ors $L_1 \oplus y_{<2>}$ and $L_2 \oplus y_{<4>}$. These yield $v_{<2>}^4$ and

Algorithm 4.2: LINEARATTACK($\mathcal{T}, T, {\pi_S}^{-1}$)

for $(L_1, L_2) \leftarrow (0,0)$ **to** (F,F)
  do $Count[L_1, L_2] \leftarrow 0$
for each $(x, y) \in \mathcal{T}$
do **for** $(L_1, L_2) \leftarrow (0,0)$ **to** (F,F)
do $v^4_{<2>} \leftarrow L_1 \oplus y_{<2>}$
$v^4_{<4>} \leftarrow L_2 \oplus y_{<4>}$
$u^4_{<2>} \leftarrow {\pi_S}^{-1}(v^4_{<2>})$
$u^4_{<4>} \leftarrow {\pi_S}^{-1}(v^4_{<4>})$
$z \leftarrow x_5 \oplus x_7 \oplus x_8 \oplus u^4_6 \oplus u^4_8 \oplus u^4_{14} \oplus u^4_{16}$
if $z = 0$
then $Count[L_1, L_2] \leftarrow Count[L_1, L_2] + 1$
$max \leftarrow -1$
for $(L_1, L_2) \leftarrow (0,0)$ **to** (F,F)
do $Count[L_1, L_2] \leftarrow |Count[L_1, L_2] - T/2|$
if $Count[L_1, L_2] > max$
then $max \leftarrow Count[L_1, L_2]$
$maxkey \leftarrow (L_1, L_2)$
output $(maxkey)$

$v^4_{<4>}$, respectively, when (L_1, L_2) is the correct subkey. $u^4_{<2>}$ and $u^4_{<4>}$ can then be computed from $v^4_{<2>}$ and $v^4_{<4>}$ by using the inverse S-box ${\pi_S}^{-1}$; again, the values obtained are correct if (L_1, L_2) is the correct subkey. Then we compute (4.4) and we increment the counter for the pair (L_1, L_2) if (4.4) has the value 0. After having computed all the relevant counters, we just find the pair (L_1, L_2) corresponding to the maximum counter; this is the output of Algorithm 4.2.

In general, it is suggested that a linear attack based on a linear approximation having bias equal to ϵ will be successful if the number of plaintext-ciphertext pairs, which we denote by T, is approximately $c\,\epsilon^{-2}$, for some "small" constant c. We implemented the attack described in Algorithm 4.2, and found that the attack was usually successful if we took $T = 8000$. Note that $T = 8000$ corresponds to $c \approx 8$, because $\epsilon^{-2} = 1024$.

4.4 Differential Cryptanalysis

Differential cryptanalysis is similar to linear cryptanalysis in many respects. The main difference from linear cryptanalysis is that differential cryptanalysis involves comparing the x-or of two inputs to the x-or of the corresponding two out-

puts. In general, we will be looking at inputs x and x^* (which are assumed to be binary strings) having a specified (fixed) x-or value denoted by $x' = x \oplus x^*$. Throughout this section, we will use prime markings ($'$) to indicate the x-or of two bitstrings.

Differential cryptanalysis is a chosen-plaintext attack. We assume that an attacker has a large number of tuples (x, x^*, y, y^*), where the x-or value $x' = x \oplus x^*$ is fixed. The plaintext elements (i.e., x and x^*) are encrypted using the same unknown key, K, yielding the ciphertexts y and y^*, respectively. For each of these tuples, we will begin to decrypt the ciphertexts y and y^*, using all possible candidate keys for the last round of the cipher. For each candidate key, we compute the values of certain state bits, and determine if their x-or has a certain value (namely, the most likely value for the given input x-or). Whenever it does, we increment a counter corresponding to the particular candidate key. At the end of this process, we hope that the candidate key that has the highest frequency count contains the correct values for these key bits. (As we did with linear cryptanalysis, we will illustrate the attack with a particular example.)

