Chapter 5
More About Block Ciphers

A block cipher is much more than just an encryption algorithm. It can be used as a versatile building block with which a diverse set of cryptographic mechanisms can be realized. For instance, we can use them for building different types of block-based encryption schemes, and we can even use block ciphers for realizing stream ciphers. The different ways of encryption are called *modes of operation* and are discussed in this chapter. Block ciphers can also be used for constructing hash functions, message authentication codes which are also knowns as MACs, or key establishment protocols, all of which will be described in later chapters. There are also other uses for block ciphers, e.g., as pseudo-random generators. In addition to modes of operation, this chapter also discusses two very useful techniques for increasing the security of block ciphers, namely key whitening and multiple encryption.

In this chapter you will learn

- the most important modes of operation for block ciphers in practice
- security pitfalls when using modes of operations
- the principles of key whitening
- why double encryption is not a good idea, and the meet-in-the-middle attack
- triple encryption

Christof Paar et al., *Understanding Cryptography*

5.1 Encryption with Block Ciphers: Modes of Operation

In the previous chapters we introduced how DES, 3DES and AES encrypt a block of data. Of course, in practice one wants typically to encrypt more than one single 8-byte or 16-byte block of plaintext, e.g., when encrypting an e-mail or a computer file. There are several ways of encrypting long plaintexts with a block cipher. We introduce several popular modes of operation in this chapter, including

- Electronic Code Book mode (ECB),
- Cipher Block Chaining mode (CBC),
- Cipher Feedback mode (CFB),
- Output Feedback mode (OFB),
- Counter mode (CTR).

The latter three modes use the block cipher as a building block for a stream cipher.

All of the five modes have one goal: They encrypt data and thus provide confidentiality for a message sent from Alice to Bob. In practice, we often not only want to keep data confidential, but Bob also wants to know whether the message is really coming from Alice. This is called authentication and the Galois Counter mode (GCM), which we will also introduce, is a mode of operation that lets the receiver (Bob) determine whether the message was really sent by the person he shares a key with (Alice). Moreover, authentication also allows Bob to detect whether the ciphertext was altered during transmission. More on authentication is found in Chap. 10.

The ECB and CFB modes require that the length of the plaintext be an exact multiple of the block size of the cipher used, e.g., a multiple of 16 bytes in the case of AES. If the plaintext does not have this length, it must be padded. There are several ways of doing this padding in practice. One possible padding method is to append a single "1" bit to the plaintext and then to append as many "0" bits as necessary to reach a multiple of the block length. Should the plaintext be an exact multiple of the block length, an extra block consisting only of padding bits is appended.

5.1.1 Electronic Codebook Mode (ECB)

The *Electronic Code Book (ECB)* mode is the most straightforward way of encrypting a message. In the following, let $e_k(x_i)$ denote the encryption of plaintext block x_i with key k using some arbitrary block cipher. Let $e_k^{-1}(y_i)$ denote the decryption of ciphertext block y_i with key k. Let us assume that the block cipher encrypts (decrypts) blocks of size b bits. Messages which exceed b bits are partitioned into b-bit blocks. If the length of the message is not a multiple of b bits, it must be padded to a multiple of b bits prior to encryption. As shown in Fig. 5.1, in ECB mode each block is encrypted separately. The block cipher can, for instance, be AES or 3DES.

Encryption and decryption in the ECB mode is formally described as follows:

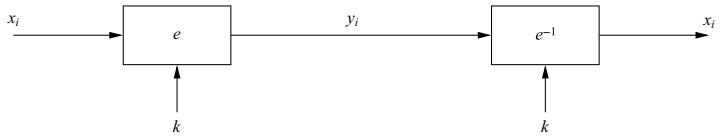

Fig. 5.1 Encryption and decryption in ECB mode

Definition 5.1.1 Electronic Codebook Mode (ECB)
Let $e()$ be a block cipher of block size b, and let x_i and y_i be bit strings of length b.
***Encryption**: $y_i = e_k(x_i), \quad i \geq 1$*
***Decryption**: $x_i = e_k^{-1}(y_i) = e_k^{-1}(e_k(x_i)), \quad i \geq 1$*

It is straightforward to verify the correctness of the ECB mode:

$$e_k^{-1}(y_i) = e_k^{-1}(e_k(x_i)) = x_i.$$

The ECB mode has advantages. Block synchronization between the encryption and decryption parties Alice and Bob is not necessary, i.e., if the receiver does not receive all encrypted blocks due to transmission problems, it is still possible to decrypt the received blocks. Similarly, bit errors, e.g., caused by noisy transmission lines, only affect the corresponding block but not succeeding blocks. Also, block ciphers operating in ECB mode can be parallelized, e.g., one encryption unit encrypts (or decrypts) block 1, the next one block 2, and so on. This is an advantage for high-speed implementations, but many other modes such as the CFB do not allow parallelization.

However, as is often the case in cryptography, there are some unexpected weaknesses associated with the ECB mode which we will discuss in the following. The main problem of the ECB mode is that it encrypts highly deterministically. This means that identical plaintext blocks result in identical ciphertext blocks, as long as the key does not change. The ECB mode can be viewed as a gigantic code book — hence the mode's name — which maps every input to a certain output. Of course, if the key is changed the entire code book changes, but as long as the key is static the book is fixed. This has several undesirable consequences. First, an attacker recognizes if the same message has been sent twice simply by looking at the ciphertext. Deducing information from the ciphertext in this way is called *traffic analysis*. For instance, if there is a fixed header that always precedes a message, the header always results in the same ciphertext. From this, an attacker can, for instance, learn when a new message has been sent. Second, plaintext blocks are encrypted independently of previous blocks. If an attacker reorders the ciphertext blocks, this might result in valid plaintext and the reordering might not be detected. We demonstrate two simple attacks which exploit these weaknesses of the ECB mode.

The ECB mode is susceptible to *substitution attacks*, because once a particular plaintext to ciphertext block mapping $x_i \rightarrow y_i$ is known, a sequence of ciphertext

blocks can easily be manipulated. We demonstrate how a substitution attack could work in the real world. Imagine the following example of an electronic wire transfer betweens banks.

Example 5.1. Substitution attack against electronic bank transfer
Let's assume a protocol for wire transfers between banks (Fig. 5.2). There are five fields which specify a transfer: the sending bank's ID and account number, the receiving bank's ID and account number, and the amount. We assume now (and this is a major simplification) that each of the fields has exactly the size of the block cipher width, e.g., 16 bytes in the case of AES. Furthermore, the encryption key between the two banks does not change too frequently. Due to the nature of the ECB, an attacker can exploit the deterministic nature of this mode of operation by simple substitution of the blocks. The attack details are as follows:

Block #	1	2	3	4	5
	Sending Bank A	Sending Account #	Receiving Bank B	Receiving Account #	Amount $

Fig. 5.2 Example for a substitution attack against ECB encryption

1. The attacker, Oscar, opens one account at bank A and one at bank B.
2. Oscar taps the encrypted line of the banking communication network.
3. He sends $1.00 transfers from his account at bank A to his account at bank B repeatedly. He observes the ciphertexts going through the communication network. Even though he cannot decipher the random-looking ciphertext blocks, he can check for ciphertext blocks that repeat. After a while he can recognize the five blocks of his own transfer. He now stores blocks 1, 3 and 4 of these transfers. These are the encrypted versions of the ID numbers of both banks as well as the encrypted version of his account at bank B.
4. Recall that the two banks do not change the key too frequently. This means that the same key is used for several other transfers between bank A and B. By comparing blocks 1 and 3 of *all* subsequent messages with the ones he has stored, Oscar recognizes all transfers that are made from some account at bank A to some account at bank B. He now simply replaces block 4 — which contains the receiving account number — with the block 4 that he stored before. This block contains Oscar's account number in encrypted form. As a consequence, *all transfers* from some account of bank A to some account of bank B are redirected to go into Oscar's B account! Note that bank B now has means of detecting that the block 4 has been replaced in some of the transfers it receives.
5. Withdraw money from bank B quickly and fly to a country that has a relaxed attitude about the extradition of white-collar criminals.

