Copyright ©1997–2000 by Mihir Bellare (UCSD) and Phillip Rogaway (UCD)

Chapter 2

Block Ciphers

Block ciphers are the central tool in the design of protocols for shared-key cryptography. They are the main available "technology" we have at our disposal. This chapter will take a look at these objects and describe the state of the art in their construction.

It is important to stress that block ciphers are just tools; they don't, by themselves, do something that an end-user would care about. As with any powerful tool, one has to learn to use this one. Even a wonderful block cipher won't give you security if you use don't use it right. But used well, these are powerful tools indeed. Accordingly, an important theme in several upcoming chapters will be on how to use block ciphers well. We won't be emphasizing how to design or analyze block ciphers, as this remains very much an art.

The main purpose of this chapter is just to get you acquainted with what typical block ciphers look like. We'll look at two examples, DES, and RC6. `(Replace RC6 by AES.)` DES is the "old standby." It is currently (year 2000) the most widely-used block cipher in existence, and it is of sufficient historical significance that every trained cryptographer needs to have seen its description. RC6 is a more modern block cipher. It is a possible replacement for DES.

2.1 What is a block cipher?

A block cipher is a function E: $\{0,1\}^k \times \{0,1\}^n \to \{0,1\}^n$ that takes two inputs, a k-bit key K and an n-bit "plaintext" M, to return an n-bit "ciphertext" $C = E(K, M)$. The *key-length* k and the *block-length* n are parameters associated to the block cipher and vary from cipher to cipher, as of course does the design of the algorithm itself. For each key $K \in \{0,1\}^k$ we let $E_K\colon \{0,1\}^n \to \{0,1\}^n$ be the function defined by $E_K(M) = E(K,M)$. For any block cipher, and any key K, the function E_K is a *permutation* on $\{0,1\}^n$. This means that it is a bijection, ie. a one-to-one and onto function of $\{0,1\}^n$ to $\{0,1\}^n$. Accordingly it has an inverse, and we can denote it E_K^{-1}. This function also maps $\{0,1\}^n$ to $\{0,1\}^n$, and of course we have $E_K^{-1}(F_K(M)) = M$ and $E_K(E_K^{-1}(C)) = C$ for all $M, C \in \{0,1\}^n$. We let $E^{-1}\colon \{0,1\}^k \times \{0,1\}^n \to \{0,1\}^n$ be defined by $E^{-1}(K,C) = F_K^{-1}(C)$; this is the inverse cipher to E.

The block cipher is a public and fully specified algorithm. Both the cipher E and its inverse E^{-1} should be easily computable, meaning given K, M we can compute $F(K,M)$, and given K, C we can compute $F^{-1}(K,C)$.

In typical usage, a random key K is chosen and kept secret between a pair of users. The function E_K is then used by the two parties to process data in some way before they send it to each other. The adversary may see input-output examples of E_K, meaning pairs of the form (M,C) where $C = F_K(M)$, but is not directly shown the key K. Security relies on the secrecy of the key. (This is necessary, but not always sufficient, for security!). So at a first cut at least, you might view the adversary's goal as recovering the key K given some input-output examples of E_K. The block cipher should be designed to make this task computationally difficult. Later we will refine this view, but it is the classical one, so let's go with it for now.

How do block ciphers work? Lets take a look at some of them to get a sense of this.

2.2 The Data Encryption Standard (DES)

The Data Encryption Standard (DES) is the quintessential block cipher. Even though it is now feeling its age, and on the way out, no discussion of block ciphers can really omit mention of this construction. DES is a remarkably well-engineered algorithm which has had a powerful influence on cryptography. DES is in very widespread use, and probably will be for some years to come. Every time you use an ATM machine, you are using DES.

A brief history

Beef-up the history a bit. In 1972 the NBS (National Bureau of Standards, now NIST, the National Institute of Science and Technology) initiated a program for data protection and wanted as part of it an encryption algorithm that could be standardized. They put out a request for such an algorithm. In 1974, IBM responded with a design based on their "Lucifer" algorithm. This design would eventually evolve into the DES.

DES has a key-length of $k = 56$ and a block-length of $n = 64$. It consists of 16 rounds of what is called a "Feistel network." We will describe the details shortly.

After NBS, several other bodies adopted DES as a standard, including ANSI (the American National Standards Institute) and the American Bankers Association.

The standard was to be reviewed every five years to see whether or not it should be re-adopted. Although there were claims that it would not be re-certified, the algorithm was re-certified again and again. Only recently did the work for finding a replacement begin in earnest, in the form of the AES (Advanced Encryption Standard) effort.