Definition 4.2: Let $\pi_S : \{0,1\}^m \to \{0,1\}^n$ be an S-box. Consider an (ordered) pair of bitstrings of length m, say (x, x^*). We say that the ***input x-or*** of the S-box is $x \oplus x^*$ and the ***output x-or*** is $\pi_S(x) \oplus \pi_S(x^*)$. Note that the output x-or is a bitstring of length n.

For any $x' \in \{0,1\}^m$, define the set $\Delta(x')$ to consist of all the ordered pairs (x, x^*) having input x-or equal to x'.

It is easy to see that any set $\Delta(x')$ contains 2^m pairs, and that

$$\Delta(x') = \{(x, x \oplus x') : x \in \{0,1\}^m\}.$$

For each pair in $\Delta(x')$, we can compute the output x-or of the S-box. Then we can tabulate the resulting distribution of output x-ors. There are 2^m output x-ors, which are distributed among 2^n possible values. A non-uniform output distribution will be the basis for a successful differential attack.

Example 4.3 We again use the S-box from Example 4.1. Suppose we consider input x-or $x' = 1011$. Then

$$\Delta(1011) = \{(0000, 1011), (0001, 1010), \ldots, (1111, 0100)\}.$$

For each ordered pair in the set $\Delta(1011)$, we compute output x-or of π_S in Table 4.3. In each row of this table, we have $x \oplus x^* = 1011$, $y = \pi_S(x)$, $y^* = \pi_S(x^*)$, and $y' = y \oplus y^*$.

Looking at the last column of Table 4.3, we obtain the following distribution of output x-ors:

0000	0001	0010	0011	0100	0101	0110	0111
0	0	8	0	0	2	0	2

TABLE 4.3: Input and output x-ors

x	x^*	y	y^*	y'
0000	1011	1110	1100	0010
0001	1010	0100	0110	0010
0010	1001	1101	1010	0111
0011	1000	0001	0011	0010
0100	1111	0010	0111	0101
0101	1110	1111	0000	1111
0110	1101	1011	1001	0010
0111	1100	1000	0101	1101
1000	0011	0011	0001	0010
1001	0010	1010	1101	0111
1010	0001	0110	0100	0010
1011	0000	1100	1110	0010
1100	0111	0101	1000	1101
1101	0110	1001	1011	0010
1110	0101	0000	1111	1111
1111	0100	0111	0010	0101

1000	1001	1010	1011	1100	1101	1110	1111
0	0	0	0	0	2	0	2

□

In Example 4.3, only five of the 16 possible output x-ors actually occur. This particular example has a very non-uniform distribution.

We can carry out computations, as was done in Example 4.3, for any possible input x-or. It will be convenient to have some notation to describe the distributions of the output x-ors, so we state the following definition.

Definition 4.3: For a bitstring x' of length m and a bitstring y' of length n, define

$$N_D(x', y') = |\{(x, x^*) \in \Delta(x') : \pi_S(x) \oplus \pi_S(x^*) = y'\}|.$$

In other words, $N_D(x', y')$ counts the number of pairs with input x-or equal to x' that also have output x-or equal to y' (for a given S-box). All the values $N_D(a', b')$ for the S-box from Example 4.1 are tabulated in Table 4.4 (a' and b' are the hexadecimal representations of the input and output x-ors, respectively). Observe that the distribution computed in Example 4.3 corresponds to row "B" in the table in Table 4.4.

