Chapter 4

Random Numbers

The Security of many cryptographic systems depends upon the generation of unpredictable quantities. As we have seen earlier, the one time pad requires a random key in order to operate effectively. As well as that, the DES key needs to be random and unpredictable. It must be remembered that a cryptographic algorithm's security rests with its key (this is known as Kerckhoffs' principle which was stated by Auguste Kerckhoffs in the 19th Century). As a result, any predictability in the value of the key leads to a weakness in the entire system.

This chapter will look at the role of random numbers in cryptography and a couple of methods for producing them. It will be seen that true random numbers are very hard to produce accurately and require some physical phenomena which is not always practical. Failing the use of a true random source, a number of so called **pseudorandom sources** have been developed. These work much better in practical environments.

Some other examples of random numbers include the following:

- Reciprocal authentication schemes. The use of random numbers for nonce's (described in the last chapter) frustrate replay attacks. Figure 3.6 is an example.
- Session key generation for conventional encryption, and generation of keys for the RSA public key algorithm.

4.1 Generating Random numbers

It is not that easy to generate random numbers and the numbers produced must be random in some well-defined statistical sense. Two criteria used to validate the randomness of a sequence of numbers are:

1. **Uniform distribution** - the frequency of occurrence of each of the numbers should be approximately the same.
2. **Independence** - no one value in the sequence can be inferred from the others.

Although there are many tests to determine whether a sequence has uniform distribution there is no test to prove independence. However, a number of tests can be applied to demonstrate that a sequence doesn't exhibit independence. The approach therefore is to apply a sufficient number of tests to achieve a certain level of confidence that independence exists.

Sources of true random numbers are hard to come by. Physical noise generators such as ionising radiation events, leaky capacitors etc. are examples but are not too useful in network security applications. Deterministic algorithmic techniques are used instead and if the algorithm is good, the resulting sequence will pass many tests of randomness. These are called **PseudoRandom Number Generators (PRNG)**.

4.1.1 Linear Congruential Generator

The most widely used technique is the **linear congruential method** which was first introduced by D.H. Lehmer in 1948. The algorithm is parameterised as follows:

m	the modulus	$m > 0$
a	the multiplier	$0 < a < m$
c	the increment	$0 \leq c < m$
X_0	the starting value, or seed	$0 \leq X_0 < m$

The sequence of random numbers X_n is obtained via the following iterative equation:

$$X_{n+1} = (aX_n + c) \bmod m \tag{4.1}$$

If m,a,c and X_0 are integers then the technique will produce a sequence of integers with each integer in the range $0 \leq X_n < m$. The selection of a,c and m is critical in developing a good generator. For example, if $a = c = 1$ the sequence produced is of no use as we simply obtain a sequence of integers incremented by 1 each time. However, consider $a = 7, c = 0, m = 32$ and $X_0 = 1$. The resultant sequence is also clearly unsatisfactory $\{7, 17, 23, 1, 7, \ldots\}$. Of the 32 possible values, only 4 are used (the sequence is said to have a period of 4). Changing $a = 5$ then the sequence is $\{5, 25, 29, 17, 21, 9, 13, 1, etc.\}$ which gives a period of 8. This is better but still far from what is desired.

The value m should be large so that there is potential for long sequences, usually m is nearly equal to the maximum nonnegative integer for a given computer e.g. $2^{31} - 1$.

Three criteria that have been proposed to assess random number generators are as follows:

T_1 : The function should be a full-period generating function (i.e. should generate all numbers between 0 and m before repeating).

T_2 : The generated sequence should appear random. The sequence is not random at all but there is a wide variety of tests to assess the degree of randomness exhibited.

T_3 : The function should implement efficiently with 32-bit arithmetic.

With appropriate values of a, c and m, these criteria may be achieved. It can be shown that if m is prime and c = 0, then for certain values of a the period of the generated sequence is $m - 1$, with only the value 0 missing. For 32 bit arithmetic, a convenient large prime of m is $2^{31} - 1$ and the function is as follows:

$$X_{n+1} = (aX_n) \bmod (2^{31} - 1)$$

Of the more that 2 billion choices for a only a handful of multipliers pass all three tests. One of these is $a = 7^5 = 16,807$, which was originally designed for use in the IBM360 family of computers and has been extensively tested.

The strength of the linear congruential algorithm is that if the multiplier and modulus are chosen properly, then the resultant sequence will be *indistinguishable* from a sequence drawn randomly (but without replacement) from the set $1, 2, 3, 4, \ldots, m - 1$. But there is nothing random about the algorithm, apart from the choice of the initial value X_0. Once that value has been chosen, the remaining numbers in the sequence follow deterministically. If an opponent knows the algorithm is being used, he/she can easily discover the sequence.

For example, assume the attacker knows X_0, X_1, X_2, X_3 then it is possible to set up three simultaneous equations as follows:

$$\begin{aligned} X_1 &= (a \cdot X_0 + c) \bmod m \\ X_2 &= (a \cdot X_1 + c) \bmod m \\ X_3 &= (a \cdot X_2 + c) \bmod m \end{aligned}$$

As there are three equations in three unknowns, it is possible to solve for the three unknowns.

