CHAPTER

3

Block Ciphers

Block ciphers are one of the fundamental building blocks for cryptographic systems. There is a lot of literature on block ciphers, and they are among the best-understood parts of cryptography. They are, however, building blocks. For most applications, you probably don't want to use a block cipher directly. Instead, you'll want to use a block cipher in what is called a ''mode of operation,'' which we'll discuss in subsequent chapters. This chapter is designed to give you a firmer understanding of block ciphers: what they are, how cryptographers view them, and how to choose between different options.

3.1 What Is a Block Cipher?

A *block cipher* is an encryption function for fixed-size blocks of data. The current generation of block ciphers has a block size of 128 bits (16 bytes). These block ciphers encrypt a 128-bit plaintext and generate a 128-bit ciphertext as the result. The block cipher is reversible; there is a decryption function that takes the 128-bit ciphertext and decrypts it to the original 128-bit plaintext. The plaintext and ciphertext are always the same size, and we call this the block size of the block cipher.

To encrypt with a block cipher, we need a secret key. Without a secret key, there is no way to hide the message. Like the plaintext and ciphertext, the key is also a string of bits. Common key sizes are 128 and 256 bits. We often write $E(K, p)$ or $E_K(p)$ for the encryption of plaintext p with key K and $D(K, c)$ or $D_K(c)$ for the decryption of ciphertext c with key K.

Block ciphers are used for many purposes, most notably to encrypt information. For security purposes, however, one rarely uses a block cipher directly. Instead, one should use a *block cipher mode*, which we will discuss in Chapter 4.

When using block ciphers, as with any encryption task, we always follow Kerckhoffs' principle and assume that the algorithms for encryption and decryption are publicly known. Some people have a hard time accepting this, and they want to keep the algorithms secret. Don't ever trust a secret block cipher (or any other secret cryptographic primitive).

It is sometimes useful to look at a block cipher as a very big key-dependent table. For any fixed key, you could compute a lookup table that maps the plaintext to the ciphertext. This table would be huge. For a block cipher with 32-bit block size, the table would be 16 GB; for a 64-bit block size, it would be 150 million TB; and for a 128-bit block size it would be $5 \cdot 10^{39}$ bytes, a number so large there is not even a proper name for it. Of course, it is not practical to build such a table in reality, but this is a useful conceptual model. We also know that the block cipher is reversible. In other words, no two entries of the table are the same, or else the decryption function could not possibly decrypt the ciphertext to a unique plaintext. This big table will therefore contain every possible ciphertext value exactly once. This is what mathematicians call a permutation: the table is merely a list of all the possible elements where the order has been rearranged. A block cipher with a block size of k bits specifies a permutation on k-bit values for each of the key values.

As a point of clarification, since it is often confused, a block cipher does not permute the bits of the input plaintext. Rather, it takes all the 2^k possible k-bit inputs and maps each to a unique k-bit output. As a toy example, if $k = 8$, an input 00000001 might encrypt to 0100000 under a given key but it might also encrypt to 11011110 under a different key, depending on the design of the block cipher.

3.2 Types of Attack

Given the definition of a block cipher, the definition of a secure block cipher seems simple enough: it is a block cipher that keeps the plaintext secret. Although this certainly is one of the requirements, it is not sufficient. This definition only requires that the block cipher be secure against ciphertext-only attacks, in which the attacker gets to see only the ciphertext of a message. There are a few published attacks of this type [74, 121], but they are rare against well-known and established block ciphers. Most published attacks are of the chosen-plaintext type. (See Section 2.6 for an overview of attack types.) All of

these attack types apply to block ciphers, and there are a few more that are specific to block ciphers.

The first one is the related-key attack. First introduced by Eli Biham in 1993 [13], a related-key attack assumes that the attacker has access to several encryption functions. These functions all have an unknown key, but their keys have a relationship that the attacker knows. This sounds very strange, but it turns out that this type of attack is useful against real systems [70]. There are real-world systems that use different keys with a known relationship. At least one proprietary system changes the key for every message by incrementing the key by one. Consecutive messages are therefore encrypted with consecutively numbered keys. It turns out that key relationships like this can be used to attack some block ciphers.

There are even more esoteric attack types. When we designed the Twofish block cipher (Section 3.5.4), we introduced the concept of a chosen-key attack, in which the attacker specifies some part of the key and then performs a related-key attack on the rest of the key [115].[1]

Why would we even consider far-fetched attack types like related-key attacks and chosen-key attacks? We have several reasons. First, we have seen actual systems in which a related-key attack on the block cipher was possible, so these attacks are not that far-fetched at all. In fact, we have even seen standardized protocols that required implementations to key a block cipher with two related keys—one key K that is chosen at random and another key K' that is equal to K plus a fixed constant.

Second, block ciphers are very useful building blocks. But, as building blocks, they tend to get abused in every imaginable way. One standard technique of constructing a hash function from a block cipher is the Davies-Meyer construction [128]. In a Davies-Meyer hash function, the attacker suddenly gets to choose the key of the block cipher, which allows related-key and chosen-key attacks. We talk about hash functions in Chapter 5, but won't go into the details of the Davies-Meyer construction in this book. It is safe to say, however, that any definition of block-cipher security that ignores these attack types, or any other attack type, is incomplete.

The block cipher is a module that should have a simple interface. The simplest interface is to ensure that it has all the properties that anyone could reasonably expect the block cipher to have. Allowing imperfections in the block cipher just adds a lot of complexity, in the form of cross-dependencies, to any system using the cipher. In short, we want to over-engineer our block ciphers for security. The challenge is to define the properties that one reasonably expects from a block cipher.

[1]Later analysis showed that this attack does not work on Twofish [50], but it might be successful against other block ciphers.

