CHAPTER 13

Block Ciphers and Modes of Operation

Chapter Goals

- To introduce the notion of block ciphers.
- To understand the workings of the DES algorithm.
- To understand the workings of the AES algorithm.
- To learn about the various standard modes of operation of block ciphers.

13.1. Introduction to Block Ciphers

The basic description of a block cipher is shown in Figure 13.1. Block ciphers operate on blocks

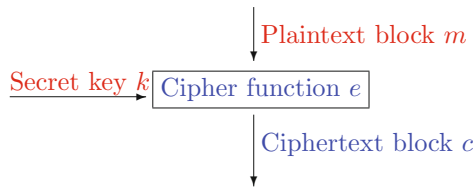

FIGURE 13.1. Operation of a block cipher

of plaintext one at a time to produce blocks of ciphertext. The block of plaintext and the block of ciphertext are assumed to be of the same size, e.g. a block of n bits. Every string of n bits in the domain should map to a string of n bits in the codomain, and every string of n bits in the codomain should result from the application of the function to a string in the domain. This means that for a fixed key a block cipher is bijective and hence is a permutation. We write

$$c \leftarrow e_k(m),$$
$$m \leftarrow d_k(c)$$

where

- $m \in \{0,1\}^n$ is the plaintext block,
- $k \in K$ is the secret key, chosen from key space K,
- e is the encryption function,
- d is the decryption function,
- $c \in \{0,1\}^b$ is the ciphertext block.

The block sizes taken are usually reasonably large, 64 bits in DES and 128 bits or more in modern block ciphers.

In terms of our prior definitions from Chapter 11, a block cipher should "act like" a family of pseudo-random permutations (PRPs), indexed by the key space K, $\{F_k\}_K$. We put the "act like"

N.P. Smart, *Cryptography Made Simple*, Information Security and Cryptography, DOI 10.1007/978-3-319-21936-3_13

in quotes, as for all existing (efficient) block ciphers used in practice we cannot mathematically prove that they are PRPs. Thus we can only "hope" that no adversary can break the PRP security game (Figure 11.4 of Chapter 11) for the specific block cipher[1]. In particular we hope that the advantage of an adversary against the PRP property is something like

$$\mathrm{Adv}^{\mathsf{PRP}}_{\{F_k\}_K}(A) \approx 1/|K|$$

for all adversaries A. So we require the key space to be rather large to ensure this advantage is small. But note that just because a block cipher has a large key space does not mean it will be a secure PRP.

Despite its name a block cipher *is not* an encryption scheme; it is a building block to create an encryption scheme. The term for how one creates an encryption scheme out of a block cipher is a *mode of operation*. The key advantage of this division, between a mode of operation and a block cipher design, is that we can design our modes and our block ciphers independently. As remarked, the design goal for a block cipher is that it is a secure pseudo-random permutation, whereas the design goal of a mode of operation is one of our security goals, such as IND-CCA. A designer of a mode of operation tries to prove mathematically that the mode satisfies the required security definition on the assumption that the block cipher is a secure PRP.

There are many block ciphers in use today, some which you may find used in your web browser; these include AES, CAMELLIA, DES or 3DES. The most famous of these is DES, or the Data Encryption Standard. This was first published in the mid-1970s as a US Federal standard and soon became the de facto international standard for banking applications.

The DES algorithm stood up remarkably well to the test of time, but in the early 1990s it became clear that a new standard was required. This was because both the block length (64 bits) and the key length (56 bits) of basic DES were too small for new applications. It is now possible to recover a 56-bit DES key using either a network of computers or specialized hardware for relatively little cost. Therefore DES has been phased out of most applications; although it still exists as a component in the variant called triple DES (3DES). In response to the problem of DES being deemed insecure, the US National Institute of Standards and Technology (NIST) initiated a competition to find a new block cipher, to be called the Advanced Encryption Standard or AES.

Unlike the process used to design DES, which was kept essentially secret, the design of the AES was performed in public. A number of groups from around the world submitted designs for the AES. Eventually five algorithms, known as the AES finalists, were chosen to be studied in depth. These were

- MARS from a group at IBM,
- RC6 from a group at RSA Security,
- Twofish from a group based at Counterpane, UC Berkeley and elsewhere,
- Serpent from a group of three academics based in Israel, Norway and the UK,
- Rijndael from a couple of Belgian cryptographers.

Finally in the fall of 2000, NIST announced that the overall AES winner had been chosen to be Rijndael, and so from hence forth Rijndael was known as AES.

DES and all the AES finalists are examples of *iterated* block ciphers. Block ciphers obtain their security by repeated use of a simple *round function*. The round function takes an n-bit block and returns an n-bit block, where n is the block size of the overall cipher. The number of rounds r can either be variable or fixed. As a general rule increasing the number of rounds will increase the level of security of the block cipher.

[1] An interesting side effect of our constructions is that we make no assumption on whether it is hard to break the block cipher given an oracle for the PRP in *both* the forwards or backwards directions. After reading this chapter you might want to consider why this is.

Each use of the round function employs a round key k_i for $1 \leq i \leq r$ derived from the main secret key k, using an algorithm called a *key schedule*. To allow decryption, for every round key the function implementing the round must be invertible, and for decryption the round keys are used in the opposite order to that in which they were used for encryption. That the whole round is invertible does not imply that the functions used to implement the round need to be invertible. This may seem strange at first reading but will become clearer when we discuss the DES cipher later. In DES the functions needed to implement the round function are not invertible, but the whole round is invertible. For AES not only is the whole round function invertible but every function used to create the round function is also invertible.

There are a number of general-purpose techniques which can be used to break a block cipher, for example: exhaustive search, using pre-computed tables of intermediate values or divide and conquer. Some (badly designed) block ciphers can be susceptible to chosen plaintext attacks, where encrypting a specially chosen plaintext can reveal properties of the underlying secret key. In cryptanalysis one needs a combination of mathematical and puzzle-solving skills, plus luck. There are a few more advanced techniques which can be employed:

- **Differential Cryptanalysis:** In differential cryptanalysis one looks at ciphertext pairs, where the corresponding plaintexts have a particular difference. The exclusive-or of such pairs is called a differential and certain differentials have certain probabilities associated with them, depending on what the key is. By analysing the probabilities of the differentials computed in a chosen plaintext attack one can hope to reveal the underlying structure of the key.
- **Linear Cryptanalysis:** Even though a good block cipher should contain non-linear components the idea behind linear cryptanalysis is to approximate the behaviour of the non-linear components with linear functions. Again the goal is to use a probabilistic analysis to determine information about the key.

Surprisingly these two methods are quite successful against some ciphers. Both DES and AES are designed to resist differential cryptanalysis, whereas AES is designed to also resist linear cryptanalysis.

Since DES and AES are likely to be the most important block ciphers in use for the next few years we shall study them in some detail. We also do this as they both show general design principles in their use of substitutions and permutations. Recall that the historical ciphers in Chapter 7 made use of such operations, so we see that not much has changed. Now, however, the substitutions and permutations used are far more intricate. On their own they do not produce security, but when used over a number of rounds one can obtain enough security for our applications.

We end this section by discussing the question, which is best, a block cipher or a stream cipher? The main difference between a block cipher and a stream cipher is that block ciphers are stateless, whilst stream ciphers maintain an internal state which is needed to determine which part of the keystream should be generated next. Here are just a few general points.

- Block ciphers are more general, and we shall see that one can easily turn a block cipher into a stream cipher.
- Stream cipher designs generally have a more mathematical structure. This either makes them easier to break or easier to study to convince oneself that they are secure.
- Stream ciphers are generally not suitable for software, since they usually encrypt one bit at a time. However, stream ciphers are highly efficient in hardware.
- Block ciphers are suitable for both hardware and software, but are generally not as fast in hardware as stream ciphers.

- Hardware is always faster than software, but this performance improvement comes at the cost of less flexibility.
- One can use block ciphers to build more complex functions via modes of operation, and rigorously analyse them if we assume the block cipher is a secure PRP.

13.2. Feistel Ciphers and DES

The DES cipher is a variant of the basic Feistel cipher described in Figure 13.2. Feistel ciphers are named after H. Feistel, who worked at IBM and performed some of the earliest non-military research on encryption algorithms. The interesting property of a Feistel cipher is that the round function is invertible regardless of the choice of the function in the box marked F. To see this notice that each encryption round is given by

$$L_i \leftarrow R_{i-1},$$
$$R_i \leftarrow L_{i-1} \oplus F(K_i, R_{i-1}).$$

Hence, the decryption can be performed via

$$R_{i-1} \leftarrow L_i,$$
$$L_{i-1} \leftarrow R_i \oplus F(K_i, L_i).$$

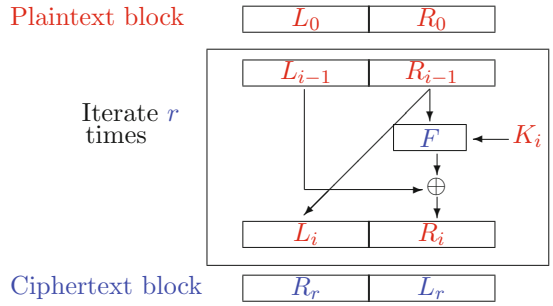

FIGURE 13.2. Basic operation of a Feistel cipher

This means that in a Feistel cipher we have simplified the design somewhat, since

- we can choose any function for the function F, and we will still obtain an encryption function which can be inverted using the secret key,
- the same code/circuitry can be used for the encryption and decryption functions. We only need to use the round keys in the reverse order for decryption.

Of course to obtain a secure cipher we still need to take care with

- how the round keys are generated,
- how many rounds to take,
- how the round function F is defined.

Work on DES was started in the early 1970s by a team in IBM which included Horst Feistel. It was originally based on an earlier cipher of IBM's called Lucifer, but some of the design was known to have been amended by the National Security Agency (NSA). For many years this led conspiracy theorists to believe that the NSA had placed a trapdoor in the design of the function F. However, it is now widely accepted that the modifications made by the NSA were done to make the cipher

more secure. In particular, the changes made by the NSA made the cipher resistant to differential cryptanalysis, a technique that was not discovered in the open research community until the 1980s.

DES is also known as the Data Encryption Algorithm (DEA) in documents produced by the American National Standards Institute (ANSI). The International Organization for Standardization (ISO) refers to DES by the name DEA-1. It was a worldwide standard for around thirty years and stands as the first publicly available cryptographic algorithm to have an "official status". It therefore marks an important step on the road from cryptography being a purely military area to being a tool for the masses. The use of DES is now no longer recommended on its own, and it has been withdrawn from all standards.

The basic properties of the DES cipher are that it is a variant of the Feistel cipher design in which

- the number of rounds r is 16,
- the block length n is 64 bits,
- the key length is 56 bits,
- the round keys $K_1, \ldots, K_{16}$ are each 48 bits.