◇

What's interesting about this attack is that it works completely without attacking the block cipher itself. So even if we would use AES with a 256-bit key and if

we would encrypt each block, say, 1000 times, this would not prevent the attack. It should be stressed, however, that this is not an attack that breaks the block cipher itself. Messages that are unknown to Oscar still remain confidential. He simply replaced parts of the ciphertext with some other (previous) ciphertexts. This is called a violation of the *integrity* of the message. There are available techniques for preserving the integrity of a message, namely message authentication codes (MACs) and digital signatures. Both are widely used in practice to prevent such an attack, and are introduced in Chaps. 10 and 12. Also, the Galois Counter mode, which is described below, is an encryption mode with a built-in integrity check. Note that this attack only works if the key between bank A and B is not changed too frequently. This is another reason why frequent key freshness is a good idea.

We now look at another problem posed by the ECB mode.

Example 5.2. Encryption of bitmaps in ECB mode
Figure 5.3 clearly shows a major disadvantage of the ECB mode: Identical plaintexts are mapped to identical ciphertexts. In case of a simple bitmap, the information (text in the picture) can still be read out from the encrypted picture even though we used AES with a 256-bit key for encryption. This is because the background consists of only a few different plaintext blocks which yields a fairly uniformly looking background in the ciphertext. On the other hand, all plaintext blocks which contain part of the letters result in random-looking ciphertexts. These random-looking ciphertexts are clearly distinguishable from the uniform background by the human eye.

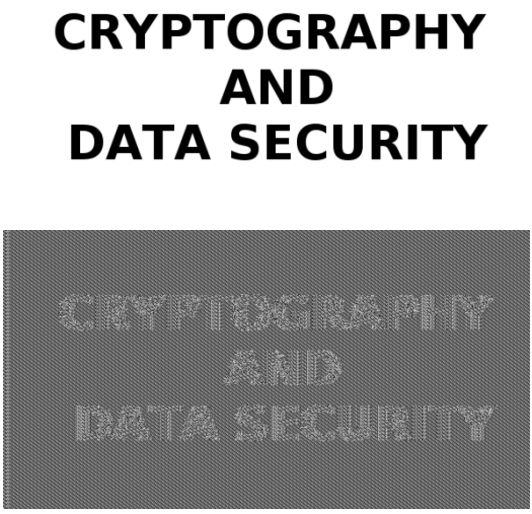

Fig. 5.3 Image and encrypted image using AES with 256-bit key in ECB mode

◇

This weakness is similar to the attack of the substitution cipher that was introduced in the first example. In both cases, statistical properties in the plaintext are preserved in the ciphertext. Note that unlike an attack against the substitution cipher or the above banking transfer attack, an attacker does not have to do anything in the case here. The human eye automatically makes use of the statistical information.

Both attacks above were examples of the weakness of a deterministic encryption scheme. Thus, it is usually preferable that different ciphertexts are produced every time we encrypt the same plaintext. This behavior is called *probabilistic encryption*. This can be achieved by introducing some form of randomness, typically in form of an initialization vector (IV). The following modes of operation all encrypt probabilistically by means of an IV.

5.1.2 Cipher Block Chaining Mode (CBC)

There are two main ideas behind the *Cipher Block Chaining (CBC)* mode. First, the encryption of all blocks are "chained together" such that ciphertext y_i depends not only on block x_i but on all previous plaintext blocks as well. Second, the encryption is randomized by using an initialization vector (IV). Here are the details of the CBC mode.

The ciphertext y_i, which is the result of the encryption of plaintext block x_i, is fed back to the cipher input and XORed with the succeeding plaintext block x_{i+1}. This XOR sum is then encrypted, yielding the next ciphertext y_{i+1}, which can then be used for encrypting x_{i+2}, and so on. This process is shown on the left-hand side of Fig. 5.4. For the first plaintext block x_1 there is no previous ciphertext. For this an IV is added to the first plaintext, which also allows us to make each CBC encryption nondeterministic. Note that the first ciphertext y_1 depends on plaintext x_1 (and the IV). The second ciphertext depends on the IV, x_1 and x_2. The third ciphertext y_3 depends on the IV and x_1, x_2, x_3, and so on. The last ciphertext is a function of all plaintext blocks and the IV.

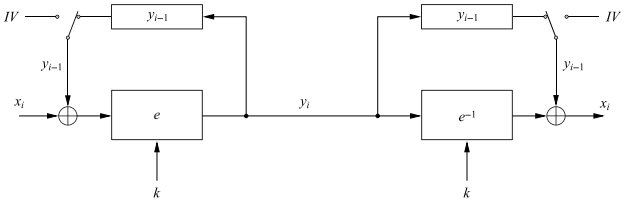

Fig. 5.4 Encryption and decryption in CBC mode

When decrypting a ciphertext block y_i in CBC mode, we have to reverse the two operations we have done on the encryption side. First, we have to reverse the block cipher encryption by applying the decryption function $e^{-1}()$. After this we have to

undo the XOR operation by again XORing the correct ciphertext block. This can be expressed for general blocks y_i as $e_k^{-1}(y_i) = x_i \oplus y_{i-1}$. The right-hand side of Fig. 5.4 shows this process. Again, if the first ciphertext block y_1 is decrypted, the result must be XORed with the initialization vector IV to determine the plaintext block x_1, i.e., $x_1 = IV \oplus e_k^{-1}(y_1)$. The entire process of encryption and decryption can be described as:

Definition 5.1.2 Cipher block chaining mode (CBC)
Let $e()$ be a block cipher of block size b; let x_i and y_i be bit strings of length b; and IV be a nonce of length b.
Encryption (first block): $y_1 = e_k(x_1 \oplus IV)$
Encryption (general block): $y_i = e_k(x_i \oplus y_{i-1}), \quad i \geq 2$
Decryption (first block): $x_1 = e_k^{-1}(y_1) \oplus IV$
Decryption (general block): $x_i = e_k^{-1}(y_i) \oplus y_{i-1}, \quad i \geq 2$

We now verify the mode, i.e., we show that the decryption actually reverses the encryption. For the decryption of the first block y_1, we obtain:

$$d(y_1) = e_k^{-1}(y_1) \oplus IV = e_k^{-1}(e_k(x_1 \oplus IV)) \oplus IV = (x_1 \oplus IV) \oplus IV = x_1$$

For the decryption of all subsequent blocks y_i, $i \geq 2$, we obtain:

$$d(y_i) = e_k^{-1}(y_i) \oplus y_{i-1} = e_k^{-1}(e_k(x_i \oplus y_{i-1})) \oplus y_{i-1} = (x_i \oplus y_{i-1}) \oplus y_{i-1} = x_i$$

If we choose a new IV every time we encrypt, the CBC mode becomes a probabilistic encryption scheme. If we encrypt a string of blocks $x_1, \ldots, x_t$ once with a first IV and a second time with a different IV, the two resulting ciphertext sequences look completely unrelated to each other for an attacker. Note that we do *not* have to keep the IV secret. However, in most cases, we want the IV to be a nonce, i.e., a number used only once. There are many different ways of generating and agreeing on initialization values. In the simplest case, a randomly chosen number is transmitted in the clear between the two communication parties prior to the encrypted session. Alternatively it is a counter value that is known to Alice and Bob, and it is incremented every time a new session starts (which requires that the counter value must be stored between sessions). It could be derived from values such as Alice's and Bob's ID number, e.g., their IP addresses, together with the current time. Also, in order to strengthen any of these methods, we can take a value as described above and ECB-encrypt it once using the block cipher with the key known to Alice and Bob, and use the resulting ciphertext as the IV. There are some advanced attacks which also require that the IV is nonpredictable.

