Lecture Notes for CS-578 Cryptography (SS2007) Prof. Michael Backes

3. Block Ciphers

Lectures 3, 4 Saarland University

Block ciphers are symmetric ciphers operating block-wise, i.e., on bitstrings of a fixed length. In the following we will see two examples of widely used block ciphers, namely the somehow outdated DES and its successor AES. In the sequel we will learn how such ciphers can be used to securely encrypt a larger amount of data, and we will briefly discuss some attacks on block ciphers. First, however, we will introduce the concept of Feistel networks, which constitutes an important design principle underlying many block ciphers.

3.1 Feistel Networks

Feistel networks are a specific construction for designing symmetric encryption schemes. They were described first by Horst Feistel during his work on the cipher *Lucifer* at IBM. Lucifer was the predecessor of the Data Encryption Standard (DES), and both are built upon the same design. Other ciphers using Feistel networks include IDEA, RC5, and Skipjack.

A Feistel network defines a function $F\colon \{0,1\}^{2n} \to \{0,1\}^{2n}$. It is parameterized by the *number of rounds* $d \in \mathbb{N}$ and the *round functions* $f_1, \ldots, f_d\colon \{0,1\}^n \to \{0,1\}^n$. The function F is defined as follows. It operates in d rounds, typically between 12 and 16. In the i-th round the following operations are executed:

1. The round input is split into two halves $L_{i-1} \parallel R_{i-1}$,
2. the round function f_i is applied to the right half yielding $f_i(R_{i-1})$,
3. the Exclusive OR of this value and the left half is computed yielding $L_{i-1} \oplus f_i(R_{i-1})$, and
4. the left and right side are swapped, thus yielding the round output $L_i \parallel R_i := R_{i-1} \parallel L_{i-1} \oplus f_i(R_{i-1})$.

This construction is depicted in Figure 3.1. The main innovation is that the function F defined by a Feistel network is a permutation for arbitrary functions f_i. In particular, these do not even need to be invertible.

Proposition 3.1 (Feistel Networks are Permutations) *Every Feistel network $F\colon \{0,1\}^{2n} \to \{0,1\}^{2n}$ constitutes a permutation.*

Proof. First, we show how to invert a single round i. Let F_i denote the function computed in the i-th round. We define the candidate inverse function G_i for input $L_i \parallel R_i$ as $R_i \oplus f_i(L_i) \parallel L_i$. The following simple calculation shows that this is indeed the inverse function:

$$\begin{aligned}
G_i(F_i(L_{i-1} \parallel R_{i-1})) &= G_i(R_{i-1} \parallel L_{i-1} \oplus f_i(R_{i-1})) \\
&= (L_{i-1} \oplus f_i(R_{i-1})) \oplus f_i(R_{i-1}) \parallel R_{i-1} \\
&= L_{i-1} \parallel R_{i-1}.
\end{aligned}$$

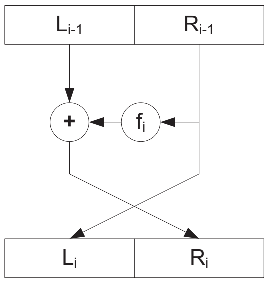

Figure 3.1: One Round in a Feistel Network

The candidate inverse of the function F having d rounds is now defined as $G_1 \circ \cdots \circ G_d$; easy calculation show that this is indeed the inverse:

$$\begin{aligned} & G_1(G_2(\ldots G_{d-1}(G_d(F_d(F_{d-1}(\ldots F_1(m)\ldots))))\ldots)) \\ = \; & G_1(G_2(\ldots G_{d-1}(F_{d-1}(\ldots F_1(m)\ldots))\ldots)) \\ = \; & \ldots \\ = \; & G_1(F_1(m)) \\ = \; & m. \end{aligned}$$

■

Feistel networks are appealing because of their simple design, and have the additional nice property that the same hardware circuit can be used for both encryption and decryption. This was particularly important in the 60's and 70's when the first block ciphers were designed, and when hardware was still much more expensive. There are two minor modification when a ciphertext should be decrypted in a Feistel network, but these do not affect the network itself but the handling of inputs of the circuit, i.e., messages and inputs of round functions. Namely (*i*) the round functions need to be applied in different order, and (*ii*) writing $c = L_d \parallel R_d$ as the ciphertext, one inputs $R_d \parallel L_d$ to the network for decryption, i.e., one exchanges the left and the right half of the ciphertext, and additionally one exchanges the left and the right half of the output of the Feistel network. One can easily verify that the Feistel network (with these modifications) computes the decryption operation as defined in the above proof.

3.2 The Data Encryption Standard (DES)

For a long time DES was one of the most widely applied block ciphers. It was designed by IBM in collaboration with the NSA and published as a standard in FIPS PUB 46-3. There were rumors about weaknesses the NSA had built into it, but until now no evidence was found for this. The main point of criticism against DES was its limited key length, which is nowadays the reason why pure DES is no longer used. In fact, the first DES break, in the sense of a total break given a certain number of plaintext/ciphertext pairs, was reported in 1997 by the DESCHALL project and took about three months. In 1998 the Electronic Frontier Foundation (EFF) built a specialized hardware device resulting in a break in roughly three days. Later they where cooperating with *distributed.net* breaking a DES challenge in 22 hours. The world record for breaking a DES encryption is currently

