Cryptography: An Introduction (3rd Edition)

Nigel Smart

Chapter 6 — Edited for CIS 331 (cut short)

CHAPTER 6

Historical Stream Ciphers

Chapter Goals

- To introduce the general model for symmetric ciphers.
- To explain the relation between stream ciphers and the Vernam cipher.
- To examine the working and breaking of the Lorenz cipher in detail.

1. Introduction To Symmetric Ciphers

A symmetric cipher works using the following two transformations

$$c = e_k(m),$$
$$m = d_k(c)$$

where

- m is the plaintext,
- e is the encryption function,
- d is the decryption function,
- k is the secret key,
- c is the ciphertext.

It should be noted that it is desirable that both the encryption and decryption functions are public knowledge and that the secrecy of the message, given the ciphertext, depends totally on the secrecy of the secret key, k. Although this well-established principle, called Kerckhoffs' principle, has been known since the mid-1800s many companies still ignore it. There are instances of companies deploying secret proprietary encryption schemes which turn out to be insecure as soon as someone leaks the details of the algorithms. The best schemes will be the ones which have been studied by a lot of people for a very long time and which have been found to remain secure. A scheme which is a commercial secret cannot be studied by anyone outside the company.

The above setup is called a symmetric key system since both parties need access to the secret key. Sometimes symmetric key cryptography is implemented using two keys, one for encryption and one for decryption. However, if this is the case we assume that given the encryption key it is easy to compute the decryption key (and vice versa). Later we shall meet public key cryptography where only one key is kept secret, called the private key, the other key, called the public key is allowed to be published in the clear. In this situation it is assumed to be computationally infeasible for someone to compute the private key given the public key.

Returning to symmetric cryptography, a moment's thought reveals that the number of possible keys must be very large. This is because in designing a cipher we assume the worst case scenario and give the attacker the benefit of

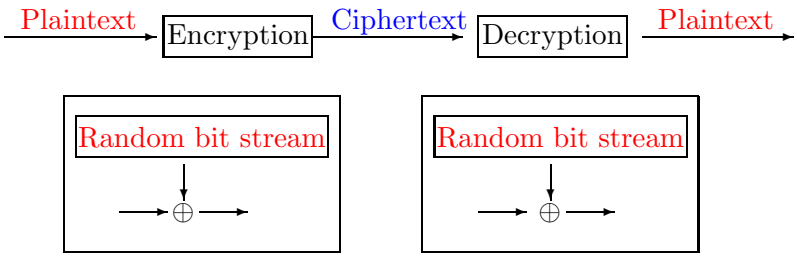
- full knowledge of the encryption/decryption algorithm,
- a number of plaintext/ciphertext pairs associated to the target key k.

If the number of possible keys is small then an attacker can break the system using an exhaustive search. The attacker encrypts one of the given plaintexts under all possible keys and determines which key produces the given ciphertext. Hence, the key space needs to be large enough to avoid such an attack. It is commonly assumed that a computation taking 2^{80} steps will be infeasible for a number of years to come, hence the key space size should be at least 80 bits to avoid exhaustive search.

The cipher designer must play two roles, that of someone trying to break as well as create a cipher. These days, although there is a lot of theory behind the design of many ciphers, we still rely on symmetric ciphers which are just believed to be strong, rather than ones for which we know a reason why they are strong. All this means is that the best attempts of the most experienced cryptanalysts cannot break them. This should be compared with public key ciphers, where there is now a theory which allows us to reason about how strong a given cipher is (given some explicit computational assumption).

Fig. 1 describes a simple model for enciphering bits, which although simple is quite suited to practical implementations. The idea of this model is to apply a reversible operation to the plaintext

FIGURE 1. Simple model for enciphering bits



to produce the ciphertext, namely combining the plaintext with a 'random stream'. The recipient can recreate the original plaintext by applying the inverse operation, in this case by combining the ciphertext with the same random stream.

This is particularly efficient since we can use the simplest operation available on a computer, namely exclusive-or $\oplus$. We saw in Chapter 5 that if the key is different for every message and the key is as long as the message, then such a system can be shown to be perfectly secure, namely we have the one-time pad. However, the one-time pad is not practical in many situations.

- We would like to use a short key to encrypt a long message.
- We would like to reuse keys.