Note that a key length of 56 bits is insufficient for many modern applications, hence often one uses DES by using three keys and three iterations of the main cipher. Such a version is called triple DES or 3DES; see Figure 13.3.

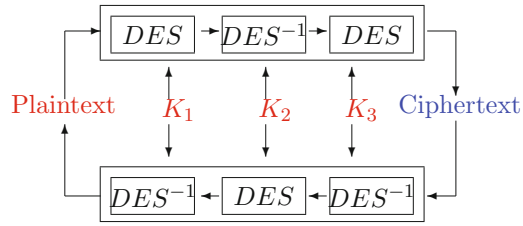

FIGURE 13.3. Triple DES

In 3DES the key length is equal to 168. There is another way of using DES three times, but using two keys instead of three giving rise to a key length of 112. In this two-key version of 3DES, one uses the 3DES basic structure but with the first and third key being equal. However, two-key 3DES is not as secure as one might inititally think. Intuitively one might suspect that it has a key size of 112 bits, however one can break it with 2^{64} effort.

Theorem 13.1. *Two-key 3DES can be broken in about 2^{64} time and about 2^{64} space, with a chosen plaintext attack.*

PROOF. The two-key variant of 3DES is given by the equation

$$c \leftarrow DES_{k_1}(DES_{k_2}{}^{-1}(DES_{k_1}(m))).$$

The technique to break this variant is a standard time/memory trade-off algorithm, which is very similar to the Baby-Step/Giant-Step algorithm of Chapter 3.

The attacker executes the following steps:

(1) For all $t_i \in K$ we compute $a_i \leftarrow DES_{t_i}{}^{-1}(0)$, where 0 is the all-zero message block,
(2) We store the 2^{64} tuples (a_i, t_i) in a table.
(3) For each tuple, we submit each a_i as a *plaintext* to our chosen plaintext attack oracle. We obtain
$$c_i \leftarrow DES_{k_1}(DES_{k_2}{}^{-1}(DES_{k_1}(a_i))).$$
(4) For each value c_i we compute $b_i \leftarrow DES_{t_i}{}^{-1}(c_i)$.

(5) Using the table, we look for pairs (a_j, b_i) for which $a_j = b_i$; this can be done fast for each b_i by sorting or hashing the initial table.
(6) Output (t_i, t_j) as a possible key, which can be tested with a few further chosen plaintext attack oracle queries.

To see why this attack works, consider the following series of identities,

$$\begin{aligned}
DES_{t_i}(DES_{t_j}{}^{-1}(DES_{t_i}(a_i))) &= DES_{t_i}(DES_{t_j}{}^{-1}(DES_{t_i}(DES_{t_i}{}^{-1}(0)))) && \text{by step one}\\
&= DES_{t_i}(DES_{t_j}{}^{-1}(0)) && \text{by definition of DES}\\
&= DES_{t_i}(a_j) && \text{by step one}\\
&= DES_{t_i}(b_i) && \text{by step five}\\
&= DES_{t_i}(DES_{t_i}{}^{-1}(c_i)) && \text{by step four}\\
&= c_i && \text{by definition of DES.}
\end{aligned}$$

This is exactly what the chosen plaintext oracle outputs. Thus it is highly likely that $(k_1, k_2) = (t_i, t_j)$, which can be confirmed by encrypting a few more plaintexts as in step six. □

A similar time/memory trade-off can be applied to the full 3DES algorithm. However, the result is that the effective complexity is 2^{112}. Thus 3DES is still considered secure, just not as secure as one would expect from its key size of 168 bits, but we have the expectation that

$$\mathrm{Adv}^{\mathsf{PRP}}_{\{3DES_k\}_K}(A) \approx 1/2^{112}.$$

13.2.1. Overview of DES Operation: Basically DES is a Feistel cipher with 16 rounds, except that before and after the main Feistel iteration a permutation is performed, as depicted in Figure 13.4. This permutation appears to produce no change in the security, and people have often wondered why it is there. One answer given by one of the original team members was that this permutation was there to make the original implementation easier to fit on the circuit board.

In summary the DES cipher operates on 64 bits of plaintext in the following manner:

- Perform an initial permutation.
- Split the blocks into left and right half.
- Perform 16 rounds of identical operations.
- Join the half blocks back together.
- Perform a final inverse permutation.

The final permutation is the inverse of the initial permutation; this allows the same hardware and/or software to be used for encryption and decryption. The key schedule provides 16 round keys of 48 bits in length by selecting 48 bits from the 56-bit main key. We shall now describe the operation of the function F in more detail. In each DES round this consists of the following six stages:

- **Expansion Permutation:** The right half of 32 bits is expanded and permuted to 48 bits. This helps the diffusion of any relationship of input bits to output bits. The expansion permutation (which is different from the initial permutation) has been chosen so that one bit of input affects two substitutions in the output, via the S-Boxes below. This helps spread dependencies and creates an avalanche effect (a small difference between two plaintexts will produce a very large difference in the corresponding ciphertexts).
- **Round Key Addition:** The 48-bit output from the expansion permutation is exclusive-or'd with the round key, which is also 48 bits in length. Note that this is the only place where the round key is used in the algorithm.
- **Splitting:** The resulting 48-bit value is split into eight lots of six-bit values.

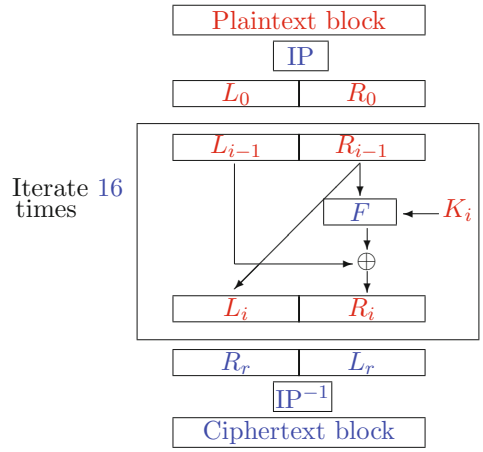

FIGURE 13.4. DES as a Feistel cipher

- **S-Boxes:** Each six-bit value is passed into one of eight different S-Boxes (Substitution Box) to produce a four-bit result. The S-Boxes represent the non-linear component in the DES algorithm and their design is a major contributor to the algorithm's security. Each S-Box is a look-up table of four rows and sixteen columns. The six input bits specify which row and column to use. Bits 1 and 6 generate the row number, whilst bits $2, 3, 4$ and 5 specify the column number. The output of each S-Box is the value held in that element in the table.
- **P-Box:** We now have eight lots of four-bit outputs which are then combined into a 32-bit value and permuted to form the output of the function F.

The overall structure of the DES F function is explained in Figure 13.5.

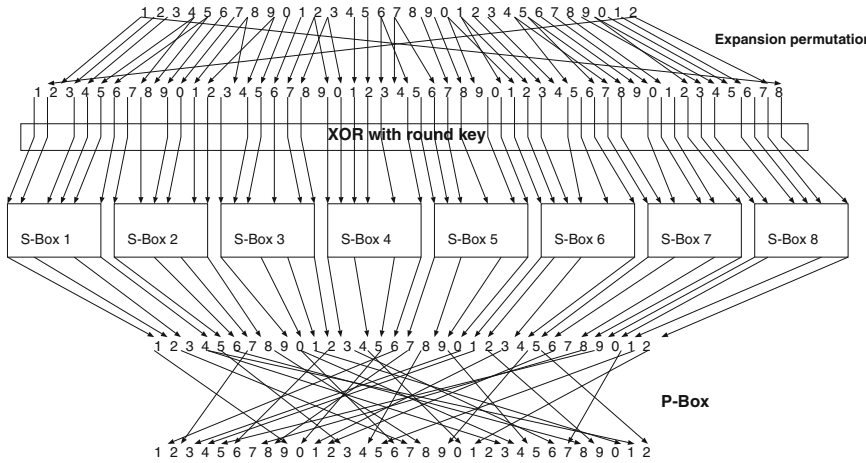

FIGURE 13.5. Structure of the DES function F

We now give details of each of the steps which we have not yet fully defined.

Initial Permutation IP: The DES initial permutation is defined in the following table. Here the 58 in the first position means that the first bit of the output from the IP is the 58th bit of the input, and so on.

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

The inverse permutation is given in a similar manner by the following table.

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

Expansion Permutation E: The expansion permutation is given in the following table. Each row corresponds to the bits which are input into the corresponding S-Box at the next stage. Notice how the bits which select the row of one S-Box (the first and last bit on each row) are also used to select the column of another S-Box.

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

S-Box: The details of the eight DES S-Boxes are given in Figure 13.6. Recall that each box consists of a table with four rows and sixteen columns.

The P-Box Permutation P: The P-Box permutation takes the eight lots of four-bit nibbles, output by the S-Boxes, and produces a 32-bit permutation of these values as given by the following table.

S-Box 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-Box 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-Box 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-Box 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-Box 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-Box 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-Box 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-Box 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 13.6. DES S-Boxes

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

DES Key Schedule: The DES key schedule takes the 56-bit key, which is actually input as a bitstring of 64 bits comprising of the key and eight parity bits, for error detection. These parity bits are in bit positions $8, 16, \ldots, 64$ and ensure that each byte of the key contains an odd number of bits set to one. We first permute the bits of the key according to the following permutation (which takes a 64-bit input and produces a 56-bit output, hence discarding the parity bits).

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

The output of this permutation, called PC-1 in the literature, is divided into a 28-bit left half C_0 and a 28-bit right half D_0. Now for each round we compute

$$C_i \leftarrow C_{i-1} \lll p_i,$$
$$D_i \leftarrow D_{i-1} \lll p_i,$$

where $x \lll p_i$ means perform a cyclic shift on x to the left by p_i positions. If the round number i is $1, 2, 9$ or 16 then we shift left by one position, otherwise we shift left by two positions. Finally the two portions C_i and D_i are joined back together and are subject to another permutation, called PC-2, to produce the final 48-bit round key. The permutation PC-2 is described below.

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

13.3. AES

The AES winner was decided in autumn 2000 to be the Rijndael algorithm designed by Joan Daemen and Vincent Rijmen. AES is a block cipher which does not rely on the basic design of the Feistel cipher; instead it is designed as a *substitution-permutation* network, or SP-network. However, AES does have a number of similarities with DES. Block ciphers based on the SP-network design consist

of a series of rounds, each of which consists of a key addition phase, a substitution phase and a permutation phase. The idea is that the permutation phase aims to produce an avalanche effect, by spreading out differences in the input to other parts of the state as quickly as possible, performing a process called *diffusion*. The substitution phase is the main non-linear component and this aims to introduce as much non-linearity, or *confusion*, into the output as possible.