3.3 The Ideal Block Cipher

It is actually very hard to define what a block cipher is. It is something that you know when you see it—but can't quite define. The theoretical community has crystallized some of these properties into specific definitions, like pseudorandomness and super-pseudorandomness [6, 86, 94]. The block cipher community itself, however, uses a much broader definition, covering things like weak keys and chosen-key attacks. Here we take the approach of trying to help you understand what the block cipher primitives community believes a block cipher to be. We call this an ''ideal'' block cipher.

What would the ideal block cipher look like? It should be a random permutation. We should be more precise: for each key value, we want the block cipher to be a random permutation, and the different permutations for the different key values should be chosen independently. As we mentioned in Section 3.1, you can think of a 128-bit block cipher (a single permutation on 128-bit values) as a huge lookup table of 2^{128} elements of 128 bits each. The ideal block cipher consists of one of these tables for each key value, with each table chosen randomly from the set of all possible permutations.

Strictly speaking, this definition of the ideal block cipher is incomplete, as the exact choice of the tables has not been specified. As soon as we specify the tables, however, the ideal cipher is fixed and no longer random. To formalize the definition, we cannot talk about a single ideal block cipher, but have to treat the ideal block cipher as a uniform probability distribution over the set of all possible block ciphers. Any time that you use the ideal block cipher, you will have to talk in terms of probabilities. This is a mathematician's delight, but the added complexity would make our explanations far more complicated—so we will keep the informal but simpler concept of a randomly chosen block cipher. We also stress that an ideal block cipher is not something that can be obtained in practice; it is an abstract concept that we use when discussing security.

3.4 Definition of Block Cipher Security

As noted above, there are formal definitions of security for block ciphers in the literature. For our purposes we can use a simpler but informal definition.

Definition 1 *A secure block cipher is one for which no attack exists.*

This is a bit of a tautology. So now we have to define an attack on a block cipher.

Definition 2 *An attack on a block cipher is a non-generic method of distinguishing the block cipher from an ideal block cipher.*

What do we mean by distinguishing a block cipher from an ideal block cipher? Given a block cipher X, we compare it to an ideal block cipher with the same block size and the same key size. A *distinguisher* is an algorithm that is given a black-box function that computes either the block cipher X or an ideal block cipher. (A black-box function is a function that can be evaluated, but the distinguisher algorithm does not know the internal workings of the function in the black box.) Both the encryption and decryption functions are available, and the distinguisher algorithm is free to choose any key for each of the encryptions and decryptions it performs. The distinguisher's task is to figure out whether the black-box function implements the block cipher X or the ideal cipher. It doesn't have to be a perfect distinguisher, as long as it provides the correct answer significantly more often than the wrong answer.

There are, of course, generic (and trivial) solutions to this. We could encrypt the plaintext 0 with the key 0 and see if the result matches what we expect to get from block cipher X. This is a distinguisher, but to make it an attack, the distinguisher has to be non-generic. This is where it becomes difficult to define block cipher security. We cannot formalize the notion of ''generic'' and ''non-generic.'' It is a bit like obscenity: we know it when we see it.[2] A distinguisher is generic if we can find a similar distinguisher for almost any block cipher. In the above case, the distinguisher is generic because we can construct one just like it for any block cipher. This ''attack'' would even allow us to distinguish between two ideal block ciphers. Of course, there's no practical reason for wanting to distinguish between two ideal block ciphers. Rather, this attack is generic because we could use it to distinguish between two ideal block ciphers if we wanted to. The attack doesn't exploit any internal property of the block cipher itself.

We can also create a more advanced generic distinguisher. Encrypt the plaintext 0 with all keys in the range $1, \ldots, 2^{32}$ and count how often each value for the first 32 bits of the ciphertext occurs. Suppose we find that for a cipher X the value t occurs 5 times instead of the expected one time. This is a property that is unlikely to hold for the ideal cipher, and would allow us to distinguish X from an ideal cipher. This is still a generic distinguisher, as we can easily construct something similar for any cipher X. (It is in fact extremely unlikely that a cipher does not have a suitable value for t.) This attack is generic since, the way it is described, it is applicable to all block ciphers and doesn't exploit a specific weakness of X. Such a distinguisher would even allow us to distinguish between two ideal ciphers.

Things become more complicated if we design a distinguisher as follows: We make a list of 1000 different statistics that we can compute about a cipher. We compute each of these for cipher X, and build the distinguisher from the

[2]In 1964, U.S. Supreme Court judge Potter Stewart used these words to define obscenity: ''I shall not today attempt further to define the kinds of material . . . but I know it when I see it.''

statistic that gives the most significant result. We expect to find a statistic with a significance level of about 1 in 1000. We can of course apply the same technique to find distinguishers for any particular cipher, so this is a generic attack, but the generic nature now depends not only on the distinguisher itself, but also on how the distinguisher was found. That's why nobody has been able to formalize a definition of generic attacks and block cipher security. We would love to give you a clean definition of block cipher security, but the cryptographic community does not yet know enough about cryptography to be able to do this in full generality. Instead, existing formal definitions often limit the capability of an attacker. For example, existing formal definitions might not allow chosen-key attacks. While these assumptions can hold in some cases, we try to build block ciphers that are much stronger.