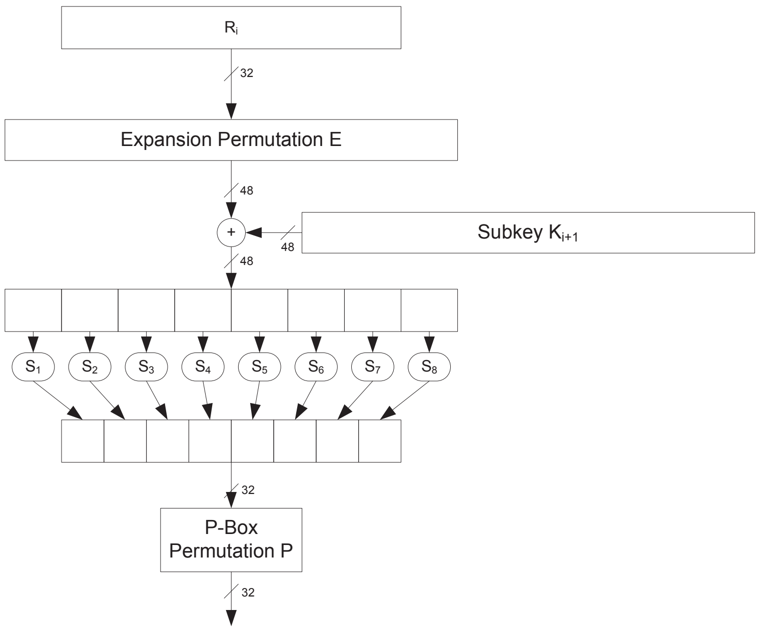

Figure 3.2: The DES Round Function f_i.

10 hours, and this is likely to decrease in the future. A variant called *Triple-DES* (3DES), which we will discuss later in this chapter, is still deployed and seems to provide good security.

3.2.1 Overall Construction of DES

DES is essentially a Feistel network of 16 rounds operating on blocks of 64 bits and using a key of 56 bits, which is randomly chosen from $\{0,1\}^{56}$. What makes DES different from a pure Feistel design is the initial permutation IP which is applied before the first round starts, and whose inverse IP^{-1} is applied after the last round. The DES round function f_i is depicted in Figure 3.2. It executes the following steps:

1. The message block of 32 bit length is expanded to a block of 48 bit by doubling 16 bits and permuting the block as defined by the function E shown in Figure 3.4.
2. The Exclusive OR of that value with the round key K_i is computed. The derivation of the round keys, i.e., the *key schedule*, is defined below.
3. This block of 48 bit is split into eight blocks with 6 bit each. Each of them is the input to one of the eight *S-boxes* $S_1, \ldots, S_8$ as shown in Figure 3.5. Each S-box yields a 4 bit output. Concatenating these outputs yields a block of 32 bit.
4. The permutation P defined in Figure 3.6 is applied to the 32-bit block, yielding the output of f_i.

3.2.2 Key Schedule

The choice of the key used in each round is called *key schedule*. Even if a DES key is composed of 64 bits, 8 bits are used for error detection, thus resulting in an actual key length of 56 bits. In the first step, the error correction bits are stripped off a 64 bit key K and the bits are permuted by a fixed permutation. Both is performed by the function PC-1 shown in Figure 3.7. The output is

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

Figure 3.3: Initial permutation (IP) in DES: The first bit of the output is the 58-th bit of the input, the second bit of the output is the 50-th bits of the input, and so on.

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

Figure 3.4: Bit-selection function E in DES: The first bit of output is the 32-nd bit of the input, the second bit of output is the first of the input and so on.

divided into a 28-bit left half C_0 and a 28-bit right half D_0. Now for each round we compute

$$\begin{aligned} C_i &= C_{i-1} \ll p_i \\ D_i &= D_{i-1} \ll p_i \end{aligned}$$

where $x \ll p_i$ means a cyclic shift on x to the left by p_i positions where $p_i = 1$ if $i \in \{1, 2, 9, 16\}$, and $p_i = 2$ otherwise. Finally, C_i and D_i are joined together and permuted according to PC-2, obtaining the 48-bit round key. This permutation is given in Figure 3.7.

3.2.3 The Security of DES

DES withstood attacks quite successfully apart from some attacks based on linear and differential cryptanalysis which we shall discuss later. However, the major weakness of DES is its limited key length of 56 bits which is nowadays not enough to provide a reasonable level of security. For this reason, several improvements on the plain DES have been proposed. We will discuss some of them in the following.

3.3 The Advanced Encryption Standard (AES)

The *advanced encryption standard* (AES) is the successor of the outdated DES standard. It was developed by two Belgian cryptographers, Joan Daemen and Vincent Rijmen, and standardized as US FIPS PUB 197 in November 2001. It works on blocks of 128 bits, supports key lengths of 128, 192, and 256 bits and operates in 10, 12 and 14 rounds, respectively. AES is motivated by algebraic operations, and its implementation in hardware and software is compact and fast.