Recall that the input to the ith S-box in round r of the SPN from Example 4.1 is denoted $u^r_{<i>}$, and

$$u^r_{<i>} = w^{r-1}_{<i>} \oplus K^r_{<i>}.$$

TABLE 4.4: Difference distribution table: values of $N_D(a', b')$

	b'															
a'	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	16	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	0	2	0	0	0	2	0	2	4	0	4	2	0	0
2	0	0	0	2	0	6	2	2	0	2	0	0	0	0	2	0
3	0	0	2	0	2	0	0	0	0	4	2	0	2	0	0	4
4	0	0	0	2	0	0	6	0	0	2	0	4	2	0	0	0
5	0	4	0	0	0	2	2	0	0	0	4	0	2	0	0	2
6	0	0	0	4	0	4	0	0	0	0	0	0	2	2	2	2
7	0	0	2	2	2	0	2	0	0	2	2	0	0	0	0	4
8	0	0	0	0	0	0	2	2	0	0	0	4	0	4	2	2
9	0	2	0	0	2	0	0	4	2	0	2	2	2	0	0	0
A	0	2	2	0	0	0	0	0	6	0	0	2	0	0	4	0
B	0	0	8	0	0	2	0	2	0	0	0	0	0	2	0	2
C	0	2	0	0	2	2	2	0	0	0	0	2	0	6	0	0
D	0	4	0	0	0	0	0	4	2	0	2	0	2	0	2	0
E	0	0	2	4	2	0	0	0	6	0	0	0	0	0	2	0
F	0	2	0	0	6	0	0	0	0	4	0	2	0	0	2	0

An input x-or is computed as

$$\begin{aligned} u^r_{<i>} \oplus (u^r_{<i>})^* &= (w^{r-1}_{<i>} \oplus K^r_{<i>}) \oplus ((w^{r-1}_{<i>})^* \oplus K^r_{<i>}) \\ &= w^{r-1}_{<i>} \oplus (w^{r-1}_{<i>})^* \end{aligned}$$

Therefore, this input x-or does not depend on the subkey bits used in round r; it is equal to the (permuted) output x-or of round $r-1$. (However, the output x-or of round r certainly does depend on the subkey bits in round r.)

Let a' denote an input x-or and let b' denote an output x-or. The pair (a', b') is called a ***differential***. Each entry in the difference distribution table gives rise to an ***x-or propagation ratio*** (or more simply, a ***propagation ratio***) for the corresponding differential.

Definition 4.4: The propagation ratio $R_p(a', b')$ for the differential (a', b') is defined as follows:

$$R_p(a', b') = \frac{N_D(a', b')}{2^m}.$$

$R_p(a', b')$ can be interpreted as a conditional probability:

$$\mathbf{Pr}[\text{output x-or } = b' | \text{ input x-or } = a'] = R_p(a', b').$$

Suppose we find propagation ratios for differentials in consecutive rounds of

the SPN, such that the input x-or of a differential in any round is the same as the (permuted) output x-ors of the differentials in the previous round. Then these differentials can be combined to form a ***differential trail***. We make the assumption that the various propagation ratios in a differential trail are independent (an assumption that may not be mathematically valid, in fact). This assumption allows us to multiply the propagation ratios of the differentials in order to obtain the propagation ratio of the differential trail.

We illustrate this process by returning to the SPN from Example 4.1. A particular differential trail is shown in Figure 4.4. Arrows are used to highlight the "1" bits in the input and output x-ors of the differentials that are used in the differential trail.

The differential attack arising from Figure 4.4 uses the following propagation ratios of differentials, all of which can be verified from Figure 4.4:

- In S_2^1, $R_p(1011, 0010) = 1/2$
- In S_3^2, $R_p(0100, 0110) = 3/8$
- In S_2^3, $R_p(0010, 0101) = 3/8$
- In S_3^3, $R_p(0010, 0101) = 3/8$

These differentials can be combined to form a differential trail. We therefore obtain a propagation ratio for a differential trail of the first three rounds of the SPN:

$$R_p(0000\ 1011\ 0000\ 0000, 0000\ 0101\ 0101\ 0000) = \frac{1}{2} \times \left(\frac{3}{8}\right)^3 = \frac{27}{1024}.$$

In other words,

$$x' = 0000\ 1011\ 0000\ 0000 \Rightarrow (v^3)' = 0000\ 0101\ 0101\ 0000$$

with probability 27/1024. However,

$$(v^3)' = 0000\ 0101\ 0101\ 0000 \Leftrightarrow (u^4)' = 0000\ 0110\ 0000\ 0110.$$

Hence, it follows that

$$x' = 0000\ 1011\ 0000\ 0000 \Rightarrow (u^4)' = 0000\ 0110\ 0000\ 0110$$

with probability 27/1024. Note that $(u^4)'$ is the x-or of two inputs to the last round of S-boxes.