We must not forget to limit the amount of computation allowed in the distinguisher. We could have done this explicitly in the definition, but that would have complicated it even further. If the block cipher has an explicit security level of n bits, then a successful distinguisher should be more efficient than an exhaustive search on n-bit values. If no explicit design strength is given, the design strength equals the key size. This formulation is rather roundabout for a reason. It is tempting to just say that the distinguisher has to work in less than 2^n steps. This is certainly true, but some types of distinguishers give you only a probabilistic result that is more like a partial key search. The attack could have a trade-off between the amount of work and the probability of distinguishing the cipher from the ideal cipher. For example: an exhaustive search of half the key space requires 2^{n-1} work and provides the right answer 75% of the time. (If the attacker finds the key, he knows the answer. If he doesn't find the key, he still has a 50% chance of guessing right simply by guessing at random. Overall, his chances of getting the right answer are therefore $0.5 + 0.5 \cdot 0.5 = 0.75$.) By comparing the distinguisher to such partial key-space searches, we take this natural trade-off into account, and stop such partial key searches from being classified as an attack.

Our definition of block cipher security covers all possible forms of attack. Ciphertext only, known plaintext, (adaptively) chosen plaintext, related key, and all other types of attack all implement a non-generic distinguisher. That is why we will use this informal definition in this book. It also captures the essence of professional paranoia that we talked about in Chapter 1; we want to capture anything that could possibly be considered a non-generic attack.

So why spend multiple pages on defining what a secure block cipher is? This definition is very important because it defines a simple and clean interface between the block cipher and the rest of the system. This sort of modularization is a hallmark of good design. In security systems, where complexity is one of our main enemies, good modularization is even more important than in most other areas. Once a block cipher satisfies our security definition, you can treat it as if it were an ideal cipher. After all, if it does not behave as an ideal cipher

in the system, then you have found a distinguisher for the cipher, which means the cipher is not secure according to our definition. If you use a secure block cipher, you no longer have to remember any particularities or imperfections; the cipher will have all the properties that you expect a block cipher to have. This makes the design of larger systems easier.

Of course, some ciphers that don't meet our stringent definition might be ''good enough'' in practice or for a specific application as currently defined, but why take the risk? Even if the weaknesses of a particular block cipher under our definition are highly theoretical—such as requiring an unrealistic amount of work to exploit and thus not being very vulnerable to compromise in practice—a block cipher that meets our definition is much more attractive.

3.4.1 Parity of a Permutation

Unfortunately, we have one more complication. As we discussed in Section 3.1, encryption under a single key corresponds to a lookup in a permutation table. Think about constructing this table in two steps. First you initialize the table with the identity mapping by giving the element at index i the value i. Then you create the permutation that you want by repeatedly swapping two elements in the table. It turns out there are two types of permutations: those that can be constructed from an even number of swaps (called the even permutations) and those that can be constructed from an odd number of swaps (called the odd permutations). It should not surprise you that half of all permutations are even, and the other half are odd.

Most modern block ciphers have a 128-bit block size, but they operate on 32-bit words. They build the encryption function from many 32-bit operations. This has proved to be a very successful method, but it has one side effect. It is rather hard to build an odd permutation from small operations; as a result, virtually all block ciphers only generate even permutations.

This gives us a simple distinguisher for nearly any block cipher, one which we call the *parity attack*. For a given key, extract the permutation by encrypting all possible plaintexts. If the permutation is odd, we know that we have an ideal block cipher, because the real block cipher never generates an odd permutation. If the permutation is even, we claim to have a real block cipher. This distinguisher will be right 75% of the time. It will produce the wrong answer only if it is given an ideal cipher that produces an even permutation. The success rate can be improved by repeating the work for other key values.

This attack has no practical significance whatsoever. To find the parity of a permutation, you have to compute all but one of the plaintext/ciphertext pairs of the encryption function. (The last one is trivial to deduce: the sole remaining plaintext maps to the sole remaining ciphertext.) You should never allow that many plaintext/ciphertext queries to a block cipher in a real system, because other types of attacks start to hurt much sooner. In particular,

once the attacker knows most of the plaintext/ciphertext pairs, he no longer needs a key to decrypt the message, but can simply use a lookup table created from those pairs.

We could declare the parity attack to be generic by definition, but that seems disingenuous, since the even parity of block ciphers is a peculiar artifact of their designs. Rather, we prefer to change the definition of the ideal block cipher, and limit it to randomly chosen *even* permutations.

Definition 3 *An ideal block cipher implements an independently chosen random even permutation for each of the key values.*

It is a pity to complicate our ''ideal'' cipher in this way, but the only alternative is to disqualify nearly all known block ciphers. For the overwhelming majority of applications, the restriction to even permutations is insignificant. As long as we never allow all plaintext/ciphertext pairs to be computed, even and odd permutations are indistinguishable.

If you ever have a block cipher that *can* generate odd permutations, you should revert to the original definition of the ideal cipher. In practice, parity attacks have more effect on the formal definition of security than on real-world systems, so you can probably forget about this whole issue of parity.

This discussion also serves as another example of how cryptographers think. It is more important to exhibit professional paranoia and consider a superset of realistic attacks, and then pare away the unrealistic ones, than to start with only realistic attacks and try to find new ones.

3.5 Real Block Ciphers

There are hundreds of block ciphers that have been proposed over the years. It is very easy to design a new block cipher. It is fiendishly hard to design a *good* new block cipher. We're not merely talking about security; that a block cipher has to be secure goes without saying. Building a secure block cipher is a challenge unto itself. But it becomes even more difficult to create a block cipher that is efficient in a wide variety of different applications. (We previously said that we'd give up performance for security. We would. But when possible, we still prefer both.)

Designing block ciphers can be fun and educational, but one shouldn't use an unknown cipher in a real system. The cryptographic community doesn't trust a cipher until it has been reviewed thoroughly by other experts. A basic prerequisite is that the cipher has been published, but this is not enough. There are so many ciphers out there that few get any effective peer review. You are much better off using one of the well-known ciphers that already has been reviewed for you.