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

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

S_3:															
10	0	9	14	6	3	15	5	1	13	12	7	11	4	2	8
13	7	0	9	3	4	6	10	2	8	5	14	12	11	15	1
13	6	4	9	8	15	3	0	11	1	2	12	5	10	14	7
1	10	13	0	6	9	8	7	4	15	14	3	11	5	2	12

S_4:															
7	13	14	3	0	6	9	10	1	2	8	5	11	12	4	15
13	8	11	5	6	15	0	3	4	7	2	12	1	10	14	9
10	6	9	0	12	11	7	13	15	1	3	14	5	2	8	4
3	15	0	6	10	1	13	8	9	4	5	11	12	7	2	14

S_5:															
2	12	4	1	7	10	11	6	8	5	3	15	13	0	14	9
14	11	2	12	4	7	13	1	5	0	15	10	3	9	8	6
4	2	1	11	10	13	7	8	15	9	12	5	6	3	0	14
11	8	12	7	1	14	2	13	6	15	0	9	10	4	5	3

S_6:															
12	1	10	15	9	2	6	8	0	13	3	4	14	7	5	11
10	15	4	2	7	12	9	5	6	1	13	14	0	11	3	8
9	14	15	5	2	8	12	3	7	0	4	10	1	13	11	6
4	3	2	12	9	5	15	10	11	14	1	7	6	0	8	13

S_7:															
4	11	2	14	15	0	8	13	3	12	9	7	5	10	6	1
13	0	11	7	4	9	1	10	14	3	5	12	2	15	8	6
1	4	11	13	12	3	7	14	10	15	6	8	0	5	9	2
6	11	13	8	1	4	10	7	9	5	0	15	14	2	3	12

S_8:															
13	2	8	4	6	15	11	1	10	9	3	14	5	0	12	7
1	15	13	8	10	3	7	4	12	5	6	11	0	14	9	2
7	11	4	1	9	12	14	2	0	6	10	13	15	3	5	8
2	1	14	7	4	10	8	13	15	12	9	0	3	5	6	11

Figure 3.5: S-Boxes in DES: For a binary string $x_0x_1x_2x_3x_4x_5x_6x_7$, the output is computed by looking up the entry in row x_0x_7 and column $x_1x_2x_3x_4x_5x_6$. This representation has historic reasons.

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

Figure 3.6: DES permutation P: The first bit of the output is the 16-th bit of the input.

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

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

Figure 3.7: DES Key schedule permutations *PC*-1 and *PC*-2 (Note that the indices refer to the original key with error-correcting bits.)

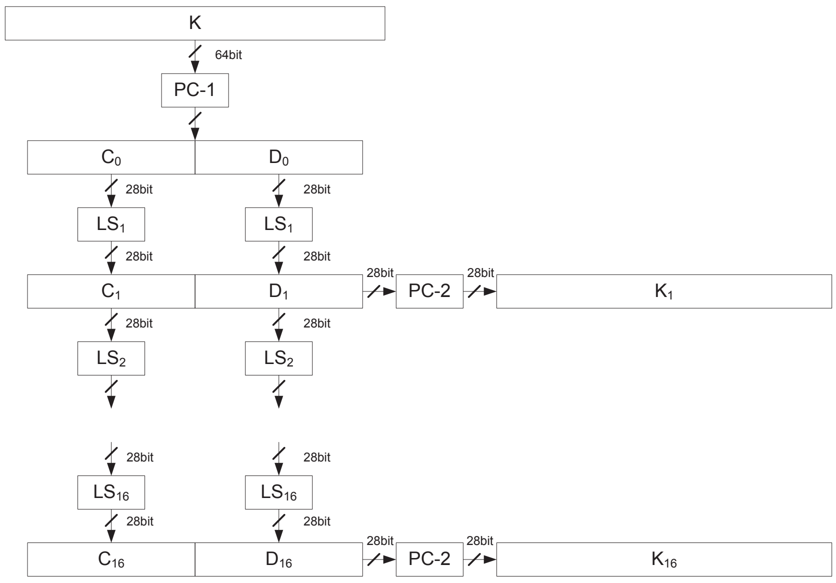

Figure 3.8: Key Scheduling of DES

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

Figure 3.9: S-box in AES: Substitution values for the byte xy (in hexadecimal format).

3.3.1 Construction

AES operates on an array of 4×4 bytes, which is initialized with a message $m = m_0 \parallel m_1 \parallel \ldots \parallel m_{15}$ as follows:

$$\begin{array}{|c|c|c|c|} \hline m_0 & m_4 & m_8 & m_{12} \\ \hline m_1 & m_5 & m_9 & m_{13} \\ \hline m_2 & m_6 & m_{10} & m_{14} \\ \hline m_3 & m_7 & m_{11} & m_{15} \\ \hline \end{array} \rightarrow \begin{array}{|c|c|c|c|} \hline a_{0,0} & a_{0,1} & a_{0,2} & a_{0,3} \\ \hline a_{1,0} & a_{1,1} & a_{1,2} & a_{1,3} \\ \hline a_{2,0} & a_{2,1} & a_{2,2} & a_{2,3} \\ \hline a_{3,0} & a_{3,1} & a_{3,2} & a_{3,3} \\ \hline \end{array}$$

Each round applies the following manipulations to the state (i.e., to the array):

1. A substitution operation on every single byte SubBytes(),
2. a byte permutation ShiftRows(),
3. a column manipulation MixColumns(),
4. and the Exclusive OR of the state with the round key AddRoundKey().

An exception is the last round, where the MixColumns() operation is skipped. Now we discuss these operations in detail.

SubBytes(): This operation is similar to the S-Box substitution of DES. Each byte $a_{i,j}$ of the state is substituted by the output of a single S-Box. This S-Box has also a nice algebraic representation over the finite field $\mathbb{F}_{2^8}$; However, we do not want to develop all the theory and simply give it in the form of a table, see Table 3.9. This table can also be used in implementations where spending space to a fully specified table is not a major concern since table look-up is significantly faster than on-the-fly calculations.