Now we can present an algorithm, for this particular example, based on the informal description at the beginning of this section; see Algorithm 4.3. The input and output of this algorithm are similar to linear attack; the main difference is that $\mathcal{T}$ is a set of tuples of the form (x, x^*, y, y^*), where x' is fixed, in the differential attack.

Algorithm 4.3 makes use of a certain ***filtering operation***. Tuples (x, x^*, y, y^*)

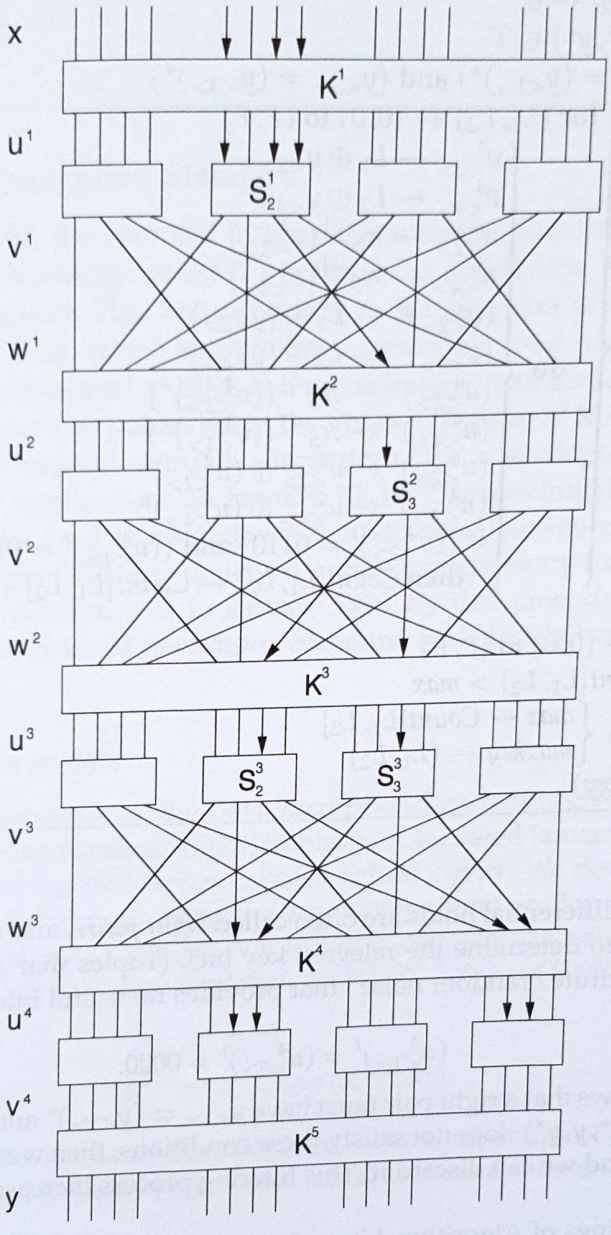

FIGURE 4.4: A differential trail for a substitution-permutation network

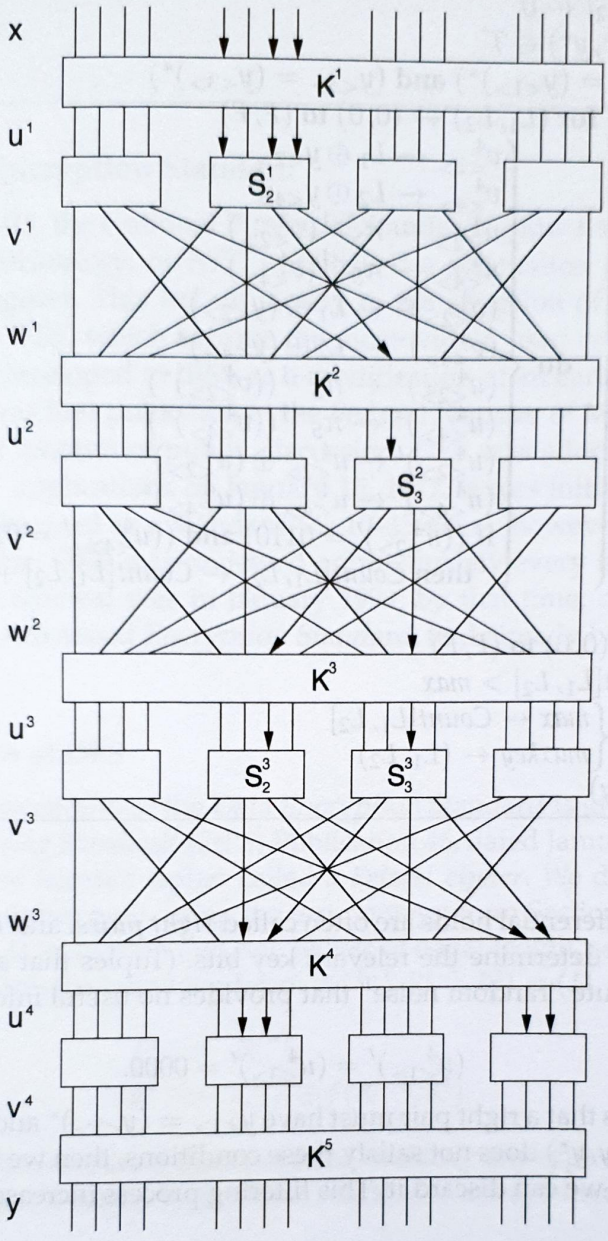

FIGURE 4.4: A differential trail for a substitution-permutation network

Algorithm 4.3: DIFFERENTIALATTACK($\mathcal{T}, T, {\pi_S}^{-1}$)

for $(L_1, L_2) \leftarrow (0,0)$ **to** (F,F)
  do $Count[L_1, L_2] \leftarrow 0$
for each $(x, y, x^*, y^*) \in \mathcal{T}$