DES proved remarkably secure. There has, since the beginning, been concerns, especially about exhaustive key-search. But for a fair length of time, the key size of 56 bits was good enough against all but very well-funded organizations. Interesting attacks significantly different from key search emerged only in the nineties, and even then don't break DES in a sense significant in practice. But with today's technology, 56 bits is too small a key size for many security applications.

Construction

The construction is described in FIPS 46 [2]. **Add in the full description, including pretty pictures.**

We refer the reader there. The following discussion is a quick guide that you can follow if you have the FIPS document at your side. (A full version of these notes ought of course to have its own pictures and descriptions so that it is self-contained ... maybe some day ...).

Begin at page 87 where you see a big picture. The input is 64 bits and in addition there is a 56 bit key K. (They say 64, but actually every eighth bit is ignored. It can be set to the parity of the previous seven.) Notice the algorithm is public. You operate with a hidden key, but nothing about the algorithm is hidden.

The first thing the input is hit with is something called the initial permutation, or IP. This just shuffles bit positions. That is, each bit is moved to some other position. How? In a fixed and specified way: see page 88. Similarly, right at the end, notice they apply the inverse of the same permutation. From now on, ignore these. They do not affect security. (As far as anyone can tell, they are there to make loading the chips easier.)

The essence of DES is in the round structure. There are 16 rounds. Each round i has an associated subkey K_i which is 48 bits long. The subkeys $K_1, \ldots, K_{16}$ are derived from the main key K, in a manner explained on page 95 of the FIPS document.

In each round, the input is viewed as a pair (L_i, R_i) of 32 bit blocks, and these are transformed into the new pair (L_{i+1}, R_{i+1}), via a certain function f that depends on a subkey K_i pertaining to round i. The structure of this transformation is important: it is called the Feistel transformation.

The Feistel transformation, in general, is like this. For some function g known to the party computing the transformation, it takes input (L, R) and returns (L', R') where $L' = R$ and $R' = g(R) \oplus L$. A central property of this transformation is that it is a permutation, and moreover if you can compute g then you can also invert the transform. Indeed, given (L', R') we can recover (L, R) via $R = L'$ and $L = g(R) \oplus R'$. For DES, the role of g in round i is played by $f(K_i, \cdot)$, the round function specified by the subkey K_i. Since $\mathrm{DES}_K(\cdot)$ is a sequence of Feistel transforms, each of which is a permutation, the whole algorithm is a permutation, and knowledge of the key K permits computation of $\mathrm{DES}_K^{-1}(\cdot)$.

Up to now the structure has been quite generic, and indeed many block-ciphers use this high level design: a sequence of Feistel rounds. For a closer look we need to see how the function $f(K_i, \cdot)$ works. See the picture on page 90 of the FIPS document. Here K_i is a 48 bit subkey, derived from the 56 bit key in a way depending on the round number. The 32 bit R_i is first expanded into 48 bits. How? In a precise, fixed way, indicated by the table on the same page, saying E-bit selection table. It has 48 entries. Read it as which inputs bits are output. Namely, output bits 32, 1, 2, 3, 4, 5, then 4, 5 again, and so on. It is NOT random looking! In fact barring that 1 and 32 have been swapped (see top left and bottom right) it looks almost sequential. Why did they do this? Who knows. That's the answer to most things about DES.

Now K_i is XORed with the output of the E-box and this 48 bit input enters the famous S-boxes. There are eight S-boxes. Each takes 8 bits to 6 bits. Thus we get out 32 bits. Finally, there is a P-box, a permutation applied to these 32 bits to get another 32 bits. You can see it on page 91.

What are the S-boxes? Each is a fixed, tabulated function, which the algorithm stores as tables in the code or hardware. You can see them on page 93. How to read them? Take the 6 bit input $b_1, b_2, b_3, b_4, b_5, b_6$. Interpret the first two bits as a row number, between 1 and 4. Interpret the rest as a column number, 1 through 16. Now index into the table.

Well, without going into details, the main objective of the above was to give you some idea of the kind of structure DES has. Of course, the main questions about the design are: why, why and why? What motivated these design choices? We don't know too much about this, although we can guess a little.

Speed

How fast can you compute DES? Let's begin by looking at hardware implementations since DES was in fact designed to be fast in hardware. We will be pretty rough; you can find precise estimates, for different architectures, in various places.

