FOURTH EDITION

NETWORK DEFENSE AND COUNTERMEASURES

Principles and Practices

DR.CHUCK EASTTOM

Network Defense and Countermeasures

Principles and Practices

Fourth Edition

Dr. Chuck Easttom

PEARSON

this shortened World War II by as much as two years. This story is the basis for the 2014 movie *The Imitation Game*.

There have been systems either derived from Enigma or similar in concept. These include the Japanese system codenamed GREEN by American cryptographers, the SIGABA system, NEMA, and others.

Binary Operations

Part of modern symmetric cryptography ciphers involves using binary operations. Various operations on binary numbers (numbers made of only zeroes and ones) are well known to programmers and programming students. But for those readers not familiar with them, a brief explanation follows. When working with binary numbers, three operations are not found in normal math: AND, OR, and XOR operations. Each is illustrated next.

AND

To perform the AND operation, you take two binary numbers and compare them one place at a time. If both numbers have a one in both places, then the resultant number is a one. If not, then the resultant number is a zero, as you see here:

```
1 1 0 1
1 0 0 1
-------
1 0 0 1
```

OR

The OR operation checks to see whether there is a one in either or both numbers in a given place. If so, then the resultant number is one. If not, the resultant number is zero, as you see here:

```
1 1 0 1
1 0 0 1
-------
1 1 0 1
```

XOR

The XOR operation impacts your study of encryption the most. It checks to see whether there is a one in a number in a given place, but not in both numbers at that place. If it is in one number but not the other, then the resultant number is one. If not, the resultant number is zero, as you see here:

```
1 1 0 1
1 0 0 1
-------
0 1 0 0
```

XORing has a an interesting property in that it is reversible. If you XOR the resultant number with the second number, you get back the first number. And, if you XOR the resultant number with the first number, you get the second number.

```
0 1 0 0
1 0 0 1
-------
1 1 0 1
```

Binary encryption using the XOR operation opens the door for some rather simple encryption. Take any message and convert it to binary numbers and then XOR that with some key. Converting a message to a binary number is a simple two-step process. First, convert a message to its ASCII code, and then convert those codes to binary numbers. Each letter/number will generate an eight-bit binary number. You can then use a random string of binary numbers of any given length as the key. Simply XOR your message with the key to get the encrypted text, and then XOR it with the key again to retrieve the original message.

This method is easy to use and great for computer science students; however, it does not work well for truly secure communications because the underlying letter and word frequency remains. This exposes valuable clues that even an amateur cryptographer can use to decrypt the message. Yet, it does provide a valuable introduction to the concept of single-key encryption, which is discussed in more detail in the next section. Although simply XORing the text is not the method typically employed, single-key encryption methods are widely used today. For example, you could simply include a multi-alphabet substitution that was then XORed with some random bit stream—variations of which do exist in a few actual encryption methods currently used.

Modern cryptography methods, as well as computers, make decryption a rather advanced science. Therefore, encryption must be equally sophisticated in order to have a chance of success.

What you have seen so far regarding encryption is simply for educational purposes. As has been noted several times, you would not have a truly secure system if you implemented any of the previously mentioned encryption schemes. You might feel that this has been overstated in this text. However, having an accurate view of what encryption methods do and do not work is critical. It is now time to discuss a few methods that are actually in use today.

The following websites offer more information about cryptography:

- Cryptography I course on Coursera: https://www.coursera.org/course/crypto
- Applied cryptography course on Udacity: https://www.udacity.com/course/applied-cryptography--cs387
- Cypher Research Laboratories: www.cypher.com.au/crypto_history.htm

Understanding the simple methods described here and other methods provided by the aforementioned websites should give you a sense of how cryptography works as well as what is involved in encrypting a message. Regardless of whether you go on to study modern, sophisticated encryption methods, having some basic idea of how encryption works at a conceptual level is important. Having a basic

grasp of how encryption works, in principle, will make you better able to understand the concepts of any encryption method you encounter in the real world.

FYI: Careers in Cryptography

Some readers might be interested in a career in cryptography. Basic knowledge of cryptography is enough to be a security administrator, but not enough to be a cryptographer. A strong mathematics background is essential for in-depth exploration of cryptography, particularly when pursuing a career in this field. An adequate background includes a minimum of the complete calculus sequence (through differential equations), statistics through basic probability theory, abstract algebra, linear algebra, and number theory. A double major in computer science and mathematics is ideal. A minimum of a minor in mathematics is required, and familiarity with existing encryption methods is critical.