ShiftRow(): The lower three rows are shifted by one, two, and three positions, respectively.

$$\begin{array}{|c|c|c|c|} \hline a_{0,0} & a_{0,1} & a_{0,2} & a_{0,3} \\ \hline a_{1,0} & a_{1,1} & a_{1,2} & a_{1,3} \\ \hline a_{2,0} & a_{2,1} & a_{2,2} & a_{2,3} \\ \hline a_{3,0} & a_{3,1} & a_{3,2} & a_{3,3} \\ \hline \end{array} \rightarrow \begin{array}{|c|c|c|c|} \hline a_{0,0} & a_{0,1} & a_{0,2} & a_{0,3} \\ \hline a_{1,1} & a_{1,2} & a_{1,3} & a_{1,0} \\ \hline a_{2,2} & a_{2,3} & a_{2,0} & a_{2,1} \\ \hline a_{3,3} & a_{3,0} & a_{3,1} & a_{3,2} \\ \hline \end{array}$$

MixColumns(): The MixColumns operation replaces each column i as follows:

$$\begin{pmatrix} a'_{0,i} \\ a'_{1,i} \\ a'_{2,i} \\ a'_{3,i} \end{pmatrix} = \begin{pmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{pmatrix} \begin{pmatrix} a_{0,i} \\ a_{1,i} \\ a_{2,i} \\ a_{3,i} \end{pmatrix},$$

where the multiplication is calculated in $\mathbb{F}_{2^8}$.

AddRoundKey(): The round key is represented as a 4×4 matrix with bytes as elements. The new state is computed as the Exclusive OR of the key and the current state.

3.3.2 Key Schedule

The round keys are derived from the main key by successively using functions RotWord() rotating by a whole word and SubWord() applying the S-Box of Figure 3.9 to four bytes in parallel. It is described in detail in FIPS PUB 197.

3.3.3 Security of AES

AES can, to a certain extent, be attacked with linear and differential cryptanalysis. However, these attacks can only reduce the effective key length of AES with 128-bit key by a few bits, i.e., AES currently yields a very comfortable degree of security.

It is worth noting that AES with 256-bit keys is even used for protecting “top secret” documents of the NSA (this is the highest NSA secrecy level). It is the first time in history that a publicly known algorithm is used for these highly classified documents.

3.4 Some Attacks on Block Ciphers

In the sequel we sketch some possible attacks on block ciphers and briefly discuss their effects on DES and AES.

3.4.1 Exhaustive Key Search

The conceptually simplest attack that can be mounted against any block cipher is *exhaustive key search*, also called a *brute-force attack*. Knowing a few plaintext/ciphertext pairs $(m_1, c_1), (m_2, c_2), \ldots$ (i.e. $c_i = \mathsf{E}(K, m_i)$), one tries to find the key K by testing every possible key.

To estimate the effectiveness of this attack we need to know how many keys would encrypt a fixed plaintext to a specific ciphertext. We will give an estimation for this in the sequel, where we replace the block cipher with a random function for every key. While this technique is not accurate, it makes use of the intuition that the encryption function should essentially constitute a random function, and is usually considered to be a reasonable abstraction. One says that the block cipher is considered to be an *ideal cipher*. This heuristic is used quite often, because an accurate analysis of a real block cipher such as DES is extremely complex.

For the case of DES we prove the following statement: For a randomly chosen message block $m \in \{0,1\}^{64}$ and a randomly chosen key $K \in \{0,1\}^{56}$, the probability that there exists another key encrypting the message to the same ciphertext is very low, provided that DES behaves like an ideal cipher. One also says that the *unicity distance*, i.e., the probability that a key is already uniquely determined after seeing one plaintext/ciphertext pair, is very high.

Lemma 3.2 *Let* $(\mathsf{K}^{\mathsf{DES}}, \mathsf{E}^{\mathsf{DES}}, \mathsf{D}^{\mathsf{DES}})$ *denote the idealized DES cipher, i.e., for randomly chosen key* $K \in [\mathsf{K}^{\mathsf{DES}}]$*, the function* $\mathsf{E}^{\mathsf{DES}}(K,\cdot)$ *is "as good as" a randomly chosen function* $R\colon \{0,1\}^{64} \to \{0,1\}^{64}$*. Then*

$$\mathsf{Pr}\left[\exists K' \in [\mathsf{K}^{\mathsf{DES}}] : K' \neq K \wedge \mathsf{E}^{\mathsf{DES}}(K,m) = \mathsf{E}^{\mathsf{DES}}(K',m);\ m \leftarrow_{\mathcal{R}} \mathcal{M},\ K \leftarrow \mathsf{K}^{\mathsf{DES}}\right] \leq \frac{1}{256}$$

□

Proof. The calculation is rather straightforward.

$$\begin{aligned}
&\mathsf{Pr}\left[\exists K' \in [\mathsf{K}^{\mathsf{DES}}] : K' \neq K \wedge \mathsf{E}^{\mathsf{DES}}(K,m) = \mathsf{E}^{\mathsf{DES}}(K',m);\ m \leftarrow_{\mathcal{R}} \mathcal{M},\ K \leftarrow \mathsf{K}^{\mathsf{DES}}\right] \\
\overset{(1)}{\leq}\ & \sum_{K' \in \{0,1\}^{56}} \mathsf{Pr}\left[\mathsf{E}^{\mathsf{DES}}(K,m) = \mathsf{E}^{\mathsf{DES}}(K',m) \wedge K' \neq K;\ m \leftarrow_{\mathcal{R}} \mathcal{M},\ K \leftarrow \mathsf{K}^{\mathsf{DES}}\right] \\
=\ & \sum_{K,K' \in \{0,1\}^{56}} \frac{1}{2^{56}} \cdot \mathsf{Pr}\left[\mathsf{E}^{\mathsf{DES}}(K,m) = \mathsf{E}^{\mathsf{DES}}(K',m) \wedge K' \neq K;\ m \leftarrow_{\mathcal{R}} \mathcal{M}\right] \\
\overset{(2)}{=}\ & \sum_{K,K' \in \{0,1\}^{56}} \frac{1}{2^{56}} \cdot \begin{cases} 0 & \text{if } K = K' \\ \frac{1}{2^{64}} & \text{if } K \neq K' \end{cases} \\
\leq\ & \frac{2^{56} \cdot 2^{56}}{2^{56} \cdot 2^{64}} \\
=\ & 2^{-8}
\end{aligned}$$