AES has, unlike DES, a strong mathematical structure, as most of its operations are based on arithmetic in the finite fields $\mathbb{F}_{2^8}$ and $\mathbb{F}_2$. However, unlike DES the encryption and decryption operations are distinct, and do not just require a re-ordering of the round keys.

Recall from Chapter 6 that elements of $\mathbb{F}_{2^8}$ are stored as bit vectors (or bytes) representing binary polynomials. For example the byte given by `0x83` in hexadecimal gives the bit pattern

$$1, 0, 0, 0, 0, 0, 1, 1$$

since

$$\texttt{0x83} = 8 \cdot 16 + 3 = 131$$

in decimal. One can obtain the bit pattern directly by noticing that 8 in binary is $1, 0, 0, 0$ and 3 in 4-bit binary is $0, 0, 1, 1$ and one simply concatenates these two bit strings together. The bit pattern itself then corresponds to the binary polynomial

$$x^7 + x + 1.$$

So we say that the hexadecimal number `0x83` represents the binary polynomial

$$x^7 + x + 1.$$

Arithmetic in $\mathbb{F}_{2^8}$ in the AES algorithm is performed using polynomial arithmetic modulo the irreducible polynomial

$$m(x) = x^8 + x^4 + x^3 + x + 1.$$

AES identifies 32-bit words with polynomials in $\mathbb{F}_{2^8}[X]$ of degree less than four. This is done in a big-endian format, in that the smallest index corresponds to the least important coefficient. Hence, the word

$$a_0 \| a_1 \| a_2 \| a_3$$

will correspond to the polynomial

$$a_3 \cdot X^3 + a_2 \cdot X^2 + a_1 \cdot X + a_0.$$

Arithmetic is performed on polynomials in $\mathbb{F}_{2^8}[X]$ modulo the reducible polynomial

$$M(X) = X^4 + 1.$$

Hence, arithmetic is done on these polynomials in a ring rather than a field, since $M(X)$ is reducible.

Rijndael was a parametrized algorithm, in that it could operate on block sizes of 128, 192 or 256 bits, but in the final AES standard the block size was fixed at 128 bits. However, AES does support keys of size 128, 192 or 256 bits. For each key size a different number of rounds is specified. To make our discussion simpler we shall consider the simpler, and probably more used, variant which uses a block size of 128 bits and a key size of 128 bits, in which case 10 rounds are specified. From now on our discussion is only of this simpler version.

AES operates on an internal four-by-four matrix of bytes, called the state matrix

$$S = \begin{pmatrix} s_{0,0} & s_{0,1} & s_{0,2} & s_{0,3} \\ s_{1,0} & s_{1,1} & s_{1,2} & s_{1,3} \\ s_{2,0} & s_{2,1} & s_{2,2} & s_{2,3} \\ s_{3,0} & s_{3,1} & s_{3,2} & s_{3,3} \end{pmatrix},$$

which is usually held as a vector of four 32-bit words, each word representing a column. Each round key is also held as a four-by-four matrix

$$K_i = \begin{pmatrix} k_{0,0} & k_{0,1} & k_{0,2} & k_{0,3} \\ k_{1,0} & k_{1,1} & k_{1,2} & k_{1,3} \\ k_{2,0} & k_{2,1} & k_{2,2} & k_{2,3} \\ k_{3,0} & k_{3,1} & k_{3,2} & k_{3,3} \end{pmatrix}.$$

13.3.1. AES Operations: The AES round function operates using a set of four operations which we shall now describe.

SubBytes: Two types of S-Boxes are used in AES: one for the encryption rounds and one for the decryption rounds, each one being the inverse of the other. We shall describe the encryption S-Box; the decryption one will follow immediately. The S-Boxes of DES were chosen by searching through a large space of possible S-Boxes, so as to avoid attacks such as differential cryptanalysis. The S-Box of AES is chosen to have a simple mathematical structure, which allows one to formally argue how resilient the cipher is to differential and linear cryptanalysis. Not only does this mathematical structure help protect against differential cryptanalysis, but it also convinces users that it has not been engineered with some hidden trapdoor.

Each byte $s = [s_7, \ldots, s_0]$ of the AES state matrix is taken in turn and considered as an element of $\mathbb{F}_{2^8}$. The S-Box can be mathematically described in two steps:

(1) The multiplicative inverse of s in $\mathbb{F}_{2^8}$ is computed, to produce a new byte $x = [x_7, \ldots, x_0]$. For the element $[0, \ldots, 0]$, which has no multiplicative inverse, one uses the convention that this is mapped to zero, so as to maintain a one-to-one mapping from the input to the output of the S-Box.
(2) The bitvector x is then mapped, via the following affine $\mathbb{F}_2$ transformation, to the bitvector y:

$$\begin{pmatrix} y_0 \\ y_1 \\ y_2 \\ y_3 \\ y_4 \\ y_5 \\ y_6 \\ y_7 \end{pmatrix} \leftarrow \begin{pmatrix} 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 \end{pmatrix} \cdot \begin{pmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \\ x_6 \\ x_7 \end{pmatrix} \oplus \begin{pmatrix} 1 \\ 1 \\ 0 \\ 0 \\ 0 \\ 1 \\ 1 \\ 0 \end{pmatrix}.$$

The new byte is given by y. The decryption S-Box is obtained by first inverting the affine transformation and then taking the multiplicative inverse. These byte substitutions can either be implemented using table look-up or by implementing circuits, or code, which implement the inverse operation in $\mathbb{F}_{2^8}$ and the affine transformation.

ShiftRows: The ShiftRows operation in AES performs a cyclic shift on the state matrix. Each row is shifted by different offsets. For AES this is given by

$$\begin{pmatrix} s_{0,0} & s_{0,1} & s_{0,2} & s_{0,3} \\ s_{1,0} & s_{1,1} & s_{1,2} & s_{1,3} \\ s_{2,0} & s_{2,1} & s_{2,2} & s_{2,3} \\ s_{3,0} & s_{3,1} & s_{3,2} & s_{3,3} \end{pmatrix} \longmapsto \begin{pmatrix} s_{0,0} & s_{0,1} & s_{0,2} & s_{0,3} \\ s_{1,1} & s_{1,2} & s_{1,3} & s_{1,0} \\ s_{2,2} & s_{2,3} & s_{2,0} & s_{2,1} \\ s_{3,3} & s_{3,0} & s_{3,1} & s_{3,2} \end{pmatrix}.$$

The inverse of the ShiftRows operation is simply the equivalent shift in the opposite direction. The ShiftRows operation ensures that the columns of the state matrix "interact" with each other over a number of rounds.

MixColumns: The MixColumns operation ensures that the rows in the state matrix "interact" with each other over a number of rounds; combined with the ShiftRows operation it ensures each byte of the output state depends on each byte of the input state. We consider each column of the state $[a_0, a_1, a_2, a_3]$ in turn, and consider it as a polynomial of degree less than four with coefficients in $\mathbb{F}_{2^8}$. The new column $[b_0, b_1, b_2, b_3]$ is produced by taking the polynomial

$$a(X) = a_0 + a_1 \cdot X + a_2 \cdot X^2 + a_3 \cdot X^3$$

and multiplying it by the polynomial

$$c(X) = \texttt{0x02} + \texttt{0x01} \cdot X + \texttt{0x01} \cdot X^2 + \texttt{0x03} \cdot X^3$$

modulo

$$M(X) = X^4 + 1.$$

This operation is conveniently represented by the following matrix operation in $\mathbb{F}_{2^8}$,

$$\begin{pmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \end{pmatrix} \leftarrow \begin{pmatrix} \texttt{0x02} & \texttt{0x03} & \texttt{0x01} & \texttt{0x01} \\ \texttt{0x01} & \texttt{0x02} & \texttt{0x03} & \texttt{0x01} \\ \texttt{0x01} & \texttt{0x01} & \texttt{0x02} & \texttt{0x03} \\ \texttt{0x03} & \texttt{0x01} & \texttt{0x01} & \texttt{0x02} \end{pmatrix} \cdot \begin{pmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \end{pmatrix}.$$

In $\mathbb{F}_{2^8}$ the above matrix is invertible, hence the inverse of the MixColumns operation can also be implemented using a matrix multiplication such as that above.

AddRoundKey: The round key addition is particularly simple. One takes the state matrix and exclusive-or's it, byte by byte, with the round key matrix. The inverse of this operation is clearly the same operation.

Round Structure: The AES algorithm can now be described using the pseudo-code in Algorithm 13.1. The message block to encrypt is assumed to be entered into the state matrix S, the output encrypted block is also given by the state matrix S. Notice that the final round does not perform a MixColumns operation. The corresponding decryption operation is described in Algorithm 13.2.

Algorithm 13.1: AES encryption outline

```
AddRoundKey(S, K_0).
for i = 1 to 9 do
    SubBytes(S).
    ShiftRows(S).
    MixColumns(S).
    AddRoundKey(S, K_i).
SubBytes(S).
ShiftRows(S).
AddRoundKey(S, K_10).
```

Algorithm 13.2: AES decryption outline

AddRoundKey(S, K_{10}).
InverseShiftRows(S).
InverseSubBytes(S).
for $i = 9$ **downto** 1 **do**
  AddRoundKey(S, K_i).
  InverseMixColumns(S).
  InverseShiftRows(S).
  InverseSubBytes(S).
AddRoundKey(S, K_0).

AES Key Schedule: The only thing left to describe is how AES computes the round keys from the main key. Recall that the main key is 128 bits long, and we need to produce 11 round keys $K_0, \ldots, K_{11}$ all of which consist of four 32-bit words, each word corresponding to a column of a matrix as described above. The key schedule makes use of a round constant which we shall denote by

$$RC_i \leftarrow x^i \pmod{x^8 + x^4 + x^3 + x + 1}.$$

We label the round keys as $(W_{4i}, W_{4i+1}, W_{4i+2}, W_{4i+3})$ where i is the round. The initial main key is first divided into four 32-bit words (k_0, k_1, k_2, k_3). The round keys are then computed as in Algorithm 13.3, where RotBytes is the function which rotates a word to the left by a single byte, and SubBytes applies the AES encryption S-Box to every byte in a word.

Algorithm 13.3: AES key schedule

$W_0 \leftarrow K_0, W_1 \leftarrow K_1, W_2 \leftarrow K_2, W_3 \leftarrow K_3$.
for $i \leftarrow 1$ **to** 10 **do**
  $T \leftarrow$RotBytes(W_{4i-1}).
  $T \leftarrow$SubBytes(T).
  $T \leftarrow T \oplus RC_i$.
  $W_{4i} \leftarrow W_{4i-4} \oplus T$.
  $W_{4i+1} \leftarrow W_{4i-3} \oplus W_{4i}$.
  $W_{4i+2} \leftarrow W_{4i-2} \oplus W_{4i+1}$.
  $W_{4i+3} \leftarrow W_{4i-1} \oplus W_{4i+2}$.