do **if** $(y_{<1>} = (y_{<1>})^*)$ **and** $(y_{<3>} = (y_{<3>})^*)$
then **for** $(L_1, L_2) \leftarrow (0,0)$ **to** (F,F)
do $v^4_{<2>} \leftarrow L_1 \oplus y_{<2>}$
$v^4_{<4>} \leftarrow L_2 \oplus y_{<4>}$
$u^4_{<2>} \leftarrow {\pi_S}^{-1}(v^4_{<2>})$
$u^4_{<4>} \leftarrow {\pi_S}^{-1}(v^4_{<4>})$
$(v^4_{<2>})^* \leftarrow L_1 \oplus (y_{<2>})^*$
$(v^4_{<4>})^* \leftarrow L_2 \oplus (y_{<4>})^*$
$(u^4_{<2>})^* \leftarrow {\pi_S}^{-1}((v^4_{<2>})^*)$
$(u^4_{<4>})^* \leftarrow {\pi_S}^{-1}((v^4_{<4>})^*)$
$(u^4_{<2>})' \leftarrow u^4_{<2>} \oplus (u^4_{<2>})^*$
$(u^4_{<4>})' \leftarrow u^4_{<4>} \oplus (u^4_{<4>})^*$
if $((u^4_{<2>})' = 0110)$ **and** $((u^4_{<4>})' = 0110)$
then $Count[L_1, L_2] \leftarrow Count[L_1, L_2] + 1$
$max \leftarrow -1$
for $(L_1, L_2) \leftarrow (0,0)$ **to** (F,F)
do **if** $Count[L_1, L_2] > max$
then $max \leftarrow Count[L_1, L_2]$
$maxkey \leftarrow (L_1, L_2)$
output $(maxkey)$