For inequality (1) we exploited that the probability of a union is always less of equal than the sum of the probabilities of all elements of the union. Equality (2) holds because $\mathsf{E}^{\mathsf{DES}}$ was considered to be an ideal cipher: Consequently both $\mathsf{E}^{\mathsf{DES}}(K,m)$ and $\mathsf{E}^{\mathsf{DES}}(K',m)$ are (independent) uniformly random in $\{0,1\}^{64}$ over the choice of the random functions, provided that the keys are not equal. Thus the probability that both encryptions are equal is $\frac{1}{2^{64}}$. ■

Thus the unicity distance of DES, when considered as an ideal encryption, is at least $1 - 2^{-8}$.

Exhaustive Key Search in Practice: The DES Challenge was put forward by RSA Security to encourage research on the security of DES. A reward of 10000$ was offered for solving the challenge, i.e., for computing the key that was used to encrypt a specific plaintext/ciphertext pair of the form "`The unknown message is:  ----`". In 1997 the DESCHALL project needed about three months to break the DES challenge with a distributed search. In 1998 The Electronic Frontier Foundation (EFF) built the specialized hardware device *Deep Crack* that was able to break DES keys in roughly three days, at rather moderate costs of about 250.000$. While this is certainly too expensive for individuals, this amount is reasonable for large organizations or governments. Later the EFF and *distributed.net* together broke the challenge in 22 hours.

3.4.2 Linear Cryptanalysis

Suppose messages m and keys K are drawn uniformly at random from their respective sets. Suppose one knows $i_1, \ldots, i_r, j_1, \ldots, j_s, k_1, \ldots, k_t$ such that

$$\mathsf{Pr}\left[m^{(i_1)} \oplus \cdots \oplus m^{(i_r)} \oplus c^{(j_1)} \oplus \cdots \oplus c^{(j_s)} \oplus K^{(k_1)} \oplus \cdots \oplus K^{(k_t)} = 1\right] \geq \frac{1}{2} + \epsilon,$$

where $m^{(i)}, c^{(i)}, K^{(i)}$ denote the i-th bits of the bitstring m, the ciphertext c, and the key K, respectively. Such relations can be found for DES with $\epsilon \approx 2^{-21}$. If one gets enough plaintext/ciphertext pairs, one can obtain partial information about the key. Namely for $1/\epsilon^2$ plaintext/ciphertext pairs (m_l, c_l) one has

$$\mathsf{Pr}\left[K^{(k_1)} \oplus \cdots \oplus K^{(k_t)} = \mathsf{MAJ}\left(m_l^{(i_1)} \oplus \cdots \oplus m_l^{(i_r)} \oplus c_l^{(j_1)} \oplus \cdots \oplus c_l^{(j_s)} \mid l = 1, \ldots, 1/\epsilon^2\right)\right] \geq 97.7\%.$$

where MAJ denotes the majority function, taken over all Exclusive ORs computed from any given plaintext/ciphertext pair. One can calculate that obtaining 2^{42} plaintext/ciphertext pairs for DES allows for retrieving 14 bits of information of the key using these techniques. This enables an attacker to reduce the complexity of an exhaustive key search to 2^{42} DES operations, as one only has to test keys fulfilling the relation.

An even more involved method to break block ciphers is *differential cryptanalysis*, which we will not discuss here.

3.4.3 Side-channel Attacks

All the previous attacks are concerned with the input/output behavior of the algorithm only. This is the most common view, sometimes called the standard view, on cryptographic algorithms. However, in certain scenarios an attacker can gain access to more data. This can, for example, be the time needed to compute an encryption, the electric power consumption of a processor, or the electromagnetic emanation of a device, in particular of smartcards. Some of these attacks were able to completely break a system in seconds, and several smartcards have been withdrawn after it was discovered that they were highly vulnerable to such attacks.

One of the most famous side-channel attacks is due to Paul Kocher, who measured electric power consumption of microprocessors. He showed that by inspection of power consumption curves one can identify which conditional branch the executed code took. Consequently, if these conditions depend on the secret key, then information about the key is leaked. In fact, from several popular algorithms the key could quite easily be retrieved using such an attack.

3.5 Strengthening DES

Several methods for strengthening DES, in particular for increasing its key length, have been proposed. In the following, we present the most important ones.

3.5.1 Double-DES (2DES)

As a first attempt on strengthening DES, one might consider applying the encryption operation two times with different keys (K_1, K_2), i.e., computing $\mathsf{E}^{\mathsf{DES}}(K_1, \mathsf{E}^{\mathsf{DES}}(K_2, m))$. While the original intention would be to double the effective key length, this modification does not improve much on the security of DES, as there exists a "meet-in-the-middle" attack, which can be summarized as follows. Assume that we are given a plaintext/ciphertext pair (m, c), i.e., $c = \mathsf{E}^{\mathsf{DES}}(K_1, \mathsf{E}^{\mathsf{DES}}(K_2, m))$ for some K_1 and K_2.