Virtually all block ciphers consist of several repetitions of a weak block cipher, known as a *round*. Several of these weak rounds in sequence make a

strong block cipher. This structure is easy to design and implement, and is also a great help in the analysis. Most attacks on block ciphers begin by attacking versions with a reduced number of rounds. As the attacks improve, more and more rounds can be attacked.

We will discuss several block ciphers in more detail, but we won't define them exhaustively. The full specifications can be found in the references or on the Internet. We will instead concentrate on the overall structure and the properties of each cipher.

3.5.1 DES

The venerable workhorse of cryptography, the Data Encryption Standard (DES) [96] has finally outlived its usefulness. Its restricted key size of 56 bits and small block size of 64 bits make it unsuitable for today's fast computers and large amounts of data. It survives in the form of 3DES [99], which is a block cipher built from three DES encryptions in sequence—encrypt with DES with one 56-bit key, decrypt with a second 56-bit key, and then encrypt again either with the first key or a third 56-bit key. This solves the most immediate problem of the small key size, but there is no known fix for the small block size. DES is not a particularly fast cipher by current standards, and 3DES is one-third the speed of DES. You will still find DES in many systems, but we do not recommend using either DES or 3DES in new designs. It is, however, a classic design worth studying in its own right.

Figure 3.1 gives an overview of a single round of DES. This is a line diagram of the DES computations; you will commonly find diagrams like this in cryptographic literature. Each box computes a particular function, and the lines show which value is used where. There are a few standard conventions. The XOR or exclusive-or operation, sometimes called bitwise addition or addition without carry, is shown in formulas as a $\oplus$ operator and in figures as a large version of the $\oplus$ operator. You might also find drawings that include integer additions, which often are drawn to look like the $\boxplus$ operator.

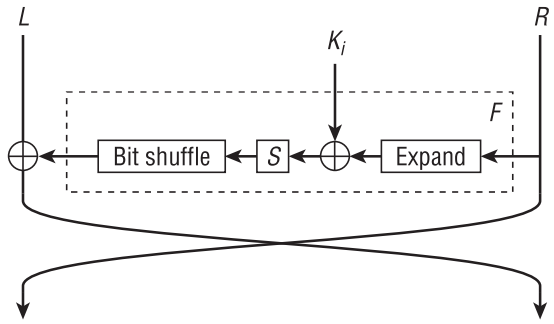

Figure 3.1: Structure of a single round of DES

DES has a 64-bit plaintext, which is split into two 32-bit halves L and R. This splitting is done by rearranging the bits in a semi-ordered fashion. Nobody seems to know why the designers bothered to rearrange the bits of the plaintext—it has no cryptographic effect—but that's how DES is defined. A similar swapping of bits is implemented at the end of the encryption to create the 64-bit ciphertext from the two halves L and R.

DES consists of 16 rounds numbered 1 through 16. Each round i uses a separate 48-bit round key K_i. Each round key is formed by selecting 48 bits from the 56-bit key, and this selection is different for each round key.[3] The algorithm that derives these round keys from the main block cipher key is called the key schedule.

Round i transforms the (L, R) pair into a new (L, R) pair under control of a round key K_i. Most of the work is done by the round function F, shown in the dashed box. As shown in the figure, the R value is first processed by an expand function, which duplicates a number of bits to produce 48 bits of output from the 32-bit input. The 48-bit result is XORed with the 48-bit round key K_i. The result of this is used in the S-box tables. An S-box (the term derives from *substitution box*) is basically just a lookup table that is publicly known. As you cannot build a lookup table with 48 input bits, the S-boxes consist of eight small lookup tables, each of which maps 6 bits to 4 bits. This brings the result size back to 32 bits. These 32 bits are then swapped around by the bit shuffle function before being XORed into the left value L. Finally, the values of L and R are swapped. This entire computation is repeated 16 times for a single DES encryption.

The basic structure of DES is called the Feistel construction [47]. It is a really elegant idea. Each round consists of XORing L with $F(K_i, R)$ for some function F, and then swapping L and R. The beauty of the construction is that decryption requires exactly the same set of operations as encryption. You need to swap L and R, and you need to XOR L with $F(K_i, R)$. This makes it much easier to implement the encryption and decryption functions together. It also means that you only have to analyze one of the two functions, as they are almost identical. A final trick used in most Feistel ciphers is to leave out the swap after the last round, which makes the encryption and decryption functions identical except for the order of the round keys. This is particularly nice for hardware implementations, as they can use the same circuit to compute both encryptions and decryptions.

The different parts of the DES cipher have different functions. The Feistel structure makes the cipher design simpler and ensures that the two halves L and R are mixed together. XORing the key material ensures that the key and data are mixed, which is the whole point of a cipher. The S-boxes provide nonlinearity. Without them, the cipher could be written as a bunch of binary additions, which would allow a very easy mathematical attack based on linear

[3]There is some structure to this selection, which you can find in the DES specifications [96].

algebra. Finally, the combination of the S-box, expand, and bit shuffle functions provide diffusion. They ensure that if one bit is changed in the input of F, more than one bit is changed in the output. In the next round there will be more bit changes, and even more in the round after that, etc. Without good diffusion, a small change in the plaintext would lead to a small change in the ciphertext, which would be very easy to detect.

DES has a number of properties that disqualify it according to our security definition. Each of the round keys consists purely of some of the bits selected from the cipher key. If the cipher key is 0, then all the round keys are 0 as well. In particular, all the round keys are identical. Remember that the only difference between encryption and decryption is the order of the round keys. But all round keys are zero here. So encryption with the 0 key is the same function as decryption with the 0 key. This is a very easy property to detect, and as an ideal block cipher does not have this property, it leads to an easy and efficient distinguishing attack.[4]

DES also has a complementation property that ensures that

$$E(\overline{K}, \overline{P}) = \overline{E(K, P)}$$

for all keys K and plaintexts P, where $\overline{X}$ is the value obtained by complementing all the bits in X. In other words, if you encrypt the complement of the plaintext with the complement of the key, you get the complement of the (original) ciphertext.