13.4. Modes of Operation

A block cipher like DES or AES can be used in a variety of ways to encrypt a data string. Soon after DES was standardized another US Federal standard appeared giving four *recommended* ways of using DES for data encryption. These modes of operation have since been standardized internationally and can be used with any block cipher. The four modes are

- **ECB Mode:** This is simple to use, but suffers from possible deletion and insertion attacks. A one-bit error in the ciphertext gives a one whole block error in the decrypted plaintext.
- **CBC Mode:** This is probably the best of the original four modes of operation, since it helps protect against deletion and insertion attacks. In this mode a one-bit error in the ciphertext gives not only a one-block error in the corresponding plaintext but also a one-bit error in the next decrypted plaintext block.

- **OFB Mode:** This mode turns a block cipher into a stream cipher. It has the property that a one-bit error in the ciphertext gives a one-bit error in the decrypted plaintext.
- **CFB Mode:** This mode also turns a block cipher into a stream cipher. A single bit error in the ciphertext affects both this block and the next, just as in CBC mode.

Over the years various other modes of operation have been presented. Probably the most popular of the more modern modes is

- **CTR Mode:** This also turns the block cipher into a stream cipher, but it enables blocks to be processed in parallel, thus providing performance advantages when parallel processing is available.

We shall now describe each of these five modes of operation in detail, and show what security properties each has (or in most cases has not). Finally, we present a summary of these five basic modes of operation in Tables 13.1 and 13.2. Throughout this section we ignore difficulties related to which padding scheme should be used to pad a message out to a multiple of the block length. We discuss padding schemes in Chapter 14. In the following discussion we let $\mathsf{ECB}[F], \mathsf{CBC}[F], \mathsf{OFB}[F], \mathsf{CFB}[F]$ and $\mathsf{CTR}[F]$ denote the mode of operation when instantiated with the function F, which could be a block cipher, a pseudo-random permutation or a pseudo-random function depending on the situation we are considering.

13.4.1. ECB Mode: Electronic Code Book Mode, or ECB Mode, is the simplest way to use a block cipher. The data to be encrypted m is divided into blocks of n bits:

$$m_1, m_2, \ldots, m_q$$

with the last block padded if needed. The ciphertext blocks $c_1, \ldots, c_q$ are then defined as follows

$$c_i \leftarrow e_k(m_i),$$

as described in Figure 13.7. Decipherment is simply the reverse operation as explained in Figure 13.8.

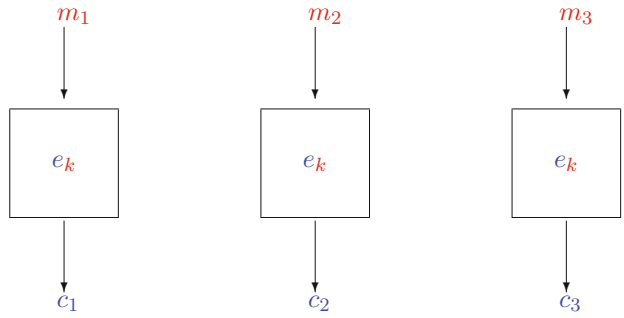

FIGURE 13.7. ECB encipherment

ECB Mode has a number of problems: the first is due to the property that if $m_i = m_j$ then we have $c_i = c_j$, i.e. the same input block always generates the same output block. This is a problem in practice since stereotyped beginnings and endings of messages are common. The second problem comes because we could simply delete blocks from the message and no one would know. Thirdly we could replay known blocks from other messages. By extracting ciphertext corresponding to a known piece of plaintext we can then amend other transactions to contain this known block of text. In terms of our security models from Chapter 11 we have the following.

Theorem 13.2. *ECB Mode* is not IND-PASS *secure.*

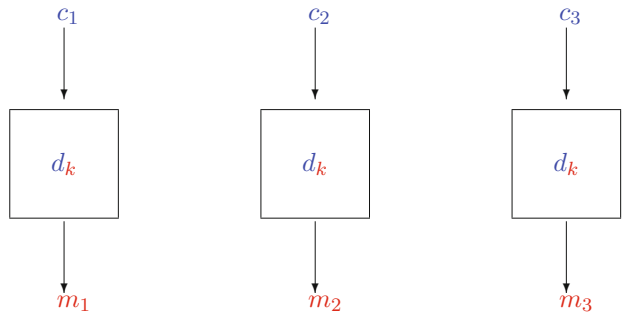

FIGURE 13.8. ECB decipherment

PROOF. To prove something is not secure we need to exhibit an attack within the model. The attack on ECB mode is very simple:

- Let $\mathbf{0}$ denote the block of all zeros, and $\mathbf{1}$ denote the block of all ones.
- Call the $\mathcal{O}_{LR}$ oracle with $m_0 = \mathbf{0}\|\mathbf{1}$ and $m_1 = \mathbf{1}\|\mathbf{1}$. The challenge ciphertext c^* is returned which is the encryption of m_b, for the hidden bit b. The challenge ciphertext consists of two blocks c_0 and c_1
- If $c_0 \neq c_1$ then output $b' = 0$, else return $b' = 1$.

□

It is clear this attack works with a one hundred percent success rate, this is due to the fact that ECB Mode is deterministic. One should also note that this attack is stronger than the standard attack on deterministic encryption schemes (which require access to an encryption oracle). The ECB attack does not require such an oracle. Even if we restrict ourseleves to a notion of one-way security ECB mode is not as secure as we would like

Theorem 13.3. *ECB Mode* is not OW-CCA *secure.*

PROOF. Again, to prove something is not secure we need to exhibit an attack within the model, and again the attack is very simple:

- Let c^* denote the target ciphertext to be decrypted.
- Let r denote an arbitrary block, and form the ciphertext $c \leftarrow c^*\|r$.
- Pass c to the decryption oracle $\mathcal{O}_{d_k}$ to obtain the plaintext $m^*\|s$ for some block s.
- Return m^*.

□

However, we can show the following positive result

Theorem 13.4. *ECB mode* is OW-CPA *secure assuming the underlying block cipher* e_k *acts like a pseudo-random permutation. In particular, let* A *denote an adversary against ECB Mode which makes* q_e *queries to its encryption oracle, where each query is a single block in length, and where the challenge ciphertext is* ℓ distinct *blocks in length*[2]*. There is an adversary* B *such that*

$$\mathrm{Adv}^{\mathsf{OW\text{-}CPA}}_{\mathsf{ECB}[e_k]}(A; q_e) \leq \mathrm{Adv}^{\mathsf{PRP}}_{e_k}(B) + \frac{\ell \cdot q_e}{2^n}.$$

where n *is the block size of the cipher* e_k*.*

[2]These assumptions can be changed by making a suitably more complex probability estimate.

PROOF. Our main step in the proof is to replace the underlying block cipher e_k by a pseudo-random permutation. This can be done by the assumption that e_k is a secure PRP, namely there is some adversary B such that

$$\text{Adv}^{\mathsf{PRP}}_{e_k}(B) = \Big|\Pr[A \text{ wins } \mathsf{ECB}[e_k]] - \Pr[A \text{ wins } \mathsf{ECB}[\mathcal{P}]]\Big| \tag{18}$$

where we let $\mathcal{P}$ denote a random permutation. We now need to bound the probability that A wins this latter game, i.e. $\Pr[A \text{ wins } \mathsf{ECB}[\mathcal{P}]]$. It is easier to bound the probability that A does not win. Since for a PRP the adversary cannot learn anything about the output value of the permutation until she queries the permutation on the specific input value, the probability that she does not win is given by the probability that out of the q_e distinct queries to the encryption oracle we do not obtain *all* of the ℓ blocks in the challenge ciphertext. Setting $N = 2^n$ this gives us, where ${}^xC_y$ is the function which returns the number of combinations of y objects selected from n,

$$\begin{aligned}
\Pr[A \text{ wins } \mathsf{ECB}[\mathcal{P}]] &= 1 - \Pr[A \text{ does not win } \mathsf{ECB}[\mathcal{P}]] \\
&\leq 1 - \frac{\ell \cdot {}^{N-1}C_{q_e}}{{}^{N}C_{q_e}} \\
&= 1 - \ell \cdot \left(\frac{(N-1)!}{q_e! \cdot (N-1-q_e)!}\right) \cdot \left(\frac{q_e! \cdot (N-q_e)!}{N!}\right) \\
&= 1 - \ell \cdot \left(\frac{N-q_e}{N}\right) = \frac{N - \ell \cdot (N-q_e)}{N} \\
&\leq \ell \cdot q_e / N.
\end{aligned}$$

Hence,

$$\begin{aligned}
\Pr[A \text{ wins } \mathsf{ECB}[e_k]] &= \Big|\Pr[A \text{ wins } \mathsf{ECB}[e_k]] \\
&\qquad + (\Pr[A \text{ wins } \mathsf{ECB}[\mathcal{P}]] - \Pr[A \text{ wins } \mathsf{ECB}[\mathcal{P}]])\Big| && \text{adding in zero} \\
&\leq \Big|\Pr[A \text{ wins } \mathsf{ECB}[e_k]] - \Pr[A \text{ wins } \mathsf{ECB}[\mathcal{P}]]\Big| \\
&\qquad + \Big|\Pr[A \text{ wins } \mathsf{ECB}[\mathcal{P}]]\Big| && \text{triangle inequality} \\
&\leq \text{Adv}^{\mathsf{PRP}}_{e_k}(B) + \ell \cdot \frac{q_e}{2^n}.
\end{aligned}$$

□

Notice that when q_e is small relative to 2^n the probability $q_e/2^n$ is very close to zero, whereas as q_e approaches 2^n we obtain a probability close to one.

13.4.2. CBC Mode: One way of countering the problems with ECB Mode is to chain the cipher, and in this way add context to each ciphertext block. The easiest way of doing this is to use Cipher Block Chaining Mode, or CBC Mode. Again, the plaintext must first be divided into a series of blocks

$$m_1, \ldots, m_q,$$

and as before the final block may need padding to make the plaintext length a multiple of the block length. Encryption is then performed as in Figure 13.9, or equivalently via the equations

$$\begin{aligned}
c_1 &\leftarrow e_k(m_1 \oplus IV), \\
c_i &\leftarrow e_k(m_i \oplus c_{i-1}) \text{ for } i > 1.
\end{aligned}$$

With the output ciphertext being $IV \| c_1 \| c_2 \| \ldots$. The transmission of IV with the ciphertext can be dropped if the receiver will know what the value will be a priori.