1. Decrypt the given ciphertext c with every possible key $K_2^{(i)}$, where $1 \leq i \leq 2^{56}$. This yields a table with entries $(K_2^{(1)}, \mathsf{D}^{\mathsf{DES}}(K_2^{(1)}, c)), \dots, (K_2^{(2^{56})}, \mathsf{D}^{\mathsf{DES}}(K_2^{(2^{56})}, c))$. This step requires 2^{56} executions of DES.
2. Sort this table with respect to the second entry. This takes a number of steps in the order of $2^{56} \log(2^{56})$.
3. Encrypt the message m with any possible key $K_1^{(i)}$, where $1 \leq i \leq 2^{56}$, and look up the resulting encryption in the second column of the table, until a matching ciphertext $\mathsf{E}^{\mathsf{DES}}(K_1^{(i_1)}, m)$ is found. Let i_2 be the row containing such a ciphertext. This step takes at most 2^{56} executions of DES.
4. Let $K_2^{(i_2)}$ denote the key in the first column of the table at row i_2. Then the correct key is $(K_1^{(i_1)}, K_2^{(i_2)})$.

Since the sorting step does not produce a noticeable overhead over executing DES 2^{56} times, we ignore its cost in the overall complexity. Then one sees that the total number of DES executions is $2 \cdot 2^{56} = 2^{57}$, so the *effective key length* of Double-DES is at most 57 bits. Note that the above argument ignored the memory necessary to store the table, which is pretty large in the above example. So there is a trade-off between memory and time the meet-in-the-middle attack uses, but with still moderate memory usage the time can be reduced quite a lot.

Since Double-DES does not improve a lot over DES, Double-DES is rarely used.

3.5.2 Triple-DES (3DES)

A very common variant of DES is Triple-DES (3DES); in fact, this construction can be applied to any block cipher to increase its effective key length. One calculates 3DES as follows: Given three independent DES keys K_1, K_2, K_3, one computes

$$\mathsf{E}^{\mathsf{3DES}}((K_1, K_2, K_3), m) := \mathsf{E}^{\mathsf{DES}}(K_1, \mathsf{D}^{\mathsf{DES}}(K_2, \mathsf{E}^{\mathsf{DES}}(K_3, m))).$$

Decryption is given by

$$\mathsf{D}^{\mathsf{3DES}}((K_1, K_2, K_3), m) := \mathsf{D}^{\mathsf{DES}}(K_3, \mathsf{E}^{\mathsf{DES}}(K_2, \mathsf{D}^{\mathsf{DES}}(K_1, m))).$$

The key length is $3 \cdot 56 = 168$ bits. However, a meet-in-the-middle attack is possible:

- As for Double-DES, compute a table with entries $(K_3^{(1)}, \mathsf{D}^{\mathsf{DES}}(K_3^{(1)}, c)), \dots, (K_3^{(2^{56})}, \mathsf{D}^{\mathsf{DES}}(K_3^{(2^{56})}, c))$.
- Sort this table with respect to the second entry.
- For every possible pair of keys (K_1, K_2), compute $\mathsf{E}^{\mathsf{DES}}(K_1, \mathsf{D}^{\mathsf{DES}}(K_2, m))$ for the message m and look up this value in the table. This step needs 2^{112} DES operations.

As a consequence, the effective key length of 3DES is at most 112 bits.

3.5.3 DESX

Another variant is DESX, which is not that widely used although it provides an even larger effective key length than 3DES and is roughly three times faster. A key is a triple (K_1, K_2, K_3) where

$K_1, K_3 \in \{0,1\}^{64}$ and $K_2 \in \{0,1\}^{56}$. Thus K_2 is a DES key and K_1 and K_3 are random binary strings of the same length as the message blocks. Encryption and decryption are defined as

$$\mathsf{E}^{\mathsf{DESX}}((K_1, K_2, K_3), m) := K_3 \oplus \mathsf{E}^{\mathsf{DES}}(K_2, K_1 \oplus m), \text{ and}$$
$$\mathsf{D}^{\mathsf{DESX}}((K_1, K_2, K_3), c) := K_1 \oplus \mathsf{D}^{\mathsf{DES}}(K_2, K_3 \oplus c).$$

The following theorem was given by Kilian and Rogaway in 1998; it gives evidence that DESX has an effective key length of at least 119 bits. The proof is omitted and can be found in the original paper.

Theorem 3.3 (Kilian, Rogaway 1998) *Let (E, D) be an ideal cipher with key length l and block-size b. Let $(K_1, K_2, K_3) \in \{0,1\}^{2b+l}$, and let*

$$\mathsf{EX}((K_1, K_2, K_3), m) := K_3 \oplus \mathsf{E}(K_2, K_1 \oplus m)$$

a cipher having key length $k = 2b+l$. Then the effective key length of EX is at least $k-b-1 = b+l-1$ bit. In particular, the effective key length of (an idealized) DESX is at least $64 + 56 - 1 = 119$ bits. □

3.6 Modes of Operation

Given a block cipher $(\mathsf{K}, \mathsf{E}, \mathsf{D})$ with message space $\mathcal{M} = \{0,1\}^n$ and key space $\mathcal{K} := [\mathsf{K}]$, there are several ways it can be used to encrypt messages that are longer than a single block. FIPS PUB 81 defines four modes of operation (ECB, CBC, OFB, CFB); another mode is the counter mode, which has several variants. We will review these in the sequel.