This is rather easy to see. Look at the figure and think about what happens if you flip all the bits in L, R, and K_i. The expand function merely copies bits around, so all the output bits are also flipped. The XOR with the key K_i has both inputs flipped, so the output remains the same. The input to the S-boxes remains the same, the output of the S-boxes remains the same, so the final XOR has one input that is flipped and one input that is the same. The new L value, soon to be swapped to the R position, is therefore also flipped. In other words, if you complement L and R at the beginning of the round and complement K_i as well, then the output is the complement of what you had originally. This property passes through the entire cipher.

The ideal block cipher would not have this curious property. More importantly, this particular property can lead to attacks on systems that use DES.

In short, DES does not pass muster anymore. The above properties disqualify DES according to our security definition. But even ignoring the properties above, the DES key length is wholly inadequate. There have already been several successful attempts to find a DES key by simple exhaustive search [44].

3DES has a larger key, but it inherits both the weak keys and the complementation property from DES, each of which is enough to disqualify the

[4]There are three other keys that have this property; together, they are called the weak keys of DES.

cipher by our standards. It is also severely limited by its 64-bit block size, which imposes severe restrictions on the amount of data we can encrypt with a single key. (See Section 4.8 for details.) Sometimes you have to use 3DES in a design for legacy reasons, but be very careful with it because of its small block size and because it does not behave like an ideal block cipher.

3.5.2 AES

The Advanced Encryption Standard (AES) is the U.S. government standard created to replace DES. Instead of designing or commissioning a cipher, the U.S. National Institute of Standards and Technology (NIST) asked for proposals from the cryptographic community. A total of 15 proposals were submitted [98]. Five ciphers were selected as finalists [100], after which Rijndael was selected to become AES.[5] AES became a standard in 2001.

AES uses a different structure than DES. It is not a Feistel cipher. Figure 3.2 shows a single round of AES. The subsequent rounds are similar. The plaintext comes in as 16 bytes (128 bits) at the very top. The first operation is to XOR the plaintext with 16 bytes of round key. This is shown by the ⊕ operators; the key bytes come into the side of the XORs. Each of the 16 bytes is then used as an index into an S-box table that maps 8-bit inputs to 8-bit outputs. The S-boxes are all identical. The bytes are then rearranged in a specific order that looks a bit messy but has a simple structure. Finally, the bytes are mixed in groups of four using a linear mixing function. The term linear just means that each output bit of the mixing function is the XOR of several of the input bits.

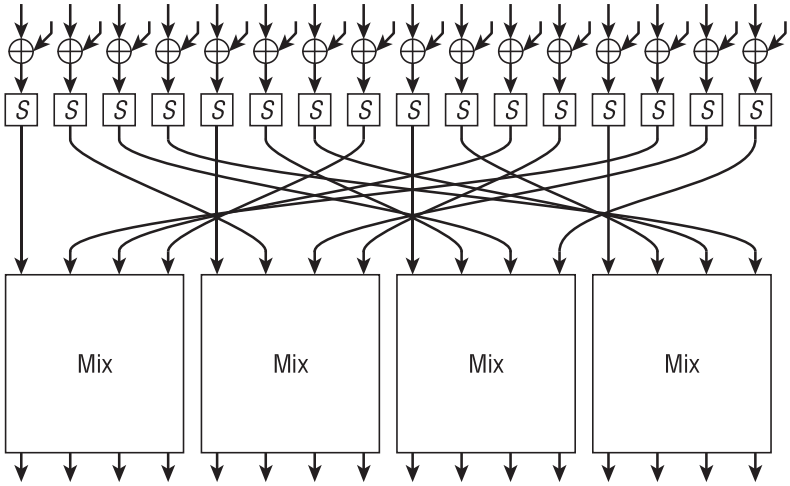

Figure 3.2: Structure of a single round of AES

[5]There has been some confusion about the correct pronunciation of ''Rijndael.'' Don't worry; it's hard to pronounce unless you speak Dutch, so just relax and pronounce it any way you like, or just call it ''AES.''

This completes a single round. A full encryption consists of 10–14 rounds, depending on the size of the key. AES is defined for 128-, 192-, and 256-bit keys, and uses 10 rounds for 128-bit keys, 12 rounds for 192-bit keys, and 14 rounds for 256-bit keys. Like DES, there is a key schedule that generates the necessary round keys, but the key schedule uses a very different structure.

The AES structure has advantages and disadvantages. Each step consists of a number of operations that can be performed in parallel. This parallelism makes high-speed implementations easy. On the other hand, the decryption operation is significantly different from the encryption operation. You need the inverse lookup table of the S-box, and the inverse mixing operation is different from the original mixing operation.

We can recognize some of the same functional blocks as in DES. The XORs add key material to the data, the S-boxes provide nonlinearity, and the byte shuffle and mixing functions provide diffusion. AES is a very clean design with clearly separated tasks for each part of the cipher.

AES has always been a fairly aggressively designed cipher. In the original presentation, the AES designers showed an attack on 6 rounds. This means that the designers knew of an attack if AES was defined to have only 6 rounds. The authors therefore chose 10–14 rounds for the full cipher, depending on the key size [27].

During the AES selection process, the attacks were improved to handle 7 rounds for 128-bit keys, 8 rounds for 192-bit keys, and 9 rounds for 256-bit keys [49]. This still left a 3 to 5 round security margin. From a different perspective: for 128-bit keys, the best attack we knew when Rijndael was selected as AES covered 70% of the cipher. In other words, the selection of Rijndael as AES relied on the assumption that future attacks would not give large improvements.