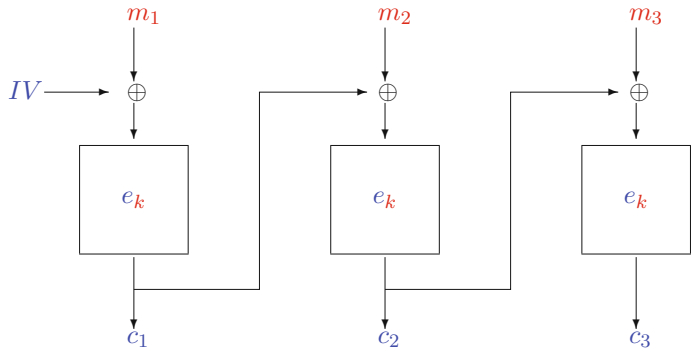

FIGURE 13.9. CBC encipherment

Decryption also requires the IV and is performed as in Figure 13.10, or via the equations

$$m_1 \leftarrow d_k(c_1) \oplus IV,$$
$$m_i \leftarrow d_k(c_i) \oplus c_{i-1} \text{ for } i > 1.$$

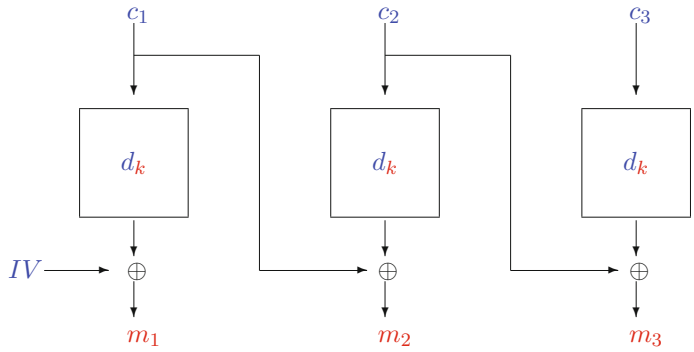

FIGURE 13.10. CBC decipherment

With ECB Mode a single-bit error in transmission of the ciphertext will result in a whole block being decrypted wrongly, whilst in CBC Mode we see that not only will we decrypt a block incorrectly but the error will also affect a single bit of the next block.

Notice that we require an additional initial value IV. This is either a unique, i.e. never repeated, value passed to the encryption function (in which case we are said to have a nonce-based encryption scheme), or it is fixed to a specific value (in which case we have a deterministic variant of CBC Mode), or it is a truly random value chosen internally by the mode of operation. Each of these choices leads to entirely different security properties as we shall now demonstrate. When a non-fixed IV is used the IV value is prepended to the ciphertext.

Fixed IV: Clearly with a fixed IV CBC Mode is deterministic and hence cannot be IND-CPA secure, however one can show it is IND-PASS secure (unlike ECB Mode). This follows from the proof method of IND-CPA security in the random IV case given below. The crucial point to note is that whilst the adversary has control over the input to the first call to the block cipher, he has no control over the other calls when encrypting a multi-block message.

Nonce IV: Here we think of CBC Mode as a nonce-based encryption scheme, as in Section 11.6.4. We start with the negative result

Theorem 13.5. *With a nonce as the IV, CBC Mode* is not IND-CPA *secure.*

PROOF. Let $\mathbf{0}$ be the all-zero block and $\mathbf{1}$ be the all-one block. The attack on the IND-CPA security is as follows:

- Send the message $\mathbf{0}$ with the nonce $IV = \mathbf{0}$ to the encryption oracle $\mathcal{O}_{e_k}$. The adversary obtains the ciphertext $\mathbf{0}\|c$ in return, where $c = e_k(\mathbf{0})$.
- Now send the messages $m_0 = \mathbf{0}$ and $m_1 = \mathbf{1}$ to the $\mathcal{O}_{\mathsf{LR}}$ oracle, with nonce $\mathbf{1}$. Notice this is a new nonce and so the encryption is allowed in the game. Let $\mathbf{1}\|c^*$ be the returned ciphertext.
- If $c^* = c$ then return $b' = 1$, else return $b' = 0$.

To see why this attack works, note that if the hidden bit is $b = 1$ then the challenger returns c^* which is the evaluation of the block cipher on the block $\mathbf{1} \oplus \mathbf{1} = \mathbf{0}$. Whereas if $b = 0$ then the evaluation is on the block $\mathbf{0} \oplus \mathbf{1} = \mathbf{1}$. □

On the positive side, when used only once nonce-based encryption is identical to a fixed IV, and so CBC Mode used in a nonce-based encryption methodology is IND-PASS secure.

Random IV: With a random IV we can be more positive, since CBC Mode is IND-CPA secure as we will now show.

Theorem 13.6. *With a random IV, CBC Mode is* IND-CPA *secure assuming the underlying block cipher e_k acts like a pseudo-random permutation. In particular let A denote an adversary against CBC Mode which makes q_e queries to its encryption oracle, and let all plaintext submitted to both the* LR *and encryption oracles be at most ℓ blocks in length. Then there is an adversary B such that*

$$\mathrm{Adv}^{\mathsf{IND\text{-}CPA}}_{\mathsf{CBC}[e_k]}(A; q_e) \leq \mathrm{Adv}^{\mathsf{PRP}}_{e_k}(B) + \frac{3 \cdot T^2}{2^n},$$

where n is the block size of the cipher e_k and $T = (q_e + 1) \cdot \ell$.

PROOF. In the security game the challenger needs to call the underlying block cipher on behalf of the adversary. The total number of such calls is bounded by $T = (q_e + 1) \cdot \ell$.

Our first step in the proof is to replace the underlying block cipher e_k by a pseudo-random permutation. This can be done by the assumption that e_k is a secure PRP, namely there is some adversary B such that

$$\mathrm{Adv}^{\mathsf{PRP}}_{e_k}(B) = \Big|\Pr[A \text{ wins } \mathsf{CBC}[e_k]] - \Pr[A \text{ wins } \mathsf{CBC}[\mathcal{P}]]\Big| \tag{19}$$

where we let $\mathcal{P}$ denote a random permutation. Our next step is to switch from the component being a random *permutation* to a random *function*. This follows in the same way as we proved the PRF-PRP Switching Lemma (Lemma 11.2). Suppose we replace $\mathcal{P}$ by a random function $\mathcal{F}$ in the CBC game and we let E denote the event, during the game $\mathsf{CBC}[\mathcal{F}]$, that the adversary makes two

calls to $\mathcal{F}$ which result in the same output value. We have

$$\begin{aligned}\Big|\Pr[A \text{ wins } \mathsf{CBC}[\mathcal{P}]] - \Pr[A \text{ wins } \mathsf{CBC}[\mathcal{F}]]\Big| &= \Big|\Pr[A \text{ wins } \mathsf{CBC}[\mathcal{P}]] \\ &\qquad - \Pr[A \text{ wins } \mathsf{CBC}[\mathcal{F}] \wedge \neg E] \\ &\qquad - \Pr[A \text{ wins } \mathsf{CBC}[\mathcal{F}] \wedge E]\Big| \\ &= \Big|\Pr[A \text{ wins } \mathsf{CBC}[\mathcal{P}]] - \Pr[A \text{ wins } \mathsf{CBC}[\mathcal{P}]] \\ &\qquad - \Pr[A \text{ wins } \mathsf{CBC}[\mathcal{F}] \mid E] \cdot \Pr[E]\Big| \\ &\leq \Pr[E] \leq \frac{T^2}{2^{n+1}}, \end{aligned} \tag{20}$$

since if E does not happen the two games are identical from the point of view of the adversary, and by the birthday bound $\Pr[E] \leq \frac{T^2}{2^{n+1}}$.

Our final task is to bound the probability of A winning the CBC game when the underlying "block cipher" is a random function. First let us consider how the challenger works in the game $\mathsf{CBC}[\mathcal{F}]$. When the adversary makes an $\mathcal{O}_{\mathsf{LR}}$ or $\mathcal{O}_{e_k}$ call, the challenger answers the query by calling the random function. As we are dealing with a random function, and not a random permutation, the challenger can select the output value of $\mathcal{F}$ *independently* from the codomain; i.e. it does not need to adjust the output values depending on the previous values. This last point will make our analysis simpler, and is why we switched to the PRF game from the PRP game.

Now notice that the adversary does not control the inputs to the random function at any stage in the game, so the only way he can find any information is by creating an input collision, i.e. two calls the challenger makes to the random function are on the same input values[3].

We thus let M_j denote the event that the adversary makes an input collision happen within the first j calls, and note that if M_T does not happen then the adversary's probability of winning is $1/2$, i.e. the best he can do is guess. We have

$$\begin{aligned}\Pr[A \text{ wins } \mathsf{CBC}[\mathcal{F}]] &= \Pr[A \text{ wins } \mathsf{CBC}[\mathcal{F}] \mid M_T] \cdot \Pr[M_T] \\ &\qquad + \Pr[A \text{ wins } \mathsf{CBC}[\mathcal{F}] \mid \neg M_T] \cdot \Pr[\neg M_T] \\ &\leq \Pr[M_T] + \Pr[A \text{ wins } \mathsf{CBC}[\mathcal{F}] \mid \neg M_T] \\ &\leq \Pr[M_T] + \frac{1}{2} \end{aligned} \tag{21}$$

So we are left with estimating $\Pr[M_T]$. Clearly, we have $\Pr[M_1] = 0$, and for all $j \geq 1$ we have

$$\Pr[M_j] \leq \Pr[M_{j-1}] + \Pr[M_j \mid \neg M_{j-1}],$$

from which it follows that

$$\Pr[M_j] \leq \frac{(j-1) \cdot (q_e + 1) \cdot \ell}{2^n}.$$

From which follows $\Pr[M_T] \leq T^2/2^n$.

[3]This is why CBC Mode is not secure in the nonce-based setting as in this setting the adversary controls the first input block, by selecting the first block of the message and the IV.