All modes have in common that the key generation algorithm is the same as the key generation for the cipher, i.e., it generates a key $K \leftarrow \mathsf{K}$; we omit its description in the sequel.

3.6.1 Electronic Codebook Mode (ECB)

The *electronic codebook* (ECB) mode corresponds to the "naive" use of a block cipher. The message is split into a sequence $m = m_1, \ldots, m_t$ of blocks $m_i \in \mathcal{M}$. Each block m_i is encrypted with the same key K. Formally, the encryption $c = c_1, \ldots, c_t$ is computed as

$$c_i := \mathsf{E}(K, m_i).$$

The decryption operation is obvious. The ECB mode is depicted in Figure 3.10.

The main weakness of the ECB mode is that identical blocks m_i are encrypted to the same ciphertext c_i. This is a particularly strong weakness if messages come from a source with low entropy, e.g., securely encrypting (uncompressed) images is not possible using ECB. On the positive side, if one block c_i gets corrupted due to unreliable channels, then only one block is affected while all other blocks can still be decrypted.

3.6.2 Cipher Block Chaining Mode (CBC)

The *cipher block chaining* (CBC) mode overcomes the main weakness of ECB by using an *initialization vector IV* which is chosen uniformly at randomly from the set $\mathcal{M}$ in the encryption process