Will AES stand the test of time? It is, as always, impossible to predict the future, but sometimes it helps to look at the past. Until recently, the best-analyzed ciphers were DES, FEAL, and IDEA. In all cases, the attacks were significantly improved many years after the initial publication. Since then, the field has progressed, but it still takes a leap of faith to think we know it all and that no significant improvements in attacks will be found.

In fact, at the time of this writing we are starting to see some pretty amazing breakthroughs in the cryptanalysis of AES [14, 15, 16]. One attack can break the full 12 rounds of AES with 192-bit keys using four related keys and 2^{176} operations, and another attack can break the full 14 rounds of AES with 256-bit keys using four related keys and 2^{119} operations [15]. Another attack can break 10 of the 14 rounds of AES with 256-bit keys using two related keys and only 2^{45} operations [14].

These are huge results. They mean we now know AES does not meet our definition of security for a block cipher. The attacks against the full 192- and 256-bit versions of AES are theoretical—not practical—so we aren't ready to

lose any sleep over them just yet. But they are attacks under our definition, so 192- and 256-bit AES have theoretically been broken. And even better attacks might be discovered over time.

The community is still trying to come to grips with what these results mean for the use of AES in a real system. Given all that we know today, using AES still seems like a reasonable decision. It is the U.S. government standard, which means a great deal. Using the standard avoids a number of discussions and problems. But it is important to realize it is still possible that future cryptanalytic advances may uncover even more serious weaknesses. If you are developing a system or standardizing a protocol, we recommend building in some flexibility or extensibility in case you need to replace AES with another block cipher in the future. We will come back to this in Section 3.5.6.

3.5.3 Serpent

Serpent was another AES finalist [1]. It is built like a tank. Easily the most conservative of all the AES submissions, Serpent is in many ways the opposite of AES. Whereas AES puts emphasis on elegance and efficiency, Serpent is designed for security all the way. The best attack we know of covers only 12 of the 32 rounds [38]. The disadvantage of Serpent is that it is about one-third the speed of AES. It can also be difficult to implement efficiently, as the S-boxes have to be converted to a Boolean formula suitable for the underlying CPU.

In some ways, Serpent has a similar structure to AES. It consists of 32 rounds. Each round consists of XORing in a 128-bit round key, applying a linear mixing function to the 128 bits, and then applying 32 four-bit S-boxes in parallel. In each round, the 32 S-boxes are identical, but there are eight different S-boxes that are used each in turn in a round.

Serpent has an especially nice software implementation trick. A straightforward implementation would be very slow, as each round requires 32 S-box lookups and there are 32 rounds. In total there are 1024 S-box lookups, and doing those one by one would be very slow. The trick is to rewrite the S-boxes as Boolean formulas. Each of the four output bits is written as a Boolean formula of the four input bits. The CPU then evaluates this Boolean formula directly, using AND, OR, and XOR instructions. The trick is that a 32-bit CPU can evaluate 32 S-boxes in parallel, as each bit position in the registers computes the same function, albeit on different input data. This style of implementation is called a *bitslice implementation*. Serpent is specifically designed to be implemented in this way. The mixing phase is relatively easy to compute in a bitslice implementation.

If Serpent had been as fast as Rijndael (now AES), it would almost certainly have been chosen as AES because of its conservative design. But speed is

always a relative thing. When measured per encrypted byte, Serpent is nearly as fast as DES and much faster than 3DES. It is only when Serpent is compared to the other AES finalists that it seems slow.

3.5.4 Twofish

Twofish was an AES finalist as well. It can be seen as a compromise between AES and Serpent. It is nearly as fast as AES, but it has a larger security margin. The best attack we know of is on 8 of the 16 rounds. The biggest disadvantage of Twofish is that it can be rather expensive to change the encryption key, as Twofish is best implemented with a lot of precomputation on the key.

Twofish uses the same Feistel structure as DES. An overview is given in Figure 3.3.[6] Twofish splits the 128-bit plaintext into four 32-bit values, and most operations are on 32-bit values. You can see the Feistel structure of Twofish, with F being the round function. The round function consists of two copies of the g function, a function called the PHT, and a key addition. The result of the F function is XORed into the right half (the two vertical lines on the right). The boxes with $\lll$ or $\ggg$ symbols in them denote rotations of the 32-bit value by the specified number of bit positions.

Each g function consists of four S-boxes followed by a linear mixing function that is very similar to the AES mixing function. The S-boxes are somewhat different. In contrast to all the other block ciphers we have seen in this book, these S-boxes are not constant; rather, their contents depend on the key. There is an algorithm that computes the S-box tables from the key material. The motivation for this design is that key-dependent S-boxes are much harder for an attacker to analyze. This is also why Twofish implementations often do precomputations for each key. They precompute the S-boxes and store the result in memory.

The PHT function mixes the two results of the g functions using 32-bit addition operations. The last part of the F function is where the key material is added. Note that addition is shown as $\boxplus$ and exclusive or as $\oplus$.

Twofish also uses *whitening*. At both the start and the end of the cipher, additional key material is added to the data. This makes the cipher harder to attack for most types of attacks, and it costs very little.

As with the other ciphers, Twofish has a key schedule to derive the round keys and the two additional keys at the start and end from the actual cipher key.

[6]There is a reason why this figure is so much larger and detailed than the others. Two of us were on the Twofish design team, so we could lift this figure straight from our Twofish book [115].