Summing up we have

$$\begin{aligned}
\mathrm{Adv}^{\mathsf{IND\text{-}CPA}}_{\mathsf{CBC}[e_k]}(A; q_e) &= 2 \cdot \left| \Pr[A \text{ wins } \mathsf{CBC}[e_k]] - \frac{1}{2} \right| \\
&= 2 \cdot \Big| \Pr[A \text{ wins } \mathsf{CBC}[e_k]] - \frac{1}{2} \\
&\qquad + (\Pr[A \text{ wins } \mathsf{CBC}[\mathcal{P}]] - \Pr[A \text{ wins } \mathsf{CBC}[\mathcal{P}]]) \Big| && \text{adding in zero} \\
&\leq 2 \cdot \left| \Pr[A \text{ wins } \mathsf{CBC}[e_k]] - \Pr[A \text{ wins } \mathsf{CBC}[\mathcal{P}]] \right| \\
&\qquad + 2 \cdot \left| \Pr[A \text{ wins } \mathsf{CBC}[\mathcal{P}]] - \frac{1}{2} \right| && \text{triangle inequality} \\
&= \mathrm{Adv}^{\mathsf{PRP}}_{e_k}(B) + 2 \cdot \left| \Pr[A \text{ wins } \mathsf{CBC}[\mathcal{P}]] - \frac{1}{2} \right| && \text{by equation (19)} \\
&= \mathrm{Adv}^{\mathsf{PRP}}_{e_k}(B) + 2 \cdot \Big| \Pr[A \text{ wins } \mathsf{CBC}[\mathcal{P}]] - \frac{1}{2} \\
&\qquad + (\Pr[A \text{ wins } \mathsf{CBC}[\mathcal{F}]] - \Pr[A \text{ wins } \mathsf{CBC}[\mathcal{F}]]) \Big| && \text{adding in zero} \\
&\leq \mathrm{Adv}^{\mathsf{PRP}}_{e_k}(B) \\
&\qquad + 2 \cdot \left| \Pr[A \text{ wins } \mathsf{CBC}[\mathcal{P}]] - \Pr[A \text{ wins } \mathsf{CBC}[\mathcal{F}]] \right| \\
&\qquad + 2 \cdot \left| \Pr[A \text{ wins } \mathsf{CBC}[\mathcal{F}]] - \frac{1}{2} \right| && \text{triangle inequality} \\
&\leq \mathrm{Adv}^{\mathsf{PRP}}_{e_k}(B) + \frac{T^2}{2^n} + 2 \cdot \left| \Pr[A \text{ wins } \mathsf{CBC}[\mathcal{F}]] - \frac{1}{2} \right| && \text{by equation (20)} \\
&\leq \mathrm{Adv}^{\mathsf{PRP}}_{e_k}(B) + \frac{T^2}{2^n} + 2 \cdot \Pr[M_T] && \text{by equation (21)} \\
&\leq \mathrm{Adv}^{\mathsf{PRP}}_{e_k}(B) + \frac{3 \cdot T^2}{2^n}.
\end{aligned}$$

□

Let us examine what this means when we use the AES block cipher in CBC Mode. First the block length of AES is $n = 128$, and let us assume the key size is 128 as well. If we assume AES behaves as a PRP, then we expect that

$$\mathrm{Adv}^{\mathsf{PRP}}_{AES}(B) \leq \frac{1}{2^{128}}$$

for all adversaries B. We can now work out the advantage *for any* adversary A to break AES when used in CBC mode, in the sense of IND-CPA. We find

$$\mathrm{Adv}^{\mathsf{IND\text{-}CPA}}_{\mathsf{CBC}[AES]}(A; q_e) \leq \frac{1 + 3 \cdot T^2}{2^{128}}.$$

Thus even if the adversary makes 2^{30} calls to the underlying block cipher, the advantage will still be less than

$$\frac{1 + 3 \cdot 2^{60}}{2^{128}} \approx 2^{-66},$$

which is incredibly small. Thus as long as we restrict the usage of AES in CBC Mode with a random IV to encrypting around 2^{30} blocks per key we will have a secure cipher. Restricting the usage of a symmetric cipher per key is enabled by requiring a user to generate a new key every so often.

For all three variants CBC Mode is not OW-CCA secure and hence not IND-CCA secure, due to a similar attack to that in Theorem 13.3. The OW-CPA security of CBC Mode, for all three ways of picking the IV, follows from Theorem 13.4 due to the proof of the following theorem.

Theorem 13.7. *For any method of choosing the IV, CBC mode* is OW-CPA *secure assuming the underlying block cipher e_k acts like a pseudo-random permutation. In particular let A denote an adversary against CBC Mode which makes q_e queries to its encryption oracle (with each query being at most ℓ blocks in length), then there is an adversary B such that*

$$\mathrm{Adv}^{\mathsf{OW\text{-}CPA}}_{\mathsf{CBC}[e_k]}(A; q_e) = \mathrm{Adv}^{\mathsf{OW\text{-}CPA}}_{\mathsf{ECB}[e_k]}(B; q_e \cdot \ell).$$

PROOF. For uniform challenge messages the distribution (for a fixed number of blocks) of the challenges given to adversary A and adversary B are identical. Algorithm B works as follows. We let $c^* = c_1 \| c_2 \| \ldots$ denote the challenge ciphertext passed to adversary B; we pass this to adversary A who returns the CBC Mode decryption of c^*, with the IV being anything that the given method allows, we let $m'_1 \| m'_2 \| \ldots$ be the returned plaintext. When algorithm A makes an encryption oracle request; this is answered by calling algorithm B's encryption oracle a block at a time, and simulating CBC Mode. Then using the known IV used when passing c^* to A, algorithm B can decrypt c^* under ECB Mode to obtain $m_1 \| m_2 \| \ldots$, using the following equations:

$$m_1 = m'_1 \oplus IV,$$
$$m_i = m'_i \oplus c_i \text{ for } i > 1.$$

□

13.4.3. OFB Mode: Output Feedback Mode, or OFB Mode, enables a block cipher to be used as a stream cipher. We use the block cipher to create the keystream, n bits at a time, where n is the block size. Again we divide the plaintext into a series of blocks, each block being n-bits long:

$$m_1, \ldots, m_q.$$

Encryption is performed as follows; see Figure 13.11 for a graphical representation. First we set $Y_0 \leftarrow IV$, then for $i = 1, 2, \ldots, q$, we perform the following steps,

$$Y_i \leftarrow e_k(Y_{i-1}),$$
$$c_i \leftarrow m_i \oplus Y_i.$$

The output ciphertext is $IV \| c_1 \| c_2 \| \ldots$. Decipherment in OFB Mode is performed in a similar manner.

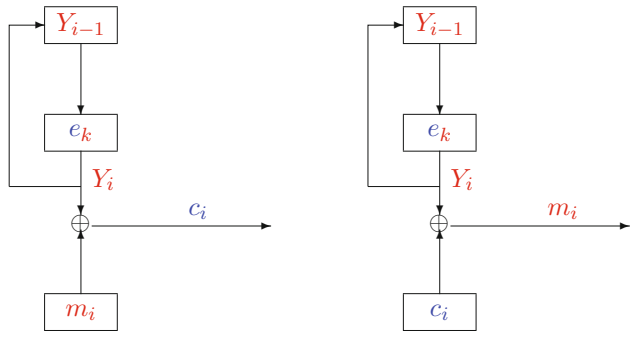

FIGURE 13.11. OFB encipherment and decipherment

We now turn to discussing security of OFB Mode. Clearly, when used with a fixed IV, OFB Mode is *not* IND-CPA secure; it turns out it is not OW-CPA secure either when used with a fixed IV as the next result demonstrates.

Theorem 13.8. *When used with a fixed IV, OFB Mode is not* OW-CPA *secure.*

PROOF. Call the encryption oracle on the plaintext consisting of all zero blocks. This reveals the "keystream", which can then be exclusive-or-ed with the challenge ciphertext to produce the required plaintext. □

Also on the negative side, for any method of selecting the IV, OFB Mode *is not* OW-CCA secure and hence is not IND-CCA secure; again the "attack" is exactly the same as in Theorem 13.3. On another negative side we have the following.

Theorem 13.9. *OFB Mode is not OW-CPA secure in the nonce-based setting, i.e. when the* IV *is a nonce.*

PROOF. The adversary first picks a nonce n and asks the encryption oracle for an encryption of $m' = m'_1 \| m'_2 \| \ldots$. The ciphertext $c_1 \| c_2 \| \ldots$ will be returned, from which the adversary can work out the keystream starting from IV n, and hence also from IV, $c_1 \oplus m'_1 = e_k(n)$. The adversary now asks for the challenge ciphertext c^* with nonce $c_1 \oplus m'_1 = e_k(n)$. Using the previously obtained keystream the adversary can recover the message encrypted by c^*. □

So we are left to consider what positive results hold. It also turns out that the proof of Theorem 13.6 can be applied to OFB Mode as well as CBC Mode, so we have that OFB Mode is IND-CPA secure, assuming the underlying block cipher is a secure pseudo-random permutation, when used with a random IV.

13.4.4. CFB Mode: The next mode we consider is called Cipher FeedBack Mode, or CFB Mode. This is very similar to OFB Mode in that we use the block cipher to produce a stream cipher. Recall that in OFB Mode the keystream was generated by encrypting IV and then iteratively encrypting the output from the previous encryption. In CFB Mode the keystream output is produced by the encryption of the ciphertext, as in Figure 13.12, by the following steps, upon setting $Y_0 \leftarrow IV$,

$$Z_i \leftarrow e_k(Y_{i-1}),$$
$$Y_i \leftarrow m_i \oplus Z_i.$$

We do not present the decryption steps, but leave this as an exercise for the reader. CFB mode has

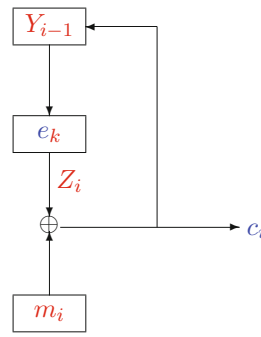

FIGURE 13.12. CFB encipherment

an interesting attack against it. Suppose the challenge ciphertext is $IV \| c_1 \| c_2 \| \ldots$, then if we query the *encryption* oracle with the zero plaintext block, but IV equal to c_1, we will obtain a ciphertext

c' such that $c_2 \oplus c' = m_2$. Continuing in this way we can recover all the plaintext blocks bar the first one. This clearly leads to an attack against IND security for nonce-based encryption. However it does not lead to an attack against the full one-wayness, as one cannot recover the first block.

Theorem 13.10. *CFB Mode* is not IND-CPA *secure when used as a nonce-based encryption scheme.*

Just as with OFB Mode, the proof of Theorem 13.6 can be applied, so we have that CFB Mode with a random IV is IND-CPA secure assuming the underlying block cipher is a secure pseudo-random permutation. We can also use the proof of Theorem 13.4, to show that CFB mode is OW-CPA secure when used as a nonce-based/random IV scheme.

13.4.5. CTR Mode: The next mode we consider is called Counter Mode, or CTR Mode. This combines many of the advantages of ECB Mode, but with none of the disadvantages. We first select a public IV, or counter, which is chosen differently for each message encrypted under the fixed key k. Then encryption proceeds for the ith block, by encrypting the value of $IV + i$ and then exclusive-or'ing this with the message block. In other words we have

$$c_i \leftarrow m_i \oplus e_k(IV \oplus \langle i \rangle_n),$$

where $\langle i \rangle_n$ is the n-bit representation of the number i. This is explained pictorially in Figure 13.13. An important property to preserve security of CTR Mode is that the counter input to *any* of the block cipher calls may not be reused in any subsequent encryption. In the case of a random IV chosen by the encryptor this will happen with overwhelming probability, assuming we limit the number of block cipher invocations with a given key. In the case of nonce-based encryption we simply have to ensure that if we encrypt a t-block message with nonce $IV = j$, then we never take a new nonce in the range $[j, \ldots, j + t - 1]$.