It is instructive to discuss whether the substitution attack against the bank transfer that worked for the ECB mode is applicable to the CBC mode. If the IV is properly chosen for every wire transfer, the attack will not work at all since Oscar will not recognize any patterns in the ciphertext. If the IV is kept the same for several transfers, he would recognize the transfers from his account at bank A to

his account at bank B. However, if he substitutes ciphertext block 4, which is his encrypted account number, in other wire transfers going from bank A to B, bank B would decrypt block 4 and 5 to some random value. Even though money would not be redirected into Oscar's account, it might be redirected to some other random account. The amount would be a random value too. This is obviously also highly undesirable for banks. This example shows that even though Oscar cannot perform specific manipulations, ciphertext alterations by him can cause random changes to the plaintext, which can have major negative consequences. Hence in many, if not in most, real-world systems, encryption itself is not sufficient: we also have to protect the integrity of the message. This can be achieved by message authentication codes (MACs) or digital signatures, which are introduced in Chap. 12. The Galois Counter mode described below provides encryption and integrity check simultaneously.

5.1.3 Output Feedback Mode (OFB)

In the *Output Feedback (OFB)* mode a block cipher is used to build a stream cipher encryption scheme. This scheme is shown in Fig. 5.5. Note that in OFB mode the key stream is not generated bitwise but instead in a blockwise fashion. The output of the cipher gives us b key stream bits, where b is the width of the block cipher used, with which we can encrypt b plaintext bits using the XOR operation.

The idea behind the OFB mode is quite simple. We start with encrypting an IV with a block cipher. The cipher output gives us the first set of b key stream bits. The next block of key stream bits is computed by feeding the previous cipher output back into the block cipher and encrypting it. This process is repeated as shown in Fig. 5.5.

The OFB mode forms a synchronous stream cipher (cf. Fig. 2.3) as the key stream does not depend on the plain or ciphertext. In fact, using the OFB mode is quite similar to using a standard stream cipher such as RC4 or Trivium. Since the OFB mode forms a stream cipher, encryption and decryption are exactly the same operation. As can be seen in the right-hand part of Fig. 5.5, the receiver does not use the block cipher in decryption mode $e^{-1}()$ to decrypt the ciphertext. This is because the actual encryption is performed by the XOR function, and in order to reverse it, i.e., to decrypt it, we simply have to perform another XOR function on the receiver side. This is in contrast to ECB and CBC mode, where the data is actually being encrypted and decrypted by the block cipher.

Encryption and decryption using the OFB scheme is as follows:

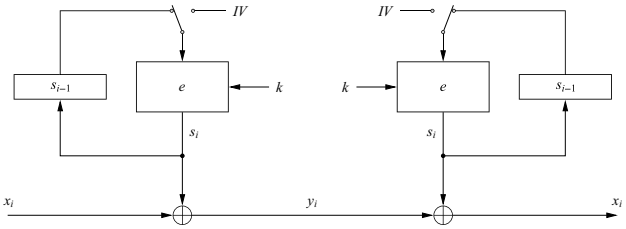

Fig. 5.5 Encryption and decryption in OFB mode

Definition 5.1.3 Output feedback mode (OFB)
Let $e()$ be a block cipher of block size b; let x_i, y_i and s_i be bit strings of length b; and IV be a nonce of length b.
Encryption (first block): $s_1 = e_k(IV)$ *and* $y_1 = s_1 \oplus x_1$
Encryption (general block): $s_i = e_k(s_{i-1})$ *and* $y_i = s_i \oplus x_i, \quad i \geq 2$
Decryption (first block): $s_1 = e_k(IV)$ *and* $x_1 = s_1 \oplus y_1$
Decryption (general block): $s_i = e_k(s_{i-1})$ *and* $x_i = s_i \oplus y_i, \quad i \geq 2$

As a result of the use of an IV, the OFB encryption is also nondeterministic, hence, encrypting the same plaintext twice results in different ciphertexts. As in the case for the CBC mode, the IV should be a nonce. One advantage of the OFB mode is that the block cipher computations are independent of the plaintext. Hence, one can precompute one or several blocks s_i of key stream material.

5.1.4 Cipher Feedback Mode (CFB)

The *Cipher Feedback (CFB)* mode also uses a block cipher as a building block for a stream cipher. It is similar to the OFB mode but instead of feeding back the output of the block cipher, the ciphertext is fed back. (Hence, a somewhat more accurate term for this mode would have been "Ciphertext Feedback mode".) As in the OFB mode, the key stream is not generated bitwise but instead in a blockwise fashion. The idea behind the CFB mode is as follows: To generate the first key stream block s_1, we encrypt an IV. For all subsequent key stream blocks $s_2, s_3, \ldots$, we encrypt the previous ciphertext. This scheme is shown in Fig. 5.6.

Since the CFB mode forms a stream cipher, encryption and decryption are exactly the same operation. The CFB mode is an example of an asynchronous stream cipher (cf. Fig. 2.3) since the stream cipher output is also a function of the ciphertext.

The formal description of the CFB mode follows:

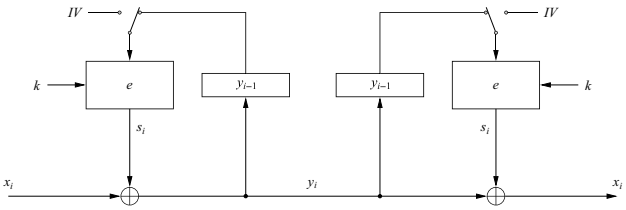

Fig. 5.6 Encryption and decryption in CFB mode

Definition 5.1.4 Cipher feedback mode (CFB)
Let $e()$ be a block cipher of block size b; let x_i and y_i be bit strings of length b; and IV be a nonce of length b.
Encryption (first block): $y_1 = e_k(IV) \oplus x_1$
Encryption (general block): $y_i = e_k(y_{i-1}) \oplus x_i, \quad i \geq 2$
Decryption (first block): $x_1 = e_k(IV) \oplus y_1$
Decryption (general block): $x_i = e_k(y_{i-1}) \oplus y_i, \quad i \geq 2$

As a result of the use of an IV, the CFB encryption is also nondeterministic, hence, encrypting the same plaintext twice results in different ciphertexts. As in the case for the CBC and OFB modes, the IV should be a nonce.

A variant of the CFB mode can be used in situations where short plaintext blocks are to be encrypted. Let's use the encryption of the link between a (remote) keyboard and a computer as an example. The plaintexts generated by the keyboard are typically only 1 byte long, e.g., an ASCII character. In this case, only 8 bits of the key stream are used for encryption (it does not matter which ones we choose as they are all secure), and the ciphertext also only consists of 1 byte. The feedback of the ciphertext as input to the block cipher is a bit tricky. The previous block cipher input is shifted by 8 bit positions to the left, and the 8 least significant positions of the input register are filled with the ciphertext byte. This process repeats. Of course, this approach works not only for plaintext blocks of length 8, but for any lengths shorter than the cipher output.