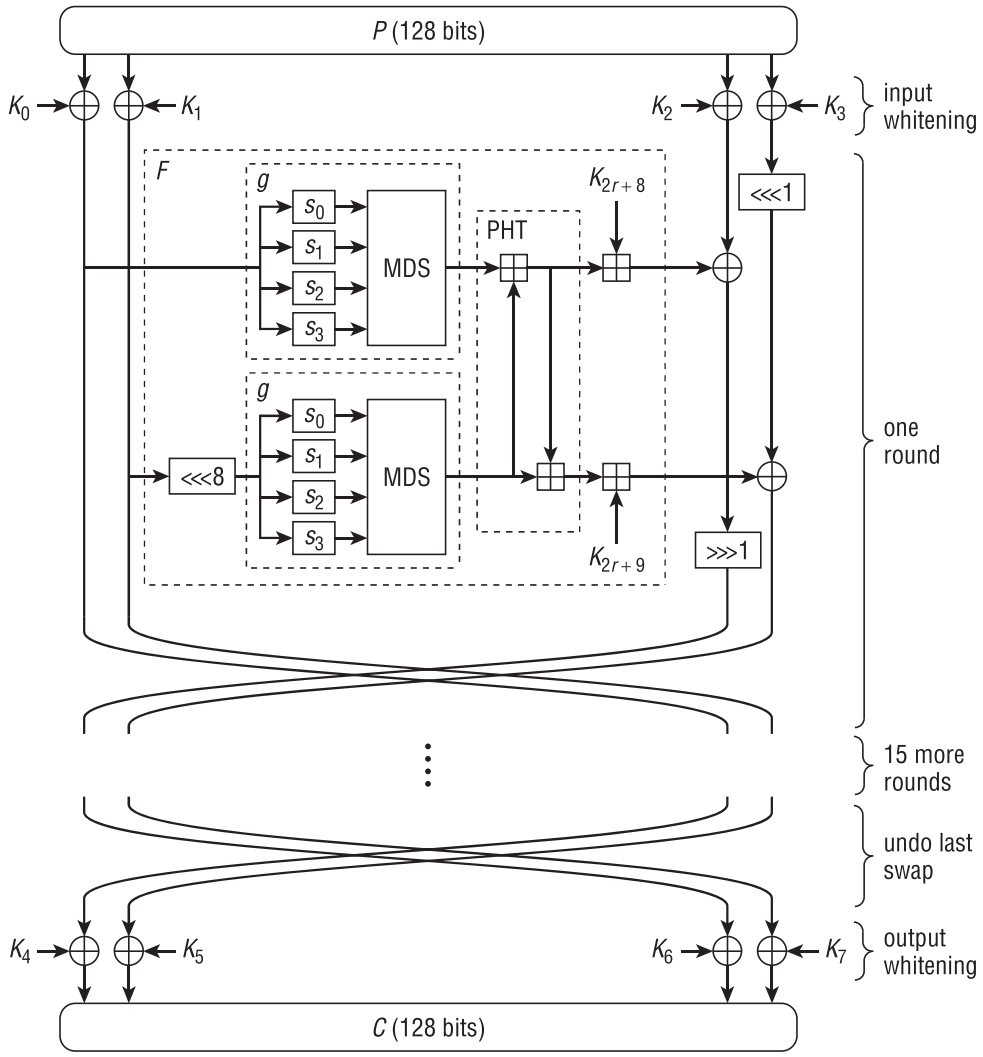

Figure 3.3: Structure of Twofish
© 1999, Niels Ferguson, John Wiley and Sons. Used with permission.

3.5.5 Other AES Finalists

We have discussed three of the five AES finalists in some detail. There were two more: RC6 [108] and MARS [22].

RC6 is an interesting design that uses 32-bit multiplications in the cipher. During the AES competition, the best attack broke a 17-round version of RC6, compared to 20 rounds of the full RC6. MARS is a design with a nonuniform structure. It uses a large number of different operations and is therefore more expensive to implement than the other AES finalists.

Both RC6 and MARS were selected as AES finalists for a reason: They are both probably good block ciphers. Details about their internal operations are in their respective specifications.

3.5.6 Which Block Cipher Should I Choose?

The recent cryptanalytic advances against AES make this a tough choice. Despite these cryptanalytic advances, AES is still what we recommend. It is fast. All known attacks are theoretical, not practical. Even though AES is now broken academically, these breaks do not imply a significant security degradation of real systems in practice. It is also the official standard, sanctioned by the U.S. government. And everybody else is using it. They used to say ''Nobody gets fired for buying IBM.'' Similarly, nobody will fire you for choosing AES.

AES has other advantages. It is relatively easy to use and implement. All cryptography libraries support it, and customers like it, because it is ''the standard.''

There are probably circumstances in which 3DES still is the best solution. If you have to be backward-compatible, or are locked into a 64-bit block size by other parts of the system, then 3DES is still your best choice. However, keep in mind that 3DES has some unique properties that cause it to not satisfy our security criteria; and be especially careful with the small 64-bit block size.

If you are really concerned about future cryptanalytic advances, you could always double encrypt—first with AES and then with Serpent or Twofish. If you do this, remember to use different, independent keys for each block cipher. Or use AES with an increased number of rounds—say, 16 rounds for AES with 128-bit keys, 20 rounds for AES with 192-bit keys, and 28 rounds for AES with 256-bit keys.

Further, remember that the recent cryptanalytic advances against AES are only just coming out as we finalize this book. It is too early to tell exactly how the community will respond. Keep an eye out for a general consensus or shift in direction from the community. Perhaps NIST will issue some specific recommendations for how to address the recent discoveries against AES. If NIST makes recommendations on how to respond to the new attacks against AES, or if there is a clear shift in the consensus of the community, no one will fault you for following those recommendations or that shift.