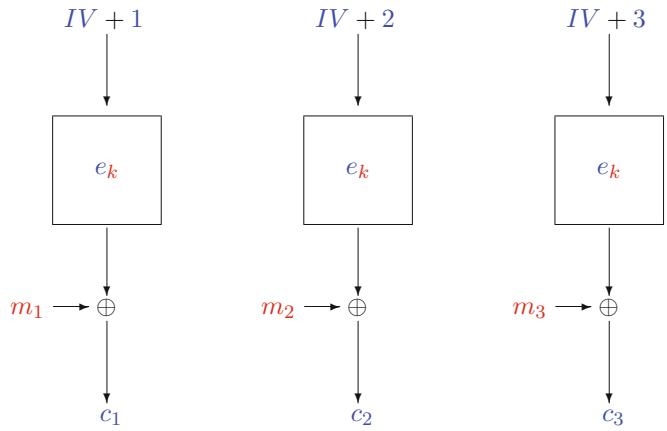

FIGURE 13.13. CTR encipherment

CTR Mode has a number of interesting properties. Firstly, since each block can be encrypted independently, much like in ECB Mode, we can process each block at the same time. Compare this to CBC Mode, OFB Mode or CFB Mode where we cannot start encrypting the second block until the first block has been encrypted. This means that encryption, and decryption, can be performed in parallel. Another performance advantage comes from the fact that we only ever apply the encryption operation of the underlying block cipher.

However, unlike ECB Mode, two equal blocks will not encrypt to the same ciphertext value. This is because each plaintext block is encrypted using a different input to the encryption function;

in some sense we are using the block cipher encryption of the different inputs to produce a stream cipher. Also unlike ECB Mode each ciphertext block corresponds to a precise position within the ciphertext, as one needs the position information to be able to decrypt it successfully.

In terms of security properties, the security proof for IND-CPA security is actually simpler than that used for CBC Mode. The reason is that with the above restrictions on the reuse of IVs we never have to worry about an input collision for the block cipher. Thus, assuming the total number of blocks encrypted with a fixed key is kept relatively low, we have the following.

Theorem 13.11. *CTR Mode is* IND-CPA *secure assuming the underlying block cipher* e_k *acts like a pseudo-random permutation. In particular let* A *denote an adversary against CTR Mode which makes* q_e *queries to its encryption oracle, and let all plaintext submitted to both the* LR *and encryption oracles be at most* ℓ *blocks in length. There is then an adversary* B *such that*

$$\mathrm{Adv}^{\mathsf{IND\text{-}CPA}}_{\mathsf{CTR}[e_k]}(A; q_e) \leq \mathrm{Adv}^{\mathsf{PRP}}_{e_k}(B) + \frac{T^2}{2^n},$$

where n *is the block size of the cipher* e_k *and* $T = (q_e + 1) \cdot \ell$.

With the above restrictions on the reuse of the nonce, we obtain an IND-CPA secure nonce-based encryption scheme with exactly the same advantage statement.

All is not totally positive however; the attack in Theorem 13.8 also applies to CTR mode and hence the scheme is not OW-CPA secure when used with a fixed IV. In addition we cannot achieve CCA security, again due to the attack presented in Theorem 13.3.

Mode	IV	OW-PASS	OW-CPA	OW-CCA
ECB Mode	-	OW-CPA $\Longrightarrow$ ✓	Thm 13.4 $\Longrightarrow$ ✓	Thm 13.3 $\Longrightarrow$ ✗
CBC Mode	fixed	IND-PASS $\Longrightarrow$ ✓	Thm 13.7 $\Longrightarrow$ ✓	Prf of Thm 13.3 $\Longrightarrow$ ✗
	nonce	IND-PASS $\Longrightarrow$ ✓	Thm 13.7 $\Longrightarrow$ ✓	Prf of Thm 13.3 $\Longrightarrow$ ✗
	random	IND-PASS $\Longrightarrow$ ✓	IND-CPA $\Longrightarrow$ ✓	Prf of Thm 13.3 $\Longrightarrow$ ✗
OFB Mode	fixed	IND-PASS $\Longrightarrow$ ✓	Thm 13.8 $\Longrightarrow$ ✗	Prf of Thm 13.3 $\Longrightarrow$ ✗
	nonce	IND-PASS $\Longrightarrow$ ✓	Thm 13.9 $\Longrightarrow$ ✗	Prf of Thm 13.3 $\Longrightarrow$ ✗
	random	IND-PASS $\Longrightarrow$ ✓	IND-CPA $\Longrightarrow$ ✓	Prf of Thm 13.3 $\Longrightarrow$ ✗
CFB Mode	fixed	IND-PASS $\Longrightarrow$ ✓	(nonce OW-CPA) $\Longrightarrow$ ✓	Prf of Thm 13.3 $\Longrightarrow$ ✗
	nonce	IND-PASS $\Longrightarrow$ ✓	Prf of Thm 13.4 $\Longrightarrow$ ✓	Prf of Thm 13.3 $\Longrightarrow$ ✗
	random	IND-PASS $\Longrightarrow$ ✓	IND-CPA $\Longrightarrow$ ✓	Prf of Thm 13.3 $\Longrightarrow$ ✗
CTR Mode	fixed	IND-PASS $\Longrightarrow$ ✓	Prf of Thm 13.8 $\Longrightarrow$ ✗	Prf of Thm 13.3 $\Longrightarrow$ ✗
	nonce	IND-PASS $\Longrightarrow$ ✓	IND-CPA $\Longrightarrow$ ✓	Prf of Thm 13.3 $\Longrightarrow$ ✗
	random	IND-PASS $\Longrightarrow$ ✓	IND-CPA $\Longrightarrow$ ✓	Prf of Thm 13.3 $\Longrightarrow$ ✗

TABLE 13.1. One-way security properties of the five basic modes of operation

We summarize our results on modes of operation in Tables 13.1 and 13.2. Against each tick or cross we give the theorem number which presents this result, or the proof of the theorem which can be modified to give the result. For nonce-based CTR mode we assume the convention with respect to nonces described earlier. To derive most of the tables we note the implications given in Figure 11.18; e.g. (¬OW-XXX) $\Longrightarrow$ (¬IND-XXX) and (equivalently) (IND-XXX) $\Longrightarrow$ (OW-XXX), also (IND-CPA) $\Longrightarrow$ (IND-PASS). We also note that in the passive security setting there is no difference between the nonce-based and fixed IV variants, and if a scheme is IND-PASS secure in the random-IV setting, then it is also IND-PASS secure in the nonce-based setting.

Mode	IV	IND-PASS	IND-CPA	IND-CCA
ECB Mode	-	Thm 13.2 $\implies$ ✗	($\neg$ IND-PASS) $\implies$ ✗	($\neg$ IND-PASS) $\implies$ ✗
CBC Mode	fixed	(nonce IND-PASS) $\implies$ ✓	Trivially ✗	Trivially ✗
	nonce	(random IND-PASS) $\implies$ ✓	Thm 13.5 $\implies$ ✗	($\neg$ OW-CCA) $\implies$ ✗
	random	IND-CPA $\implies$ ✓	Thm 13.6 $\implies$ ✓	($\neg$ OW-CCA) $\implies$ ✗
OFB Mode	fixed	(nonce IND-PASS) $\implies$ ✓	Trivially ✗	Trivially ✗
	nonce	(random IND-PASS) $\implies$ ✓	($\neg$ OW-CPA) $\implies$ ✗	($\neg$ OW-CCA) $\implies$ ✗
	random	IND-CPA $\implies$ ✓	Prf of Thm 13.6 $\implies$ ✓	($\neg$ OW-CCA) $\implies$ ✗
CFB Mode	fixed	(nonce IND-PASS) $\implies$ ✓	Trivially ✗	Trivially ✗
	nonce	(random IND-PASS) $\implies$ ✓	Thm 13.10 $\implies$ ✗	($\neg$ OW-CCA) $\implies$ ✗
	random	IND-CPA $\implies$ ✓	Prf of Thm 13.6 $\implies$ ✓	($\neg$ OW-CCA) $\implies$ ✗
CTR Mode	fixed	(nonce IND-PASS) $\implies$ ✓	Trivially ✗	Trivially ✗
	nonce	IND-CPA $\implies$ ✓	Thm 13.11 $\implies$ ✓	($\neg$ OW-CCA) $\implies$ ✗
	random	IND-CPA $\implies$ ✓	Thm 13.11 $\implies$ ✓	($\neg$ OW-CCA) $\implies$ ✗

TABLE 13.2. Indistinguishability security properties of the five basic modes of operation

13.5. Obtaining Chosen Ciphertext Security

All the prior modes of operation did not provide security against chosen ciphertext attacks. To do this one needs more advanced modes of operation, called authenticated encryption modes. There are a number of modes which provide this property for symmetric encryption based on block ciphers, for example CCM Mode, GCM Mode and OCB Mode. There is little room in this book to cover these modes, however we will present a simple technique to obtain an IND-CCA secure symmetric cipher from one of the previous IND-CPA secure modes of operation, called *Encrypt-then-MAC*.

Let (E_k, D_k) denote an IND-CPA symmetric encryption scheme, say CBC Mode or CTR Mode instantiated with AES. Let $\mathbb{K}_1$ denote the key space of the encryption scheme. The problem with the previous chosen ciphertext attacks was that an adversary could create a new ciphertext without needing to know the underlying key. So the decryption oracle would decrypt any old garbage which the adversary threw at it. The trick to obtaining CCA secure schemes is to ensure that the decryption oracle rejects almost all of the ciphertexts that the adversary creates; in fact we hope it rejects all bar the ones validly produced by an encryption oracle.

13.5.1. Encrypt-then-MAC: The standard *generic* method of adding this form of integrity protection to the ciphertext is to append a message authentication code to the ciphertext. We let $(\mathsf{Mac}_k, \mathsf{Verify}_k)$ denote a message authentication code with key space $\mathbb{K}_2$. In the next chapter we will see how such codes can be constructed, but for now just assume we can create one which satisfies the security definition of strong existential unforgeability under chosen message attacks, from Chapter 11. We then construct our CCA secure symmetric encryption scheme, called Encrypt-then-MAC (ETM), as follows:

KeyGen(): Sample $k_1 \leftarrow \mathbb{K}_1$ and $k_2 \leftarrow \mathbb{K}_2$, return $k = (k_1, k_2)$.
$\mathrm{Enc}_k(m)$: Compute the ciphertext $c_1 \leftarrow E_{k_1}(m)$, then form a MAC on the ciphertext from $c_2 \leftarrow \mathsf{Mac}_{k_2}(c_1)$. Return the ciphertext $c = (c_1, c_2)$.
$\mathrm{Dec}_k(c)$: Verify the MAC and decrypt the ciphertext: $v \leftarrow \mathsf{Verify}_{k_2}(c_2, c_1)$, and $m \leftarrow D_{k_1}(c_1)$. If $v = \texttt{valid}$ then return m, else return $\perp$.