You can get over a Gbit/sec throughput using VLSI. Specifically at least 1.6 Gbits/sec, maybe more. That's pretty fast. **I don't think these figures are meaningful. Gate depth is more interesting, or number of gates per S box, which I think is something like 50.**

Some software figures I found quoted are: 12 Mbits/sec on a HP 9000/887; 9.8 Mbits/sec on a DEC Alpha 4000/610. The performance of DES in software is not as good as one would like, and is one reason people seek alternatives. **Again, cycles per byte on a "modern" processor would be more interesting. I think it's something like 80 cpb. Boesaellaers should have the most accurate figures.**

Security

Practical cryptanalysis of DES focuses on key-recovery. They formulate the problem facing the cryptanalyst like this. A 56-bit key K is chosen at random. Let $F_K = \mathrm{DES}_K$. Let q be some parameter.

Given: The adversary has a sequence of q input-output examples of F_K, say $(x_1, y_1), \ldots, (x_q, y_q)$, where $y_i = F_K(x_i)$ for $i = 1, \ldots, q$ and $x_1, \ldots, x_q$ are all distinct.

Find: The adversary must find the key K.

Two kinds of "attack" models are considered within this framework:

Known-message attack: $x_1, \ldots, x_q$ are any distinct points; the adversary has no control over them, and must work with whatever it gets.

Chosen-message attack: $x_1, \ldots, x_q$ are chosen by the adversary, perhaps even adaptively. That is, imagine it has access to an "oracle" for the function F_K. It can feed the oracle x_1 and get back $y_1 = F_K(x_1)$. It can then decide on a value x_2, feed the oracle this, and get back y_2, and so on.

Clearly a chosen-message attack gives the adversary much more power, and is also less realistic in practice.

Now, what about the security of DES? The most obvious attack is exhaustive key search.

Exhaustive key search: Go through all possible keys $K' \in \{0,1\}^k$ until you find the right one, namely K. How do you know when you hit K? If $\mathrm{DES}_{K'}(x_1) = y_1$, you bet that $K' = K$. (Remember the assumption is that the adversary has input-output examples $(x_1, y_1), \ldots, (x_q, y_q)$ of the real function DES_K). Of course, you could be wrong. But the "chance" of being wrong is small, and gets much smaller if you do more such tests. It turns out that two tests is quite enough. That is, this attack only needs $q = 2$, a very small number of input-output examples.

Let us now describe the attack in more detail. For $i = 1, \ldots, 2^{56}$ let K_i denote the i-th 56-bit string (in lexicographic order). The following algorithm implements the attack, returning in practice the key K with "high probability".

For $i = 1, \ldots, 2^{56}$ do
  If $\mathrm{DES}_{K_i}(x_1) = y_1$
    then if $\mathrm{DES}_{K_i}(x_2) = y_2$ then return K_i

How long does this take? In the worst case, 2^{56} DES computations. (It turns out you can use a property of DES to reduce this to 2^{55}, but we won't bother with these details.) Even if you use the above mentioned 1.6 Gbits/sec chip to do these computation, the search takes about 6,000 years. So key search appears to be infeasible.

Yet, this conclusion is actually too hasty. We will return to key search and see why later.

Differential and linear cryptanalysis: The discovery of a less trivial attack waited until 1990. Differential cryptanalysis is capable of finding the key using 2^{47} input-output examples (that is, it requires $q = 2^{47}$). However, differential cryptanalysis required a chosen-message attack.

Linear cryptanalysis improved differential in two ways. The number of input-output examples required is reduced to 2^{43}, but also only a known-message attack is required.

These were major breakthroughs in cryptanalysis. Yet, their practical impact is small. Why? It difficult to obtain 2^{43} input-output examples. Furthermore, simply storing all these examples requires about 140 terabytes of data.

So what's the best possible attack? The answer is exhaustive key search. What we ignored above is *parallelism.*

Key search machines: A few years back it was argued that one can design a $1 million machine that does the exhaustive key search in about 3.5 hours. More recently, a DES key search machine was actually built, at a cost of $250,000. It finds the key in about 2.5 expected time. The builders say it will be cheaper to build more machines now that this one is built.

Thus DES is feeling its age. Yet, it would be a mistake to take away from this discussion the impression that DES is weak. Rather, what the above says is that it is an impressively strong algorithm. After all these years, the best practical attack known is still exhaustive key search. That says a lot for its design and its designers.

Later we will see that that we would like security properties from a block cipher that go beyond resistance to key-recovery attacks. It turns out that from that point of view, a limitation of DES is its block size. Birthday attacks "break" DES with about $q = 2^{32}$ input output examples. (The meaning of "break" here is very different from above.) Here 2^{32} is the square root of 2^{64}, meaning to resist these attacks we must have bigger block size. The next generation of ciphers has taken this into account.