We also need to return to one other issue with AES. We haven't talked much about side-channel or timing attacks yet (we'll talk about these in Sections 8.5 and 15.3). It turns out that even though there are no known practical attacks against the mathematics of AES, it is possible to implement AES poorly. For example, it is possible to implement AES such that the time it takes to perform an operation depends on its inputs—on some inputs it will take more time and on other inputs it will take less time. If an attacker can measure the time a system takes to perform an AES operation, she might be able to learn bits of the key. If you use AES, you should be careful to use a constant-time implementation, or to otherwise conceal the timing information from an attacker.

3.5.7 What Key Size Should I Use?

All of the AES finalists (Rijndael, Serpent, Twofish, RC6, and MARS), and hence AES, support keys of 128, 192, and 256 bits. For almost all applications, a 128-bit security level is enough. However, to achieve 128 bits of security, we suggest keys longer than 128 bits.

A 128-bit key would be great, except for one problem: collision attacks. Time and time again, we find systems that can be attacked—at least theoretically, if not practically—by a birthday attack or a meet-in-the-middle attack. We know these attacks exist. Sometimes designers just ignore them, and sometimes they think they are safe, but somebody finds a new, clever way of using them. Most block cipher modes allow meet-in-the-middle attacks of some form. We've had enough of this race, so here is our recommendation: For a security level of n bits, every cryptographic value should be at least $2n$ bits long.

Following this recommendation makes any type of collision attack useless. In real life, it is hard to keep strictly to this rule. For 128-bit security, we really want to use a block cipher with a block size of 256 bits, but all the common block ciphers have a block size of 128 bits. This is more serious than it sounds. There are quite a number of collision attacks on block cipher modes, which we will learn about later.

Still, at least we can use the large keys that all AES candidate block ciphers support. Therefore: use 256-bit keys! We are not saying that 128-bit keys are insecure per se; we are saying that 256-bit keys provide a better safety margin, assuming that the block cipher is secure.

Note that we advocate the use of 256-bit keys for systems with a design strength of 128 bits. In other words, these systems are designed to withstand attackers that can perform 2^{128} operations in their attack. Just remember to use the design strength (128 bits), not the key length of 256 bits, for sizing the rest of the system.

Finally, let's come back to the recent cryptanalytic results against AES. These results show that AES with 192- and 256-bit keys are not secure. Moreover, the attacks against AES with 192- and 256-bit keys exploit weaknesses in the AES key schedule algorithm. This is why the known attacks against AES with 256-bit keys are more efficient than the attacks against AES with 192-bit keys. This is also why we don't yet know of attacks against AES with 128-bit keys. So, while in general we'd prefer a block cipher with 256-bit keys over a block cipher with 128-bit keys, assuming the block cipher is secure, the situation is a bit more murky for AES. To emphasize our desire for 128 bits of security, and thus our quest for a secure block cipher with 256-bit keys, we will use AES with 256-bit keys throughout the rest of this book. But once there is a clear consensus of how to respond to the new cryptanalytic results against AES, we will likely replace AES with another block cipher with 256-bit keys.

3.6 Exercises

Exercise 3.1 How much space would be required to store a table for an entire idealized block cipher that operates on 64-bit blocks and that has 80-bit keys?

Exercise 3.2 How many rounds are in DES? How many bits are in a DES key? What is the DES block size? How does 3DES work as a function of DES?

Exercise 3.3 What are the possible lengths for AES keys? For each key length, how many rounds are in AES? What is the AES block size?

Exercise 3.4 Under what situations might you choose 3DES over AES? Under what situations might you chose AES over 3DES?

Exercise 3.5 Suppose you have a processor that can perform a single DES encryption or decryption operation in 2^{-26} seconds. Suppose you also have a large number of plaintext-ciphertext pairs for DES under a single, unknown key. How many hours would it take, on average, to find that DES key, using an exhaustive search approach and a single processor? How many hours would it take, on average, to find that DES key, using an exhaustive search approach and a collection of 2^{14} processors?

Exercise 3.6 Consider a new block cipher, DES2, that consists only of two rounds of the DES block cipher. DES2 has the same block and key size as DES. For this question you should consider the DES *F* function as a black box that takes two inputs, a 32-bit data segment and a 48-bit round key, and that produces a 32-bit output.

Suppose you have a large number of plaintext-ciphertext pairs for DES2 under a single, unknown key. Give an algorithm for recovering the 48-bit round key for round 1 and the 48-bit round key for round 2. Your algorithm should require fewer operations than an exhaustive search for an entire 56-bit DES key. Can your algorithm be converted into a distinguishing attack against DES2?

Exercise 3.7 Describe an example system that uses DES but is insecure because of the DES complementation property. Specifically, describe the system, and then present an attack against that system; the attack should utilize the DES complementation property.

Exercise 3.8 Familiarize yourself with a cryptographic software development package for your computer. A popular open source package is OpenSSL, though there are numerous other alternatives.

Using an existing cryptography library, decrypt the following ciphertext (in hex)

```
53 9B 33 3B 39 70 6D 14 90 28 CF E1 D9 D4 A4 07
```

with the following 256-bit key (also in hex)

```
80 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 01
```

using AES.

Exercise 3.9 Using an existing cryptography library, encrypt the following plaintext (in hex)

```
29 6C 93 FD F4 99 AA EB 41 94 BA BC 2E 63 56 1D
```

with the following 256-bit key (also in hex)

```
80 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 01
```

using AES.

Exercise 3.10 Write a program that experimentally demonstrates the complementation property for DES. This program should take as input a key K and a plaintext P and demonstrate that the DES complementation property holds for this key and plaintext. You may use an existing cryptography library for this exercise.