Note that the message authentication code is applied to the ciphertext *and not* the plaintext; this is very important to maintain security. Also note that, when decrypting, we decrypt the first ciphertext component c_1, irrespective of whether the MAC in c_2 verifies or not; this is to avoid subtle timing attacks. We now show that this construction is IND-CCA secure, assuming the underlying encryption scheme and message authentication code are suitably secure.

Theorem 13.12. *Let A denote an adversary against the* ETM *scheme constructed from the encryption scheme $\Pi_1 = (\mathsf{Enc}, \mathsf{Dec})$ and the message authentication code $\Pi_2 = (M, V)$, then there are two adversaries B_1 and B_2 such that*

$$\mathrm{Adv}^{\mathsf{IND\text{-}CCA}}_{\mathsf{ETM}}(A) \leq \mathrm{Adv}^{\mathsf{IND\text{-}CPA}}_{\Pi_1}(B_1) + \mathrm{Adv}^{\mathsf{sEUF\text{-}CMA}}_{\Pi_2}(B_2).$$

Proof. Let E denote the event that the adversary produces a ciphertext, which is not the output of a call to the encryption oracle, which when passed to the decryption oracle results in the output of something other than $\perp$. We have

$$\Pr[A \text{ wins}] = \Pr[A \text{ wins} \wedge \neg E] \cdot \Pr[\neg E] + \Pr[A \text{ wins} \wedge E] \cdot \Pr[E] \quad \leq \Pr[A \text{ wins} \wedge \neg E] + \Pr[E].$$

Now $\Pr[A \text{ wins} \wedge \neg E]$ is the same probability as running A in a IND-CPA attack, since if E does not happen then A does not learn anything from its decryption queries, and all decryption oracle queries can be answered by either returning $\perp$, or remembering what was queried to the encryption oracle. Hence in this case we can take A to be the B_1 in the theorem, and just ignore any decryption queries which the adversary A makes which do not correspond to the adversary's outputs from the encryption oracle.

If event E happens then the adversary A must have created a ciphertext, which is different from the challenge ciphertext, whose MAC verifies. In such a situation we can create an adversary B_2 which breaks the message authentication code as follows. We create B_2 by using A as a subroutine.

- Generate $k_1 \leftarrow \mathbb{K}_1$.
- Call A.
- When A makes a query to the encryption oracle, use the key k_1 to create a first ciphertext component, and then use B_2's oracle $\mathcal{O}_{\mathsf{Mac}_k}$ to create the second ciphertext component.
- When A makes a query to the $\mathcal{O}_{\mathsf{LR}}$ oracle, pick the random bit b and proceed as for the $\mathcal{O}_{e_k}$ oracle.
- When A makes a query to its decryption oracle we first check whether the ciphertext verifies (using the verification oracle provided to B_2). If it does, and the ciphertext is not the output of $\mathcal{O}_{e_k}$ then stop and return the input as a MAC forgery. Otherwise proceed as in the real game.

Let (c_1, c_2) denote the output of adversary B_2. We note that one of c_1 and c_2 must be different from the output of $\mathcal{O}_{e_k}$ (by definition of B_2) and $\mathcal{O}_{\mathsf{LR}}$ (by the rules A is following). If the difference is c_1 then c_2 is a valid MAC on a message which has not been queried to B_2's $\mathcal{O}_{\mathsf{Mac}_k}$ oracle. Whereas if c_1 is the same as a previous output from $\mathcal{O}_{e_k}$ or $\mathcal{O}_{\mathbb{K}}$, and c_2 is different, then B_2 has managed to produce a *strong* forgery. Thus in either case a MAC forgery has been created and the result follows. So $\Pr[E]$ is bound by the advantage of B_2 in winning the forgery game for the MAC function. □

Note that the same construction can be used to construct a CCA-secure DEM, i.e. a data encapsulation mechanism. Recall that this is a symmetric encryption scheme for which only one message is ever created with a given key, but for which the adversary has a decryption oracle. Using the same technique as in the proof above we can prove the following.

Theorem 13.13. *Let A denote an adversary against the* ETM *scheme constructed from the encryption scheme $\Pi_1 = (\mathsf{Enc}, \mathsf{Dec})$ and the message authentication code $\Pi_2 = (M, V)$, then there are two adversaries B_1 and B_2 such that*

$$\mathrm{Adv}^{\mathsf{ot\text{-}IND\text{-}CCA}}_{\mathsf{ETM}}(A) \leq \mathrm{Adv}^{\mathsf{IND\text{-}PASS}}_{\Pi_1}(B_1) + \mathrm{Adv}^{\mathsf{ot\text{-}sEUF\text{-}CMA}}_{\Pi_2}(B_2).$$

Note that we only require a passively secure encryption scheme and a one-time secure message authentication code. Hence we could, to construct a DEM, use CBC Mode with a fixed IV. This means that the ciphertext for a DEM can be one block shorter than for a general encryption scheme, as we no longer need to transmit the IV. The use of DEMs will become clearer when we discuss hybrid encryption in a later chapter.

13.5.2. Encrypt-and-MAC: We stressed above that it is important that one authenticates the ciphertext and not the message. One popular method in the past for trying to produce a CCA secure symmetric encryption scheme was to use a method called *Encrypt-and-MAC*. Here one applies a MAC to the plaintext and then appends this MAC to the ciphertext. Thus we have

KeyGen(): Sample $k_1 \leftarrow \mathbb{K}_1$ and $k_2 \leftarrow \mathbb{K}_2$, return $k = (k_1, k_2)$.
$\mathrm{Enc}_k(m)$: Compute the ciphertext $c_1 \leftarrow E_{k_1}(m)$, then form a MAC on the plaintext from $c_2 \leftarrow \mathsf{Mac}_{k_2}(m)$. Return the ciphertext $c = (c_1, c_2)$.
$\mathrm{Dec}_k(c)$: Decrypt the ciphertext and then verify the MAC: $m \leftarrow D_{k_1}(c_1)$. $v \leftarrow \mathsf{Verify}_{k_2}(c_2, m)$, If $v = \texttt{valid}$ then return m, else return $\perp$.

The problem is that this method is not *generically* secure. By this we mean that its security depends on the precise choice of the IND-CPA encryption scheme and MAC which one selects. In particular the scheme is *not* secure when instantiated with any of the standard MAC functions we will discuss in Chapter 14, as the following result shows.

Theorem 13.14. *Encrypt-and-MAC is* not IND-CPA *secure when instantiated with an* IND-CPA *encryption scheme and a* deterministic *MAC.*

Proof. The attack is to pick two plaintext messages m_0 and m_1 of the same length and pass these to the $\mathcal{O}_{\mathsf{LR}}$ oracle to obtain a challenge ciphertext $c^* = (c_1^*, c_2^*)$. Now the adversary passes m_0 to its encryption oracle to obtain a new ciphertext $c = (c_1, c_2)$. As the MAC is deterministic, if $c_2 = c_2^*$ then the hidden bit is zero, and if not the hidden bit is one. □

Notice that Encrypt-and-MAC is less secure (in the sense of the IND-CPA notion) than the original encryption algorithm without the MAC!

13.5.3. MAC-then-Encrypt: Another method which has been used in protocols in the past, but which again is not secure in general is called *MAC-then-Encrypt*. Here we MAC the plaintext and then encrypt the plaintext and the MAC together. Thus we have

KeyGen(): Sample $k_1 \leftarrow \mathbb{K}_1$ and $k_2 \leftarrow \mathbb{K}_2$, return $k = (k_1, k_2)$.
$\mathrm{Enc}_k(m)$: Form a MAC on the plaintext from $t \leftarrow \mathsf{Mac}_{k_2}(m)$. Compute the ciphertext $c \leftarrow E_{k_1}(m \| t)$.
$\mathrm{Dec}_k(c)$: Decrypt the ciphertext and then verify the MAC: $m \| t \leftarrow D_{k_1}(c_1)$. $v \leftarrow \mathsf{Verify}_{k_2}(t, m)$. If $v = \texttt{valid}$ then return m, else return $\perp$.

Again this method is not *generically* secure, since we have the following.

Theorem 13.15. *Encryption via the MAC-then-Encrypt method* may not *be* IND-CPA *secure when instantiated with an* IND-CPA *encryption scheme and* EUF-CMA *secure MAC.*

Proof. We present an, admittedly contrived, example although more complex real-life examples do exist. Take an IND-CPA encryption scheme (E_k, D_k) and modify it to form the following encryption scheme, which we shall denote by (E'_k, D'_k). To encrypt one performs outputs $E_k(m) \| 0$, i.e. one adds a zero bit onto the ciphertext output by E_k. To decrypt one ignores the zero bit at the end,

one does not even bother to check it is zero, and decrypts the main component using D_k. It is clear that, since (E_k, D_k) is IND-CPA secure, so is (E'_k, D'_k).

Now form the MAC-then-Encrypt cipher using (E'_k, D'_k) and any EUF-CMA secure MAC. Consider a challenge ciphertext c^*; this will end in zero. Now we can change this zero to a one, to create a new ciphertext c, which is different from c^*. Now since the decryption algorithm does not check the last bit, the decryption oracle will decrypt c to reveal the message underlying c^* and the associated MAC will verify. Thus this MAC-then-Encrypt scheme is not even OW-CCA secure. □

Chapter Summary

- Probably the most famous block cipher is DES, which is itself based on a general design called a Feistel cipher.
- A comparatively recent block cipher is the AES cipher, called Rijndael.
- Both DES and AES obtain their security by repeated application of simple rounds consisting of substitution, permutation and key addition.
- To use a block cipher one needs to also specify a mode of operation. The simplest mode is ECB mode, which has a number of problems associated with it. Hence, it is common to use a more advanced mode such as CBC or CTR mode.
- Some block cipher modes, such as CFB, OFB and CTR modes, allow the block cipher to be used as a stream cipher.
- To obtain an IND-CCA secure scheme one can use Encrypt-then-MAC.

Further Reading

The Rijndael algorithm, the AES process and a detailed discussion of attacks on block ciphers and Rijndael in particular can be found in the book by Daemen and Rijmen. Stinson's book is the best book to explain differential cryptanalysis for students. For a discussion of how to combine encryption functions and MAC functions to obtain IND-CCA secure encryption see the paper by Bellare and Namprempre.

M. Bellare and C. Namprempre. *Authenticated encryption: Relations among notions and analysis of the generic composition paradigm.* Advances in Cryptology – Asiacrypt 2000, LNCS 1976, 531–545, Springer, 2000.

J. Daemen and V. Rijmen. *The Design of Rijndael: AES – The Advanced Encryption Standard.* Springer, 2002.

D. Stinson. *Cryptography Theory and Practice.* Third Edition. CRC Press, 2005.