5.1.5 Counter Mode (CTR)

Another mode which uses a block cipher as a stream cipher is the Counter (CTR) mode. As in the OFB and CFB modes, the key stream is computed in a blockwise fashion. The input to the block cipher is a counter which assumes a different value every time the block cipher computes a new key stream block. Figure 5.7 shows the principle.

We have to be careful how to initialize the input to the block cipher. We must prevent using the same input value twice. Otherwise, if an attacker knows one of

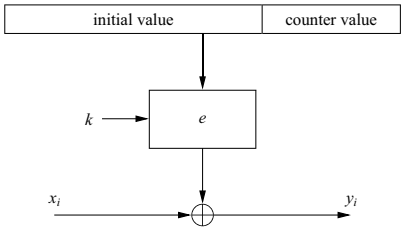

Fig. 5.7 Encryption and decryption in counter mode

the two plaintexts that were encrypted with the same input, he can compute the key stream block and thus immediately decrypt the other ciphertext. In order to achieve this uniqueness, often the following approach is taken in practice. Let's assume a block cipher with an input width of 128 bits, such as an AES. First we choose an IV that is a nonce with a length smaller than the block length, e.g., 96 bits. The remaining 32 bits are then used by a counter with the value CTR which is initialized to zero. For every block that is encrypted during the session, the counter is incremented but the IV stays the same. In this example, the number of blocks we can encrypt without choosing a new IV is 2^{32}. Since every block consists of 8 bytes, a maximum of $8 \times 2^{32} = 2^{35}$ bytes, or about 32 Gigabytes, can be encrypted before a new IV must be generated. Here is a formal description of the Counter mode with a cipher input construction as just introduced:

Definition 5.1.5 Counter mode (CTR)
Let $e()$ be a block cipher of block size b, and let x_i and y_i be bit strings of length b. The concatenation of the initialization value IV and the counter CTR_i is denoted by $(IV||CTR_i)$ and is a bit string of length b.
Encryption: $y_i = e_k(IV||CTR_i) \oplus x_i, \quad i \geq 1$
Decryption: $x_i = e_k(IV||CTR_i) \oplus y_i, \quad i \geq 1$

Please note that the string $(IV||CTR_1)$ does not have to be kept secret. It can, for instance, be generated by Alice and sent to Bob together with the first ciphertext block. The counter CTR can either be a regular integer counter or a slightly more complex function such as a maximum-length LFSR.

One might wonder why so many modes are needed. One attractive feature of the Counter mode is that it can be parallelized because, unlike the OFB or CFB mode, it does not require any feedback. For instance, we can have two block cipher engines running in parallel, where the first block cipher encrypts the counter value CTR_1 and the other CTR_2 at the same time. When the two block cipher engines are finished, the first engine encrypts the value CTR_3 and the other one CTR_4, and so on. This scheme would allow us to encrypt at twice the data rate of a single implementation. Of course, we can have more than two block ciphers running in parallel, increasing the speed-up proportionally. For applications with high throughput demands, e.g.,

in networks with data rates in the range of Gigabits per second, encryption modes that can be parallelized are very desirable.

5.1.6 Galois Counter Mode (GCM)

The *Galois Counter Mode (GCM)* is an encryption mode which also computes a message authentication code (MAC) [160]. A MAC provides a cryptographic checksum that is computed by the sender, Alice, and appended to the message. Bob also computes a MAC from the message and checks whether his MAC is the same as the one computed by Alice. This way, Bob can make sure that (1) the message was really created by Alice and (2) that nobody tampered with the ciphertext during transmission. These two properties are called message authentication and integrity, respectively. Much more about MACs is found in Chap. 12. We presented a slightly simplified version of the GCM mode in the following.

GCM protects the confidentiality of the plaintext x by using an encryption in counter mode. Additionally, GCM protects not only the authenticity of the plaintext x but also the authenticity of a string *AAD* called *additional authenticated data*. This authenticated data is, in contrast to the plaintext, left in clear in this mode of operation. In practice, the string *AAD* might include addresses and parameters in a network protocol.

The GCM consists of an underlying block cipher and a Galois field multiplier with which the two GCM functions *authenticated encryption* and *authenticated decryption* are realized. The cipher needs to have a block size of 128 bits such as AES. On the sender side, GCM encrypts data using the Counter Mode (CTR) followed by the computation of a MAC value. For encryption, first an initial counter is derived from an IV and a serial number. Then the initial counter value is incremented, and this value is encrypted and XORed with the first plaintext block. For subsequent plaintexts, the counter is incremented and then encrypted. Note that the underlying block cipher is only used in encryption mode. GCM allows for precomputation of the block cipher function if the initialization vector is known ahead of time.

For authentication, GCM performs a chained Galois field multiplication. For every plaintext x_i an intermediate authentication parameter g_i is derived. g_i is computed as the XOR sum of the current ciphertext y_i and g_i, and multiplied by the constant H. The value H is a hash subkey which is generated by encryption of the all-zero input with the block cipher. All multiplications are in the 128-bit Galois field $GF(2^{128})$ with the irreducible polynomial $P(x) = x^{128} + x^7 + x^2 + x + 1$. Since only one multiplication is required per block cipher encryption, the GCM mode adds very little computational overhead to the encryption.

Definition 5.1.6 Basic Galois Counter mode (GCM)
Let $e()$ be a block cipher of block size 128 bit; let x be the plaintext consisting of the blocks $x_1, \ldots, x_n$; and let AAD be the additional authenticated data.

1. ***Encryption***
   a. *Derive a counter value CTR_0 from the IV and compute $CTR_1 = CTR_0 + 1$.*
   b. *Compute ciphertext: $y_i = e_k(CTR_i) \oplus x_i, \quad i \geq 1$*

2. ***Authentication***
   a. *Generate authentication subkey $H = e_k(0)$*
   b. *Compute $g_0 = AAD \times H$ (Galois field multiplication)*
   c. *Compute $g_i = (g_{i-1} \oplus y_i) \times H, \quad 1 \leq i \leq n$ (Galois field multiplication)*
   d. *Final authentication tag: $T = (g_n \times H) \oplus e_k(CTR_0)$*

Figure 5.8 shows a diagram of the GCM.

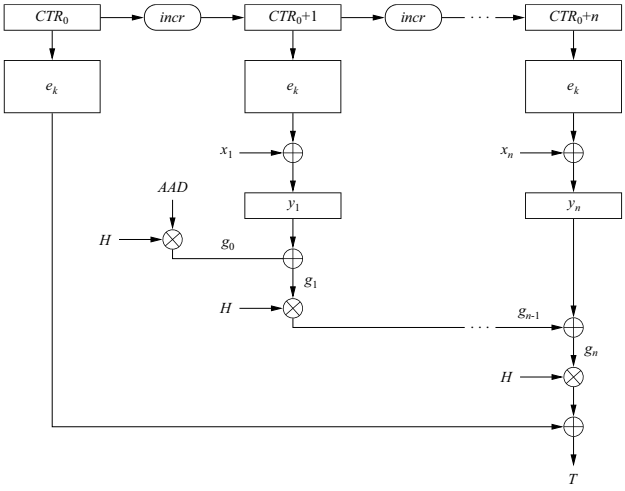

Fig. 5.8 Basic authenticated encryption in Galois Counter mode

The receiver of the packet $[(y_1, \ldots, y_n), T, ADD]$ decrypts the ciphertext by also applying the Counter mode. To check the authenticity of the data, the receiver also computes an authentication tag T' using the received ciphertext and ADD as input. He employs exactly the same steps as the sender. If T and T' match, the receiver is

assured that the cipertext (and *ADD*) were not manipulated in transit and that only the sender could have generated the message.