Learning About Modern Encryption Methods

Not surprisingly, modern methods of encryption are more secure than the historical methods just discussed. All the methods discussed in this section are in use today and are considered reasonably secure. Note that DES is an exception, but only due to its short key length.

In some cases the algorithm behind these methods requires a sophisticated understanding of mathematics. Number theory often forms the basis for encryption algorithms. Fortunately for our purposes having the exact details of these encryption algorithms is not important; this means that you don't require a strong mathematics background to follow this material. More important is a general understanding of how a particular encryption method works and how secure it is.

Symmetric Encryption

Symmetric encryption refers to those methods where the same key is used to encrypt and decrypt the plaintext.

Data Encryption Standard

Data Encryption Standard, or DES as it is often called, was developed by IBM in the early 1970s and made public in 1976. DES uses a symmetric key system. Recall from our earlier discussion that this means the same key is used to encrypt and to decrypt the message. DES uses short keys and relies on complex procedures to protect its information. The actual DES algorithm is quite complex. The basic concept, however, is as follows:

1. The data is divided into 64-bit blocks, and those blocks are then transposed.
2. Transposed data is then manipulated by 16 separate rounds of encryption, involving substitutions, bit-shifting, and logical operations using a 56-bit key.
3. Finally, the data is transposed one last time.

More information about DES is available at the Federal Information Processing Standards website at https://csrc.nist.gov/csrc/media/publications/fips/46/3/archive/1999-10-25/documents/fips46-3.pdf. A more detailed description of DES is given below, but you can skip this if you wish.

DES uses a 56-bit cipher key applied to a 64-bit block. There is actually a 64-bit key, but one bit of every byte is actually used for error detection, leaving just 56 bits for actual key operations.

DES is a Feistel cipher with 16 rounds and a 48-bit round key for each round. A round key is just a sub key that is derived from the cipher key each round, according to a key schedule algorithm. DES's general functionality follows the Feistel method of dividing the 64-bit block into two halves (32 bits each; this is not an unbalanced Feistel cipher), applying the round function to one half, then XORing that output with the other half.

The first issue to address is the key schedule. How does DES generate a new sub key each round? The idea is to take the original 56-bit key and to slightly permute it each round, so that each round is applying a slightly different key, but one that is based on the original cipher key. To generate the round keys, the 56-bit key is split into two 28-bit halves and those halves are circularly shifted after each round by one or two bits. This will provide a different sub key each round. During the round key generation portion of the algorithm (recall that this is referred to as the *key schedule*) each round, the two halves of the original cipher key (the 56 bits of key the two endpoints of encryption must exchange) are shifted a specific amount.

Once the round key has been generated for the current round, the next step is to address the half of the original block that is going to be input into the round function. Recall that the two halves are each 32 bits. The round key is 48 bits. That means that the round key does not match the size of the half block it is going to be applied to. You cannot really XOR a 48-bit round key with a 32-bit half block, unless you simply ignore 16 bits of the round key. If you did so, you would basically be making the round key effectively shorter and thus less secure, so this is not a good option.

The 32-bit half needs to be expanded to 48 bits before it is XORed with the round key. This is accomplished by replicating some bits so that the 32-bit half becomes 48 bits.

This expansion process is actually quite simple. The 32 bits that are to be expanded are broken into 4-bit sections. The bits on each end are duplicated. If you divide 32 by 4 the answer is 8. So there are eight of these 4-bit groupings. If you duplicate the end bits of each grouping, that will add 16 bits to the original 32, thus providing a total of 48 bits.

It is also important to keep in mind that it was the bits on each end that were duplicated; this will be a key item later in the round function. Perhaps this example will help you to understand what is occurring at this point. Let us assume 32 bits as shown here:

11110011010111111111000101011001

Now divide that into eight sections each of 4 bits, as shown here:

1111 0011 0101 1111 1111 0001 0101 1001

Now each of these has its end bits duplicated, as you see here:

1111 becomes 111111

0011 becomes 000111

0101 becomes 001011

1111 becomes 111111

1111 becomes 111111

0001 becomes 000011

0101 becomes 001011

1001 becomes 110011

The resultant 48-bit string is now XORed with the 48-bit round key. That is the extent of the round key being used in each round. It is now dispensed with, and on the next round another 48-bit round key will be derived from the two 28-bit halves of the 56-bit cipher key.