Exercise 2.2.1 *Show that* $\mathrm{DES}_K(X) = \overline{\mathrm{DES}_{\overline{K}}(X)}$. *This is called the key-complementation property of DES.*

Exercise 2.2.2 *Show how to use the key-complementation property of* DES *to speed up exhaustive key search by a factor of two. Explain any assumptions that you make.*

Exercise 2.2.3 *Find a key K such that* $\mathrm{DES}_K(X) = \mathrm{DES}_K(X)^{-1}$. *Such a key is sometimes called a "weak" key. How many weak keys can you find?*

2.3 RC6

There is an on-going effort to standardize a new algorithm to replace DES. It will be called the AES (Advanced Encryption Standard). The requirements are that it have a block size of 128, and a key size of at least 80. In response to the call, many proposals have been made. RC6 is one of the front-runners. Refer to [1] for its description and properties.

RC6 illustrates how cipher design has evolved: towards simplicity. It is a succinct and elegant algorithm which can be completely described in very little space. There are no large tables.

It has variable key and block size. The "standard" version sets both the key and block size to 128 bits.

The algorithm is relatively new, and not much cryptanalysis has yet been performed. It is fast in software.

Pretty much everything you would want to know is in [1] so we won't say more here.

Again, give a full description, but after the AES is selected.

2.4 Differential Cryptanalysis

To be written

2.5 Linear Cryptanalysis

To be written

2.6 Some Modes-of-Operation for Encryption

Let a block cipher F be fixed, and assume two parties share a key K for this block cipher. This gives them the ability to compute the functions $f(\cdot) = E_K(\cdot)$ and $f^{-1}(\cdot) = F_K^{-1}(\cdot)$. This function can be applied to an input of n-bits. Typically the block size n is 64 or 128. Yet in practice we want to process much larger inputs, say text files to encrypt. To do this one uses a block cipher in some mode of operation. There are several of these. We will illustrate by describing three of them that exhibit different kinds of features.

We will divide the presentation of the modes of operation into two parts. We first introduce some basic operations, then instantiate them in various ways to get the actual modes.

Three basic operations

We look at three modes: ECB (Electronic Code-Book), CBC (Cipher Block Chaining) and CTR (Counter). In each case there is an encryption process which takes an nm-bit plaintext x, for some $m \geq 1$, and returns a string y. An associated decryption process recovers x from y. The encryption operation is performed using a function f: $\{0,1\}^n \to \{0,1\}^n$. In typical usage, f will be $f = F_K$, the block-cipher induced permutation that the two parties sharing K can compute. But it is conceptually useful to separate the usage of the key from an application of the function $f = F_K$ that the key enables one to compute, because later we will want to plug in other functions in the role of f. So for now just imagine that you have available a subroutine (that is, an oracle) for computing a function $f\colon \{0,1\}^n \to \{0,1\}^n$. For ECB and CBC, the decryption requires the ability to compute f^{-1}, and thus in particular requires that f be a permutation, which is true when $f = F_K$ for a block cipher F. On the other hand CTR mode only needs access to f even for the decryption, and thus does not even require f to be a permutation.

Algorithm $\textsf{E-ECB}^f(x_1 \ldots x_m)$ For $i = 1, \ldots, m$ do $y_i \leftarrow f(x_i)$ Return $y_1 \ldots y_m$	Algorithm $\textsf{D-ECB}^{f^{-1}}(y_1 \ldots y_m)$ For $i = 1, \ldots, m$ do $x_i \leftarrow f^{-1}(y_i)$ Return $x_1 \ldots x_m$

CBC mode processes the data based on some *initial vector* IV which is an n-bit string.

Algorithm $\textsf{E-CBC}^f_{\mathrm{IV}}(x_1 \ldots x_m)$ $y_0 \leftarrow \mathrm{IV}$ For $i = 1, \ldots, m$ do $y_i \leftarrow f(y_{i-1} \oplus x_i)$ Return $y_0 y_1 \ldots y_m$	Algorithm $\textsf{D-CBC}^{f^{-1}}(y_0 y_1 \ldots y_n)$ For $i = 1, \ldots, n$ do $x_i \leftarrow f^{-1}(y_i) \oplus y_{i-1}$ Return $x_1 \ldots x_n$

Unlike $\textsf{E-ECB}^f(x)$ this operation is not length preserving: the output is n-bits longer than the input.