5.2 Exhaustive Key Search Revisited

In Sect. 3.5.1 we saw that given a plaintext–ciphertext pair (x_1, y_1) a DES key can be exhaustively searched using the simple algorithm:

$$DES_{k_i}(x_1) \stackrel{?}{=} y_1, \quad i = 0, 1, \ldots, 2^{56} - 1. \tag{5.1}$$

For most other block ciphers, however, a key search is somewhat more complicated. Somewhat surprisingly, a brute-force attack can produce *false positive* results, i.e., keys k_i are found that are not the one used for the encryption, yet they perform a correct encryption in Eq. (5.1). The likelihood of this occurring is related to the relative size of the key space and the plaintext space.

A brute-force attack is still possible, but several pairs of plaintext–ciphertext are needed. The length of the respective plaintext required to break the cipher with a brute-force attack is referred to as *unicity distance*. After trying every possible key, there should be just one plaintext that makes sense.

Let's first look why one pair (x_1, y_1) might not be sufficient to identify the correct key. For illustration purposes we assume a cipher with a block width of 64 bit and a key size of 80 bit. If we encrypt x_1 under all possible 2^{80} keys, we obtain 2^{80} ciphertexts. However, there exist only 2^{64} different ones, and thus some keys must map x_1 to the same ciphertext. If we run through all keys for a given plaintext–ciphertext pair, we find on average $2^{80}/2^{64} = 2^{16}$ keys that perform the mapping $e_k(x_1) = y_1$. This estimation is valid since the encryption of a plaintext for a given key can be viewed as a random selection of a 64-bit ciphertext string. The phenomenon of multiple "paths" between a given plaintext and ciphertext is depicted in Fig. 5.9, in which $k^{(i)}$ denote the keys that map x_1 to y_1. These keys can be considered key candidates.

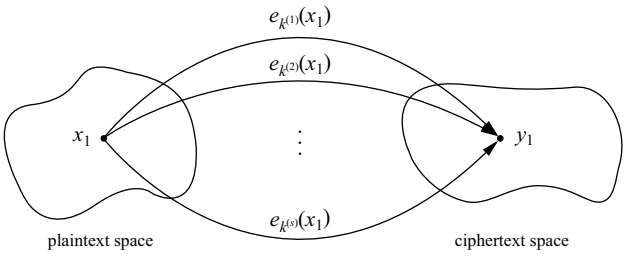

Fig. 5.9 Multiple keys map between one plaintext and one ciphertext

Among the approximately 2^{16} key candidates $k^{(i)}$ is the correct one that was used by to perform the encryption. Let's call this one the target key. In order to identify the target key we need a second plaintext–ciphertext pair (x_2, y_2). Again, there are about 2^{16} key candidates that map x_2 to y_2. One of them is the target key. The other keys can be viewed as randomly drawn from the 2^{80} possible ones. It is crucial to note that the target key must be present in *both* sets of key candidates. To determine the effectiveness of a brute-force attack, the crucial question is now: What is the likelihood that another (false!) key is contained in both sets? The answer is given by the following theorem:

Theorem 5.2.1 *Given a block cipher with a key length of κ bits and block size of n bits, as well as t plaintext–ciphertext pairs $(x_1, y_1), \ldots, (x_t, y_t)$, the expected number of false keys which encrypt all plaintexts to the corresponding ciphertexts is:*

$$2^{\kappa - tn}$$

Returning to our example and assuming two plaintext–ciphertext pairs, the likelihood of a false key k_f that performs both encryptions $e_{k_f}(x_1) = y_1$ and $e_{k_f}(x_2) = y_2$ is:

$$2^{80-2\cdot 64} = 2^{-48}$$

This value is so small that for almost all practical purposes it is sufficient to test two plaintext–ciphertext pairs. If the attacker chooses to test three pairs, the likelihood of a false key decreases to $2^{80-3\cdot 64} = 2^{-112}$. As we saw from this example, the likelihood of a false alarm decreases rapidly with the number t of plaintext–ciphertext pairs. In practice, typically we only need a few pairs.

The theorem above is not only important if we consider an individual block cipher but also if we perform multiple encryptions with a cipher. This issue is addressed in the following section.

5.3 Increasing the Security of Block Ciphers

In some situations we wish to increase the security of block ciphers, e.g., if a cipher such as DES is available in hardware or software for legacy reasons in a given application. We discuss two general approaches to strengthen a cipher, multiple encryption and key whitening. Multiple encryption, i.e., encrypting a plaintext more than once, is already a fundamental design principle of block ciphers, since the round function is applied many times to the cipher. Our intuition tells us that the security of a block cipher against both brute-force and analytical attacks increases by performing multiple encryptions in a row. Even though this is true in principle, there are a few surprising facts. For instance, doing double encryption does very little to increase the brute-force resistance over a single encryption. We study this

counterintuitive fact in the next section. Another very simple yet effective approach to increase the brute-force resistance of block ciphers is called key whitening; it is also discussed below.

We note here that when using AES, we already have three different security levels given by the key lengths of 128, 192 and 256 bits. Given that there are no realistic attacks known against AES with any of those key lengths, there appears no reason to perform multiple encryption with AES for practical systems. However, for some selected older ciphers, especially for DES, multiple encryption can be a useful tool.

5.3.1 Double Encryption and Meet-in-the-Middle Attack

Let's assume a block cipher with a key length of κ bits. For *double encryption*, a plaintext x is first encrypted with a key k_L, and the resulting ciphertext is encrypted again using a second key k_R. This scheme is shown in Fig. 5.10.

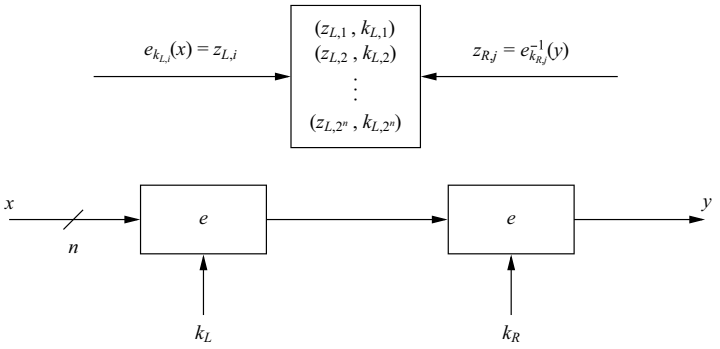

Fig. 5.10 Double encryption and meet-in-the-middle attack

A naïve brute-force attack would require us to search through all possible combinations of both keys, i.e., the effective key lengths would be 2κ and an exhaustive key search would require $2^\kappa \cdot 2^\kappa = 2^{2\kappa}$ encryptions (or decryptions). However, using the *meet-in-the-middle* attack, the key space is drastically reduced. This is a divide-and-conquer attack in which Oscar first brute-force-attacks the encryption on the left-hand side, which requires 2^κ cipher operations, and then the right encryption, which again requires 2^κ operations. If he succeeds with this attack, the total complexity is $2^\kappa + 2^\kappa = 2 \cdot 2^\kappa = 2^{\kappa+1}$. This is barely more complex than a key search of a single encryption and of course is much less complex than performing $2^{2\kappa}$ search operations.