Now we have the 48-bit output of the XOR operation. That is now split into eight sections of 6 bits each. For the rest of this explanation we will focus on just one of those 6-bit sections, but keep in mind that the same process is done to all eight sections.

The 6-bit section is used as the input to an s-box. An s-box is a table that takes input and produces an output based on that input. In other words, it is a substitution box that substitutes new values for the input. The s-boxes used in DES are published, the first of which is shown in Figure 6-1.

	x0000x	x0001x	x0010x	x0011x	x0100x	x0101x	x0110x	x0111x	x1000x	x1001x	x1010x	x1011x	x1100x	x1101x	x1110x	x1111x
0yyyy0	12	1	10	15	9	2	6	8	0	13	3	4	14	7	5	11
0yyyy1	10	15	4	2	7	12	9	5	6	1	13	14	0	11	3	8
1yyyy0	9	14	15	5	2	8	12	3	7	0	4	10	1	13	11	6
1yyyy1	4	3	2	12	9	5	15	10	11	14	1	7	6	0	8	13

FIGURE 6-1 The first DES s-box

Notice this is simply a lookup table. The 2 bits on either end are shown in the left hand column and the 4 bits in the middle are shown in the top row. They are matched, and the resulting value is the output of the s-box. For example, with the previous demonstration numbers we were using, our first block would be 111111. So you find 1xxxx1 on the left and x1111x on the top. The resulting value is 13 in decimal or 1101 in binary.

At the end of this you have produced 32 bits that are the output of the round function. Then in keeping with the Feistel structure, they get XORed with the 32 bits that were not input into the round function, and the two halves are swapped. DES is a 16-round Feistel cipher, meaning this process is repeated 16 times.

There are only two parts still left to discuss regarding DES. The first is the initial permutation, called the IP, then the final permutation, which is an inverse of the IP.

One advantage that DES offers is efficiency. Some implementations of DES offer data throughput rates on the order of hundreds of megabytes per second. In plain English, what this means is that it can encrypt a great deal of data very quickly. You might assume that 16 steps would cause encryption to be quite slow; however, that is not the case using modern computer equipment. The problem with DES is the same problem that all symmetric key algorithms have: How do you transmit the key without it becoming compromised? This issue led to the development of public key encryption.

Another advantage of DES is the complexity with which it scrambles the text. DES uses 16 separate rounds to scramble the text. This yields a scrambled text that is very difficult to break. DES is no longer used, because the short key size is no longer adequate against brute force attacks. However, the overall structure, called a Feistel network or Feistel cipher, is the basis for many algorithms that are still used today, such as Blowfish.

As has been mentioned, DES uses a key that is no longer considered long enough. Modern computers can brute-force crack a 56-bit key. The algorithm used in DES is actually quite good. It was the first widely used Feistel structure, and that structure is still a good basis for block ciphers.

As computers became more powerful, the search began for a DES replacement. Ultimately, the Rijndael cipher would be used for the Advanced Encryption Standard (AES) and would replace DES. In the interim, the idea was to use multiple DES keys to encrypt. Ideally, three separate 56-bit keys were used; thus this interim solution was called triple-DES or 3DES. In some cases only two DES keys were used, and the algorithm alternated applying them.

Blowfish

Blowfish is a symmetric block cipher. This means that it uses a single key to both encrypt and decrypt the message and works on "blocks" of the message at a time. It uses a variable-length key ranging from 32 to 448 bits. This flexibility in key size allows you to use it in various situations. Blowfish was designed in 1993 by Bruce Schneier. It has been analyzed extensively by the cryptography community and has gained wide acceptance. It is also a non-commercial (that is, free of charge) product, thus making it attractive to budget-conscious organizations.

FYI: Block Ciphers and Stream Ciphers

A block cipher operates on blocks of fixed length, often 64 or 128 bits. In a block cipher, a cryptographic key and algorithm are applied to a block of data (for example, 64 contiguous bits) at once as a group rather than to one bit at a time. Stream ciphers simply take the text as an ongoing stream, encrypting each bit as it encounters it. Stream ciphers tend to be faster than block ciphers. A stream cipher generates a keystream (a sequence of bits used as a key). Encryption is accomplished by combining the keystream with the plaintext, usually with the bitwise XOR operation.