CTR mode also uses an auxiliary value, an "initial value" IV which is an integer in the range $0, 1, \ldots, 2^n - 1$.

Algorithm $\textsf{E-CTR}^f_{\mathrm{IV}}(x_1 \ldots x_m)$ For $i = 1, \ldots, m$ do $y_i \leftarrow f(\langle \mathrm{IV} + i \rangle) \oplus x_i$ Return $\langle \mathrm{IV} \rangle y_1 \ldots y_m$	Algorithm $\textsf{D-CTR}^f(\langle \mathrm{IV} \rangle y_1 \ldots y_m)$ For $i = 1, \ldots, m$ do $x_i \leftarrow f(\langle \mathrm{IV} + i \rangle) \oplus y_i$ Return $x_1 \ldots x_m$

Here addition is done modulo 2^n, and $\langle j \rangle$ denotes the binary representation of integer j as an l-bit string. Notice that in this case, recovery did not require access to f^{-1}, and in fact did not even require that f be a permutation. Also notice the efficiency advantage over CBC: the encryption is parallelizable.

Some modes of operation

The setting for using the modes of operation is that a block cipher F is agreed upon, and two communicating parties share a key K for this cipher that was chosen at random from $\{0,1\}^k$ and is initially unknown to the adversary. They then set $f = F_K$ in the above, and also suitably choose any other parameters. Here in more detail are several possible ways to compute a ciphertext y corresponding to the plaintext $x = x_1 \ldots x_n$. All of these are symmetric encryption schemes, and their security in this light will be discussed in the chapter on symmetric encryption.

Electronic codebook mode. The ciphertext is computed as $y = \textsf{E-ECB}^{F_K}(x_1 \ldots x_m)$. If F is a block cipher then knowledge of K enables computation of F_K^{-1}, and thus the ciphertext y can be decrypted to recover x, by a party knowing K, by following the reversal process $\textsf{D-ECB}^{F_K^{-1}}$ outlined above.

Cipher-block chaining mode. To encipher x, pick an l-bit IV at random and set $y = \textsf{E-CBC}_{\mathrm{IV}}^{F_K}(x_1 \ldots x_m)$. If F is a block cipher then knowledge of K enables computation of F_K^{-1}, and thus the ciphertext y can be decrypted to recover x, by a party knowing K, by following the reversal process $\textsf{D-CBC}^{F_K^{-1}}$ outlined above.

Note that this encryption procedure is probabilistic: each time a plaintext x is to be encrypted, a new random IV is chosen by the encryptor. The choices of IV are made independently each time.

Counter based cipher-block chaining mode. In this mode of operation, the encryptor maintains an n-bit counter c, which is an integer in the range $0, 1, \ldots, 2^n - 1$, initially 0. To encrypt plaintext x the encryptor sets $y = \textsf{E-CBC}_{\langle c \rangle}^{F_K}(x_1 \ldots x_m)$. The encryptor then increments c via $c \leftarrow c + n$. Note that this encryption procedure is stateful.

Counter based CTR mode. In this mode of operation, the encryptor maintains an n-bit counter c, which is an integer in the range $0, 1, \ldots, 2^n - 1$, initially 0. To encrypt plaintext x the encryptor sets $y = \textsf{E-CTR}_c^{F_K}(x_1 \ldots x_m)$. The encryptor then increments c via $c \leftarrow c + n$. A party knowing K can recover x from y via the reversal process $\textsf{D-CTR}^{F_K}$ outlined above.

Note that this encryption procedure is stateful: the sender must maintain a counter and be sure to increment it appropriately after each encryption. We stress that only the sender maintains a counter. The receiver does not, and thus there is no synchronization problem.

Randomized CTR mode. To encrypt x, pick an n-bit integer IV at random in the range $0, 1, \ldots, 2^n - 1$, and set $y = \textsf{E-CTR}_{\mathrm{IV}}^{F_K}(x_1 \ldots x_m)$. A party knowing K can recover x from y via the reversal process $\textsf{D-CTR}^{F_K}$ outlined above. Note that this encryption procedure is randomized.

Bibliography

[1] RONALD RIVEST, MATT ROBSHAW, RAY SIDNEY, AND YIQUIN YIN. The RC6 Block Cipher. Available via `http://theory.lcs.mit.edu/~rivest/publications.html`.

[2] Data Encryption Standard. FIPS PUB 46, Appendix A, Federal Information Processing Standards Publication, January 15, 1977, US Dept. of Commerce, National Bureau of Standards.