The attack has two phases. In the first one, the left encryption is brute-forced and a lookup table is computed. In the second phase the attacker tries to find a match in the table which reveals both encryption keys. Here are the details of this approach.

Phase I: Table Computation For a given plaintext x_1, compute a lookup table for all pairs $(k_{L,i}, z_{L,i})$, where $e_{k_{L,i}}(x_1) = z_{L,i}$ and $i = 1, 2, \ldots, 2^\kappa$. These computations are symbolized by the left arrow in the figure. The $z_{L,i}$ are the intermediate values that occur in between the two encryptions. This list should be ordered by the values of the $z_{L,i}$. The number of entries in the table is 2^κ, with each entry being $n + \kappa$ bits wide. Note that one of the keys we used for encryption must be the correct target key, but we still do not know which one it is.

Phase II: Key Matching In order to find the key, we now decrypt y_1, i.e., we perform the computations symbolized by the right arrow in the figure. We select the first possible key $k_{R,1}$, e.g., the all-zero key, and compute:

$$e^{-1}_{k_{R,1}}(x_1) = z_{R,1}.$$

We now check whether $z_{R,1}$ is equal to any of the $z_{L,i}$ values in the table which we computed in the first phase. If it is *not* in the table, we increment the key to $k_{R,1}$, decrypt y_1 again, and check whether this value is in the table. We continue until we have a match.

We now have what is called a *collision* of two values, i.e., $z_{L,i} = z_{R,j}$. This gives us two keys: The value $z_{L,i}$ is associated with the key $k_{L,i}$ from the left encryption, and $k_{R,j}$ is the key we just tested from the right encryption. This means there exists a key pair $(k_{L,i}, k_{R,j})$ which performs the double encryption:

$$e_{k_{R,j}}(e_{k_{L,i}}(x_1)) = y_1 \tag{5.2}$$

As discussed in Sect. 5.2, there is a chance that this is not the target key pair we are looking for since there are most likely several possible key pairs that perform the mapping $x_1 \rightarrow y_1$. Hence, we have to verify additional key candidates by encrypting several plaintext–ciphertext pairs according to Eq. (5.2). If the verification fails for any of the pairs $(x_1, y_1), (x_2, y_2), \ldots$, we go back to beginning of Phase II and increment the key k_R again and continue with the search.

Let's briefly discuss how many plaintext–ciphertext pairs we will need to rule out faulty keys with a high likelihood. With respect to multiple mappings between a plaintext and a ciphertext as depicted in Fig. 5.9, double encryption can be modeled as a cipher with 2κ key bits and n block bits. In practice, one often has $2\kappa > n$, in which case we need several plaintext–ciphertext pairs. The theorem in Sect. 5.2 can easily be adopted to the case of multiple encryption, which gives us a useful guideline about how many (x, y) pairs should be available:

Theorem 5.3.1 *Given are l subsequent encryptions with a block cipher with a key length of κ bits and block size of n bits, as well as t plaintext–ciphertext pairs $(x_1, y_1), \ldots, (x_t, y_t)$. The expected number of false keys which encrypt all plaintexts to the corresponding ciphertexts is given by:*

$$2^{l\kappa - tn}$$

Let's look at an example.

Example 5.3. As an example, if we double-encrypt with DES and choose to test three plaintext–ciphertext pairs, the likelihood of a faulty key pair surviving all three key tests is:

$$2^{2 \cdot 56 - 3 \cdot 64} = 2^{-80}.$$

◇

Let us examine the computational complexity of the meet-in-the-middle attack. In the first phase of the attack, corresponding to the left arrow in the figure, we perform 2^κ encryptions and store them in 2^κ memory locations. In the second stage, corresponding to the right arrow in the figure, we perform a maximum of 2^κ decryptions and table look-ups. We ignore multiple key tests at this stage. The total cost for the meet-in-the-middle attack is:

$$\text{number of encryptions and decryptions} = 2^\kappa + 2^\kappa = 2^{\kappa+1}$$
$$\text{number of storage locations} = 2^\kappa$$

This compares to 2^κ encryptions or decryptions and essentially no storage cost in the case of a brute-force attack against a single encryption. Even though the storage requirements go up quite a bit, the costs in computation and memory are still only proportional to 2^κ. Thus, it is widely believed that double encryption is not worth the effort. Instead, triple encryption should be used; this method is described in the following section.

Note that for a more exact complexity analysis of the meet-in-the-middle attack, we would also need take the cost of sorting the table entries in Phase I into account as well as the table look-ups in Phase II. For our purposes, however, we can ignore these additional costs.

5.3.2 Triple Encryption

Compared to double encryption, a much more secure approach is the encryption of a block of data three times in a row:

$$y = e_{k_3}(e_{k_2}(e_{k_1}(x))).$$

In practice, often a variant of the triple encryption from above is used:

$$y = e_{k_1}(e_{k_2}^{-1}(e_{k_3}(x))).$$

This type of triple encryption is sometimes referred to as encryption–decryption–encryption (EDE). The reason for this has nothing to do with security. If $k_1 = k_2$, the operation effectively performed is

$$y = e_{k_3}(x),$$

which is single encryption. Since it is sometimes desirable that one implementation can perform both triple encryption and single encryption, i.e., in order to interoperate with legacy systems, EDE is a popular choice for triple encryption. Moreover, for a 112-bit security, it is sufficient to choose two different keys k_1 and k_2 and set $k_3 = k_1$ in case of 3DES.

Of course, we can still perform a meet-in-the-middle attack as shown in Fig. 5.11.

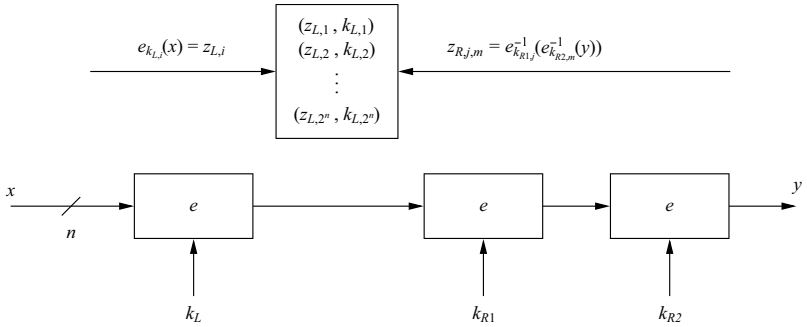

Fig. 5.11 Triple encryption and sketch of a meet-in-the-middle attack

Again, we assume κ bits per key. The problem for an attacker is that she has to compute a lookup table either after the first or after the second encryption. In both cases, the attacker has to compute two encryptions or decryptions in a row in order to reach the lookup table. Here lies the cryptographic strength of triple encryption: There are 2^{2k} possibilities to run through all possible keys of two encryptions or decryptions. In the case of 3DES, this forces an attacker to perform 2^{112} key tests, which is entirely infeasible with current technology. In summary, the meet-in-the-middle attack reduces the *effective key length* of triple encryption from $3\,\kappa$ to $2\,\kappa$. Because of this, it is often said that the *effective key length* of triple DES is 112 bits as opposed to $3 \cdot 56 = 168$ bits which are actually used as input to the cipher.

5.3.3 Key Whitening

Using an extremely simple technique called *key whitening*, it is possible to make block ciphers such as DES much more resistant against brute-force attacks. The basic scheme is shown in Fig. 5.12.