for which the differential holds are often called ***right pairs***, and it is the right pairs that allow us to determine the relevant key bits. (Tuples that are not right pairs basically constitute "random noise" that provides no useful information.) A right pair has

$$(u^4_{<1>})' = (u^4_{<3>})' = 0000.$$

Hence, it follows that a right pair must have $y_{<1>} = (y_{<1>})^*$ and $y_{<3>} = (y_{<3>})^*$. If a tuple (x, x^*, y, y^*) does not satisfy these conditions, then we know that it is not a right pair, and we can discard it. This filtering process increases the efficiency of the attack.

The workings of Algorithm 4.3 can be summarized as follows. For each tuple $(x, x^*, y, y^*) \in \mathcal{T}$, we first perform the filtering operation. If (x, x^*, y, y^*) is a right pair, then we test each possible candidate subkey (L_1, L_2) and increment an appropriate counter if a certain x-or is observed. The steps include computing an exclusive-or with candidate subkeys and applying the inverse S-box (as was done in Algorithm 4.2), followed by computation of the relevant x-or value.

A differential attack based on a differential trail having propagation ratio equal

to ϵ will often be successful if the number of tuples (x, x^*, y, y^*), which we denote by T, is approximately $c\,\epsilon^{-1}$, for a "small" constant c. We implemented the attack described in Algorithm 4.3, and found that the attack was often successful if we took T between 50 and 100. In this example, $\epsilon^{-1} \approx 38$.

4.5 The Data Encryption Standard

On May 15, 1973, the National Bureau of Standards (now the *National Institute of Standards and Technology*, or *NIST*) published a solicitation for cryptosystems in the Federal Register. This led ultimately to the adoption of the *Data Encryption Standard*, or *DES*, which became the most widely used cryptosystem in the world. *DES* was developed at IBM, as a modification of an earlier system known as *Lucifer*. *DES* was first published in the Federal Register of March 17, 1975. After a considerable amount of public discussion, *DES* was adopted as a standard for "unclassified" applications on January 15, 1977. It was initially expected that DES would only be used as a standard for 10–15 years; however, it proved to be much more durable. *DES* was reviewed approximately every five years after its adoption. Its last renewal was in January 1999; by that time, development of a replacement, the *Advanced Encryption Standard*, had already begun (see Section 4.6).

4.5.1 Description of DES

A complete description of the *Data Encryption Standard* is given in the *Federal Information Processing Standards* (*FIPS*) Publication 46, dated January 15, 1977. *DES* is a special type of iterated cipher called a ***Feistel cipher***. We describe the basic form of a Feistel cipher now, using the terminology from Section 4.1. In a Feistel cipher, each state u^i is divided into two halves of equal length, say L^i and R^i. The round function g has the following form: $g(L^{i-1}, R^{i-1}, K^i) = (L^i, R^i)$, where

$$\begin{aligned} L^i &= R^{i-1} \\ R^i &= L^{i-1} \oplus f(R^{i-1}, K^i). \end{aligned}$$

We observe that the function f does not need to satisfy any type of injective property. This is because a Feistel-type round function is always invertible, given the round key:

$$\begin{aligned} L^{i-1} &= R^i \oplus f(L^i, K^i) \\ R^{i-1} &= L^i. \end{aligned}$$

DES is a 16-round Feistel cipher having block length 64: it encrypts a plaintext bitstring x (of length 64) using a 56-bit key, K, obtaining a ciphertext bitstring (of length 64). Prior to the 16 rounds of encryption, there is a fixed ***initial permutation***