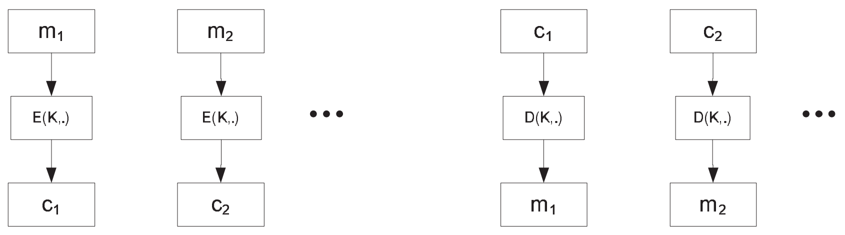

Figure 3.10: Encryption and Decryption in ECB Mode

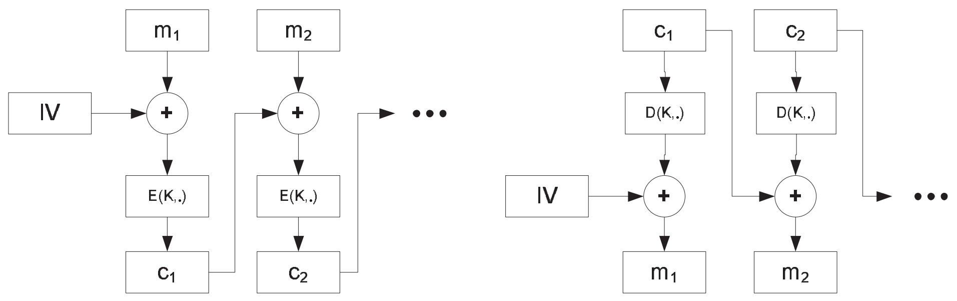

Figure 3.11: Encryption and Decryption in CBC Mode

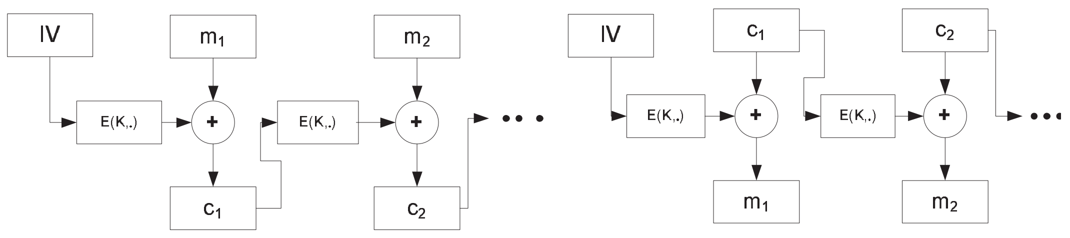

Figure 3.12: Encryption and Decryption in CFB Mode

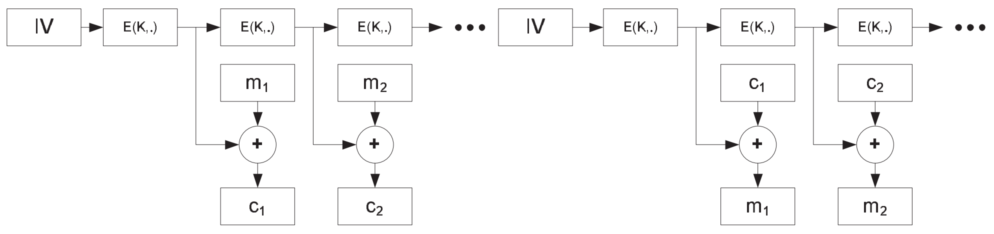

Figure 3.13: Encryption and Decryption in OFB Mode

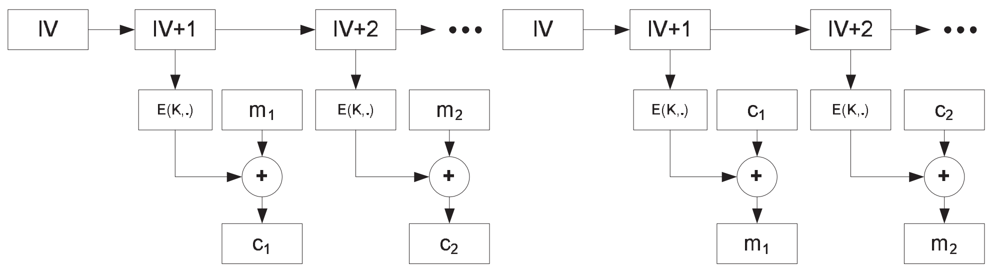

Figure 3.14: Encryption and Decryption in CTR Mode

as follows. A message $m = m_1, \ldots, m_t$ is encrypted by computing

$$\begin{aligned} c_1 &:= \mathsf{E}(K, m_1 \oplus IV) \text{ and} \\ c_i &:= \mathsf{E}(K, m_i \oplus c_{i-1}), \text{ for } i \geq 2, \end{aligned}$$

i.e., each ciphertext block c_i is Exclusively OR'ed with the next block of the plaintext. The ciphertext is $c = c_1, \ldots, c_t, IV$. The CBC mode is depicted in Figure 3.11. The decryption operation is straightforwardly derivable from the encryption; thus we omit its description.

For the CBC mode, a one-bit error in the ciphertext gives not only a one-block error in the corresponding message block but also a one-bit error in the next decrypted plaintext block. The initial value IV ensures that two encryptions of the same plaintext yield different ciphertexts, as opposed to the ECB mode.

3.6.3 Cipher Feedback Mode (CFB)

To encrypt a message $m = m_1, \ldots, m_t$ in *cipher feedback* (CFB) mode one first randomly chooses an initialization vector IV. The ciphertext $c = IV, c_1, \ldots, c_t$ is computed by appending the initialization vector IV to the c_i that are computed as follows:

$$\begin{aligned} c_1 &:= m_1 \oplus \mathsf{E}(K, IV), \text{ and} \\ c_i &:= m_i \oplus \mathsf{E}(K, c_{i-1}) \text{ for } i \geq 2. \end{aligned}$$

The decryption operation is defined in the straightforward manner. Note that the initialization vector needs to be included in the ciphertext, as otherwise the ciphertext cannot be decrypted.

A one-bit error in the ciphertext causes a one-bit error in the corresponding plaintext block and a block-error in the next decrypted plaintext block, i.e., the opposite of what happens in the CBC mode. The CFB is depicted in Figure 3.12.

3.6.4 Output Feedback Mode (OFB)

To encrypt a message $m = m_1, \ldots, m_t$ in *output feedback* (OFB) mode one randomly chooses an initialization vector IV, the ciphertext $c = IV, c_1, \ldots, c_t$ is computed as

$$c_i := m_i \oplus \mathsf{E}^i(K, IV).$$

where $\mathsf{E}^i(K, \cdot)$ means iterating the encryption operation i times. Of course, when implementing this mode one reuses previously computed iterations in the obvious way, see also in Figure 3.13. Decryption is computed analogously. Again, the initialization vector needs to be included in the ciphertext.

The OFB mode has the property that a one-bit error in the ciphertext gives only a one-bit error in the decrypted ciphertext. Furthermore, the key stream can be pre-computed, as it does not depend on the ciphertext. So this mode is good if latency between receiving a message and decrypting it is important.

A Note on the Initialization Vector (IV) We have defined the modes ECB, CBC, CFB, and OFB in such a way that the initialization vector is chosen uniformly at random and included in the ciphertext. However, the key requirement is that an initialization vector is not used twice, so one could define these modes such that subsequent initialization vectors are determined by a counter. Then the initialization vector does not have to be sent with the ciphertext; however, both the sender and the receiver have to synchronize their counters.

3.6.5 Counter Mode (CTR)

In contrast to the former modes of operations, the key stream is computed by applying the function E to values generated by a so-called counter, which can be essentially any function that will not repeat for a long time. In the following we will take a simple, deterministic counter. This mode comes in two variations, depending on how the initialization vector is chosen. The overall operation of the counter mode (both deterministic and randomized) is sketched in Figure 3.14. Let an *initialization vector IV* be given. (We will later explain how it may be chosen.) The encryption of a message $m = m_1, \ldots, m_l$, where each $m_i \in \mathcal{M}$, is given by $c = c_1, \ldots, c_l$ where

$$c_i := m_i \oplus \mathsf{E}(K, IV + i).$$

Here for computing the addition $IV + i$ both values are interpreted as values in $\mathbb{Z}_{2^n}$.

Deterministic Counter Mode (detCTR) In this variant the initialization vector is chosen to be $IV = 0$. We will see that this has the consequence that only one message can be encrypted securely, as we would re-use a value output by the function E when encrypting the second message.

Deterministic counter mode has very similar characteristics to OFB. A bit-error in ciphertext leads to a bit-error in the message; subsequent blocks are not affected. The key stream can be precomputed, and random access to the ciphertext is possible.

Randomized Counter Mode (randCTR) This variant does not use a fixed initialization vector, but randomly chooses a new one for every message that is encrypted. This value has to be sent along with the ciphertext c as constructed above, i.e., the actual ciphertext is (IV, c).

Randomized counter mode has very similar characteristics to detCTR and OFB. In contrast to detCTR we will see that it can be used to securely encrypt multiple messages. As for detCTR, the key stream can be precomputed, and random access to the ciphertext is possible.