In addition to the regular cipher key k, two whitening keys k_1 and k_2 are used to XOR-mask the plaintext and ciphertext. This process can be expressed as:

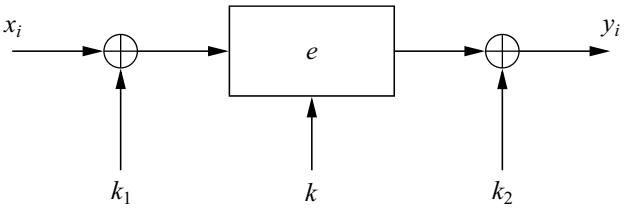

Fig. 5.12 Key whitening of a block cipher

Definition 5.3.1 Key whitening for block ciphers
Encryption: $y = e_{k,k_1,k_2}(x) = e_k(x \oplus k_1) \oplus k_2$
Decryption: $x = e^{-1}_{k,k_1,k_2}(x) = e^{-1}_k(y \oplus k_2) \oplus k_1$

It is important to stress that key whitening does not strengthen block ciphers against most analytical attacks such as linear and differential cryptanalysis. This is in contrast to multiple encryption, which often also increases the resistance to analytical attacks. Hence, key whitening is not a "cure" for inherently weak ciphers. Its main application is ciphers that are relatively strong against analytical attacks but possess too short a key space. The prime example of such a cipher is DES. A variant of DES which uses key whitening is *DESX*. In the case of DESX, the key k_2 is derived from k and k_1. Please note that most modern block ciphers such as AES already apply key whitening internally by adding a subkey prior to the first round and after the last round.

Let's now discuss the security of key whitening. A naïve brute-force attack against the scheme requires $2^{\kappa+2n}$ search steps, where κ is the bit length of the key and n the block size. Using the meet-in-the-middle attack introduced in Sect. 5.3, the computational load can be reduced to approximately $2^{\kappa+n}$ steps, plus storage of 2^n data sets. However, if the adversary Oscar can collect 2^m plaintext–ciphertext pairs, a more advanced attack exists with a computational complexity of

$$2^{\kappa+n-m}$$

cipher operations. Even though we do not introduce the attack here, we'll briefly discuss its consequences if we apply key whitening to DES. We assume that the attacker knows 2^m plaintext–ciphertext pairs. Note that the designer of a security system can often control how many plaintext–ciphertext are generated before a new key is established. Thus, the parameter m cannot be arbitrarily increased by the attacker. Also, since the number of known plaintexts grows exponentially with m, values beyond, say, $m = 40$, seem quite unrealistic. As a practical example, let's assume key whitening of DES, and that Oscar can collect a maximum of 2^{32} plaintexts. He now has to perform

$$2^{56+64-32} = 2^{88}$$

DES computations. Given that with today's technology even 2^{56} DES operations require several days with special hardware, performing 2^{88} encryptions is completely out of reach. Note that the number of plaintexts (which Oscar is not supposed to know in most circumstances) corresponds to 32 GByte of data, the collection of which is also a formidable task in most real-world situations.

A particular attractive feature of key whitening is that the additional computational load is negligible. A typical block cipher implementation in software requires several hundred instructions for encrypting one input block. In contrast, a 64-bit XOR operation only takes 2 instructions on a 32-bit machine, so that the performance impact due to key whitening is in the range of 1% or less in most cases.

5.4 Discussion and Further Reading

Modes of Operation After the AES selection process, the US National Institute of Standards and Technology (NIST) supported the process of evaluating new modes of operations in a series of special publications and workshops [124]. Currently, there are eight approved block cipher modes: five for confidentiality (ECB, CBC, CFB, OFB, CTR), one for authentication (CMAC) and two combined modes for confidentiality and authentication (CCM, GCM). The modes are widely used in practice and are part of many standards, e.g., for computer networks or banking.

Other Applications for Block Ciphers The most important application of block ciphers in practice, in addition to data encryption, is *Message Authentication Codes (MACs)*, which are discussed in Chap. 12. The schemes CBC-MAC, OMAC and PMAC are constructed with a block cipher. *Authenticated Encryption (AE)* uses block ciphers to both encrypt and generate a MAC in order to provide confidentiality and authentication, respectively. In addition to the GCM introduced in this chapter, other AE modes include the *EAX* mode, *OCB* mode, and *GC* mode.

Another application is the *Cryptographically Secure Pseudo Random Number Generators (CSPRNG)* built from block ciphers. In fact, the stream cipher modes introduced in this chapter, OFB, CFB and CTR mode, form CSPRNGs. There are also standards such as [4, Appendix A.2.4] which explicitly specify random number generators from block ciphers.

Block ciphers can also be used to build *cryptographic hash functions*, as discussed in Chap. 11.

Extending Brute-Force Attacks Even though there are no algorithmic shortcuts to brute-force attacks, there are methods which are efficient if several exhaustive key searches have to be performed. Those methods are called time–memory tradeoff attacks (TMTO). The general idea is to encrypt a fixed plaintext under a large number of keys and to store certain intermediate results. This is the precomputation phase, which is typically at least as complex as a single brute-force attack and which results in large lookup tables. In the online phase, a search through the tables takes place which is considerably faster than a brute-force attack. Thus, after the precomputa-

tion phase, individual keys can be found much more quickly. TMTO attacks were originally proposed by Hellman [91] and were improved with the introduction of distinguished points by Rivest [145]. More recently rainbow tables were proposed to further improve TMTO attacks [131]. A limiting factor of TMTO attacks in practice is that for each individual attack it is required that the same piece of known plaintext was encrypted, e.g., a file header.

Block Ciphers and Quantum Computers With the potential rise of quantum computers in the future, the security of currently used crypto algorithms has to be reevaluated. (It should be noted that the possible existence of quantum computers in a few decades from now is hotly debated.) Whereas all popular existing asymmetric algorithms such as RSA are vulnerable to attacks using quantum computers [153], symmetric algorithms are much more resilient. A potential quantum computer using Grover's algorithm [87] would require only $2^{(n/2)}$ steps in order to perform a complete key search on a cipher with a keyspace of 2^n elements. Hence, key lengths of more than 128 bit are required if resistance against quantum computer attacks is desired. This observation was also the motivation for requiring the 192-bit and 256-bit key lengths for AES. Interestingly, it can be shown that there can be no quantum algorithm which performs such an attack more efficiently than Grover's algorithm [16].

5.5 Lessons Learned

- There are many different ways to encrypt with a block cipher. Each mode of operation has some advantages and disadvantages.
- Several modes turn a block cipher into a stream cipher.
- There are modes that perform encryption together together with authentication, i.e., a cryptographic checksum protects against message manipulation.
- The straightforward ECB mode has security weaknesses, independent of the underlying block cipher.
- The counter mode allows parallelization of encryption and is thus suited for high-speed implementations.
- Double encryption with a given block cipher only marginally improves the resistance against brute-force attacks.
- Triple encryption with a given block cipher roughly *doubles* the key length. Triple DES (3DES) has an effective key length of 112 bits.
- Key whitening enlarges the DES key length without much computational overhead.

Problems

5.1. Consider the storage of data in encrypted form in a large database using AES. One record has a size of 16 bytes. Assume that the records are not related to one another. Which mode would be best suited and why?

5.2. We consider known-plaintext attacks on block ciphers by means of an exhaustive key search where the key is k bits long. The block length counts n bits with $n > k$.