Modern symmetric ciphers allow both of these properties, but this is at the expense of losing our perfect secrecy property. The reason for doing this is because using a one-time pad produces horrendous key distribution problems. We shall see that even using reusable short keys also produces bad (but not as bad) key distribution problems.

There are a number of ways to attack a bulk cipher, some of which we outline below. We divide our discussion into passive and active attacks; a passive attack is generally easier to mount than an active attack.

- **Passive Attacks:** Here the adversary is only allowed to listen to encrypted messages. Then he attempts to break the cryptosystem by either recovering the key or determining some secret that the communicating parties did not want leaked. One common form of passive attack is that of traffic analysis, a technique borrowed from the army in World War I, where a sudden increase in radio traffic at a certain point on the Western Front would signal an imminent offensive.

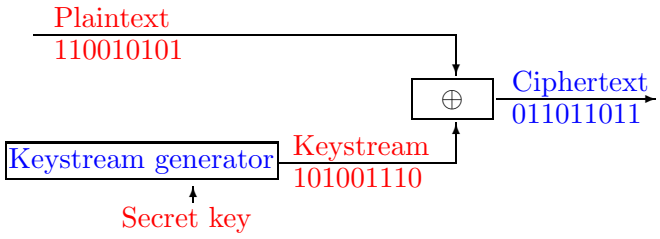
- **Active Attacks:** Here the adversary is allowed to insert, delete or replay messages between the two communicating parties. A general requirement is that an undetected insertion attack should require the breaking of the cipher, whilst the cipher needs to allow detection and recovery from deletion or replay attacks.

Bulk symmetric ciphers essentially come in two variants: stream ciphers, which operate on one data item (bit/letter) at a time, and block ciphers, which operate on data in blocks of items (e.g. 64 bits) at a time. In this chapter we look at stream ciphers, we leave block ciphers until Chapter 8.

2. Stream Cipher Basics

Fig. 2 gives a simple explanation of a stream cipher. Notice how this is very similar to our previous simple model. However, the random bit stream is now produced from a short secret key using a public algorithm, called the keystream generator.

FIGURE 2. Stream ciphers



Thus we have $c_i = m_i \oplus k_i$ where

- $m_0, m_1, \ldots$ are the plaintext bits,
- $k_0, k_1, \ldots$ are the keystream bits,
- $c_0, c_1, \ldots$ are the ciphertext bits.

This means

$$m_i = c_i \oplus k_i$$

i.e. decryption is the same operation as encryption.

Stream ciphers such as that described above are simple and fast to implement. They allow very fast encryption of large amounts of data, so they are suited to real-time audio and video signals. In addition there is no error propagation, if a single bit of ciphertext gets mangled during transit (due to an attacker or a poor radio signal) then only one bit of the decrypted plaintext will be affected. They are very similar to the Vernam cipher mentioned earlier, except now the key stream is only pseudo-random as opposed to truly random. Thus whilst similar to the Vernam cipher they are *not* perfectly secure.

Just like the Vernam cipher, stream ciphers suffer from the following problem; the same key used twice gives the same keystream, which can reveal relationships between messages. For example suppose m_1 and m_2 were encrypted under the same key k, then an adversary could work out the exclusive-or of the two plaintexts without knowing what the plaintexts were

$$c_1 \oplus c_2 = (m_1 \oplus k) \oplus (m_2 \oplus k) = m_1 \oplus m_2.$$

Hence, there is a need to change keys frequently either on a per message or on a per session basis. This results in difficult key management and distribution techniques, which we shall see later how to solve using public key cryptography. Usually public key cryptography is used to determine

session or message keys, and then the actual data is rapidly encrypted using either a stream or block cipher.

The keystream generator above needs to produce a keystream with a number of properties for the stream cipher to be considered secure. As a bare minimum the keystream should

- Have a long period. Since the keystream k_i is produced via a deterministic process from the key, there will exist a number N such that

  $$k_i = k_{i+N}$$

  for all values of i. This number N is called the period of the sequence, and should be large for the keystream generator to be considered secure.
- Have pseudo-random properties. The generator should produce a sequence which appears to be random, in other words it should pass a number of statistical random number tests.
- Have large linear complexity. See Chapter 7 for what this means.

However, these conditions are not sufficient. Generally determining more of the sequence from a part should be computationally infeasible. Ideally, even if one knows the first one billion bits of the keystream sequence, the probability of guessing the next bit correctly should be no better than one half.