1. How many plaintexts and ciphertexts are needed to successfully break a block cipher running in ECB mode? How many steps are done in the worst case?
2. Assume that the initialization vector IV for running the considered block cipher in CBC mode is known. How many plaintexts and ciphertexts are now needed to break the cipher by performing an exhaustive key search? How many steps need now maximally be done? Briefly describe the attack.
3. How many plaintexts and ciphertexts are necessary, if you do *not* know the IV?
4. Is breaking a block cipher in CBC mode by means of an exhaustive key search considerably more difficult than breaking an ECB mode block cipher?

5.3. In a company, all files which are sent on the network are automatically encrypted by using AES-128 in CBC mode. A fixed key is used, and the IV is changed once per day. The network encryption is file-based, so that the IV is used at the beginning of every file.

You managed to spy out the fixed AES-128 key, but do not know the recent IV. Today, you were able to eavesdrop two different files, one with unidentified content and one which is known to be an automatically generated temporary file and only contains the value `0xFF`. Briefly describe how it is possible to obtain the unknown initialization vector and how you are able to determine the content of the unknown file.

5.4. Keeping the IV secret in OFB mode does not make an exhaustive key search more complex. Describe how we can perform a brute-force attack with unknown IV. What are the requirements regarding plaintext and ciphertext?

5.5. Describe how the OFB mode can be attacked if the IV is *not* different for each execution of the encryption operation.

5.6. Propose an OFB mode scheme which encrypts one byte of plaintext at a time, e.g., for encrypting key strokes from a remote keyboard. The block cipher used is AES. Perform one block cipher operation for every new plaintext byte. Draw a block diagram of your scheme and pay particular attention to the bit lengths used in your diagram (cf. the descripton of a byte mode at the end of Sect. 5.1.4).

5.7. As is so often true in cryptography, it is easy to weaken a seemingly strong scheme by small modifications. Assume a variant of the OFB mode by which we only feed back the 8 most significant bits of the cipher output. We use AES and fill the remaining 120 input bits to the cipher with 0s.

1. Draw a block diagram of the scheme.
2. Why is this scheme weak if we encrypt moderately large blocks of plaintext, say 100 kByte? What is the maximum number of known plaintexts an attacker needs to completely break the scheme?
3. Let the feedback byte be denoted by FB. Does the scheme become cryptographically stronger if we feedback the 128-bit value $FB, FB, \ldots, FB$ to the input (i.e., we copy the feedback byte 16 times and use it as AES input)?

5.8. In the text, a variant of the CFB mode is proposed which encrypts individual bytes. Draw a block diagram for this mode when using AES as block cipher. Indicate the width (in bit) of each line in your diagram.

5.9. We are using AES in counter mode for encrypting a hard disk with 1 TB of capacity. What is the maximum length of the IV?

5.10. Sometimes error propagation is an issue when choosing a mode of operation in practice. In order to analyze the propagation of errors, let us assume a bit error (i.e., a substitution of a "0" bit by a "1" bit or vice versa) in a ciphertext block y_i.

1. Assume an error occurs during the transmission in one block of ciphertext, let's say y_i. Which cleartext blocks are affected on Bob's side when using the ECB mode?
2. Again, assume block y_i contains an error introduced during transmission. Which cleartext blocks are affected on Bob's side when using the CBC mode?
3. Suppose there is an error in the cleartext x_i on Alice's side. Which cleartext blocks are affected on Bob's side when using the CBC mode?
4. Assume a single bit error occurs in the transmission of a ciphertext character in 8-bit CFB mode. How far does the error propagate? Describe exactly *how* each block is affected.
5. Prepare an overview of the effect of bit errors in a ciphertext block for the modes ECB, CBC, CFB, OFB and CTR. Differentiate between random bit errors and specific bit errors when decrypting y_i.

5.11. Besides simple bit errors, the deletion or insertion of a bit yields even more severe effects since the synchronization of blocks is disrupted. In most cases, the decryption of subsequent blocks will be incorrect. A special case is the CFB mode with a feedback width of 1 bit. Show that the synchronization is automatically restored after $\kappa + 1$ steps, where κ is the block size of the block cipher.

5.12. We now analyze the security of DES double encryption (2DES) by doing a cost-estimate:

$$2DES(x) = DES_{K_2}(DES_{K_1}(x))$$

1. First, let us assume a pure key search without any memory usage. For this purpose, the whole key space spanned by K_1 and K_2 has to be searched. How much does a key-search machine for breaking 2DES (worst case) in 1 week cost?
   In this case, assume ASICs which can perform 10^7 keys per second at a cost of \$5 per IC. Furthermore, assume an overhead of 50% for building the key search machine.

2. Let us now consider the meet-in-the-middle (or time-memory tradeoff) attack, in which we can use memory. Answer the following questions:
   - How many entries have to be stored?
   - How many bytes (not bits!) have to be stored for each entry?
   - How costly is a key search in one week? Please note that the key space has to be searched before filling up the memory completely. Then we can begin to search the key space of the second key. Assume the same hardware for both key spaces.

   For a rough cost estimate, assume the following costs for hard disk space: \$8/10 GByte, where 1 GByte = 10^9 Byte.
3. Assuming Moore's Law, when do the costs move below \$1 million?

5.13. Imagine that aliens — rather than abducting earthlings and performing strange experiments on them — drop a computer on planet Earth that is particularly suited for AES key searches. In fact, it is so powerful that we can search through 128, 192 and 256 key bits in a matter of days. Provide guidelines for the number of plaintext–ciphertext pairs the aliens need so that they can rule out false keys with a reasonable likelihood. (**Remark**: Since the existence of both aliens and human-built computers for such key lengths seem extremely unlikely at the time of writing, this problem is pure science fiction.)

5.14. Given multiple plaintext–ciphertext pairs, your objective is to attack an encryption scheme based upon multiple encryptions.

1. You want to break an encryption system E, which makes use of triple AES-192 encryption (e.g. block length $n = 128$ bit, key size of $k = 192$ bit). How many tuples (x_i, y_i) with $y_i = e_K(x_i)$ do you need to level down the probability of finding a key K, which matches the condition $y_i = e_K(x_i)$ for one particular i, but fails for most other values of i (a so called *false positive*), to $Pr(K' \neq K) = 2^{-20}$?
2. What is the maximum key size of a block cipher that you could still effectively attack with an error probability of at most $Pr(K' \neq K) = 2^{-10} = 1/1024$, if this cipher always uses double encryption ($l = 2$) and has a block length of $n = 80$ bit?
3. Estimate the success probability, if you are provided with four plaintext–ciphertext blocks which are double encrypted using AES-256 ($n = 128$ bits, $k = 256$ bits). Please justify your results.

Note that this is a purely theoretical problem. Key spaces of size 2^{128} and beyond can not be brute-forced.

5.15. 3DES with three different keys can be broken with about 2^{2k} encryptions and 2^k memory cells, $k = 56$. Design the corresponding attack. How many pairs (x, y) should be available so that the probability to determine an incorrect key triple (k_1, k_2, k_3) is sufficiently low?

5.16. This is your chance to break a cryptosystem. As we know by now, cryptography is a tricky business. The following problem illustrates how easy it is to turn a strong scheme into a weak one with minor modifications.

We saw in this chapter that key whitening is a good technique for strengthening block ciphers against brute-force attacks. We now look at the following variant of key whitening against DES, which we'll call DESA:

$$DESA_{k,k_1}(x) = DES_k(x) \oplus k_1.$$

Even though the method looks similar to key whitening, it hardly adds to the security. Your task is to show that breaking the scheme is roughly as difficult as a brute-force attack against single DES. Assume you have a few pairs of plaintext–ciphertext.