To make the sequence less reproducible, the sequence could be restarted after every N numbers. This can be achieved using the current clock value ($\mathrm{mod}(m)$) as a new seed after N numbers (starting the new sequence). Another method could be to add the current value of the clock ($\mathrm{mod}(m)$) to the random numbers produced.

4.1.2 Cyclic Encryption

It makes sense to take advantage of the encryption logic available to produce the random numbers. An example of **Cyclic Encryption** is shown in fig 4.1. This is a method of generating a session key from a master key. A counter with period N provides input to the encryption logic. If 56 bit DES keys are to be produced, then a counter with a period of 2^{56} may be used. After each key is produced the counter is incremented, thus the pseudorandom numbers generated by this scheme cycle through a full period. To further strengthen the algorithm, the counter could be replaced by a full period PRNG.

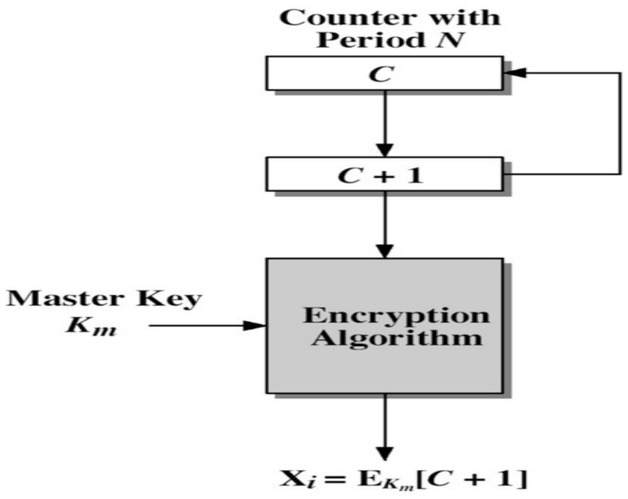

Figure 4.1: Pseudorandom number generation from a counter.

4.1.3 ANSI X9.17: Pseudorandom Number Generator

This is considered one of the strongest (cryptographically speaking) pseudorandom number generators. It is employed by a number of applications including financial security schemes and PGP (to be discussed later in the course). It is a U.S. Federal Information Processing Standard (FIPS) approved method. It makes use of Triple DES to produce random numbers. Although it uses triple DES three times, it only uses two keys as the same two keys are used three times. This is effectively the same as using a 112 bit key.

Figure 4.2 shows the scheme. The algorithm makes use of triple DES as mentioned and has the following inputs and output:

- *Input*: Two pseudorandom inputs drive the algorithm. One is a 64 bit representation of the current date and time updated on each number generation. The other is a 64 bit seed value initialized to some arbitrary value and updated during the generation process.
- *Keys*: The generator makes use of three triple DES encryption modules. All three make use of the same pair of 56 bit keys, which must be kept secret.
- *Output*: These are a 64 bit pseudorandom number (R_i) and a 64 bit seed value (V_{i+1}).

The strength of the PRNG of figure 4.2 can be attributed to the fact that there are a total of nine DES encryptions and a 112 bit key (effectively). As well as this, the scheme takes two pseudorandom inputs to begin with, both of which are not revealed (although it might be possible to work out DT_i). As a result the amount of information needed

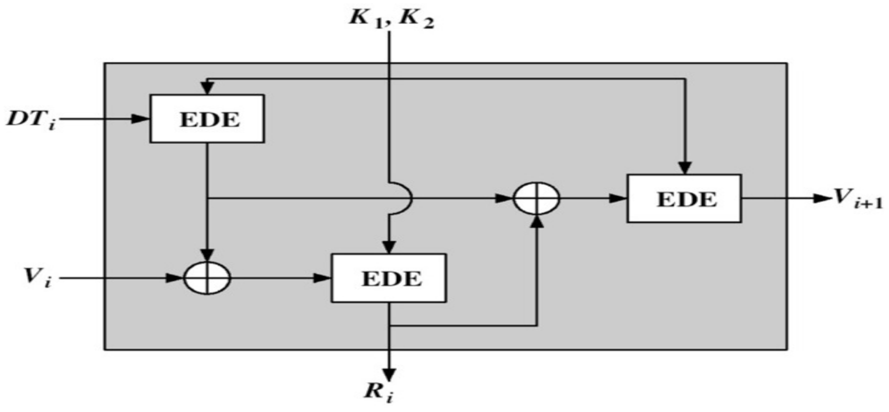

Figure 4.2: ANSI X9.17 Scheme.

by and attacker is formidable. Even if the attacker learns R_i, it would be impossible to deduce v_{i+1} due to extra EDE encryption.

4.2 Reducing Possibilities for Traffic Analysis

With the use of link encryption it is possible to mitigate traffic analysis due to the header being encrypted. However it is still possible to see the amount of traffic being sent. This may allow the attacker to deduce certain information about the communication. It is possible to use random number generation as a countermeasure to this attack. Figure 4.3 shows how.

The output of the model produces ciphertext continuously when no input (plaintext) is available as the random number generator takes over producing a series of encrypted random numbers. When an input becomes available the PRNG is switched away from the output and the input is encrypted.

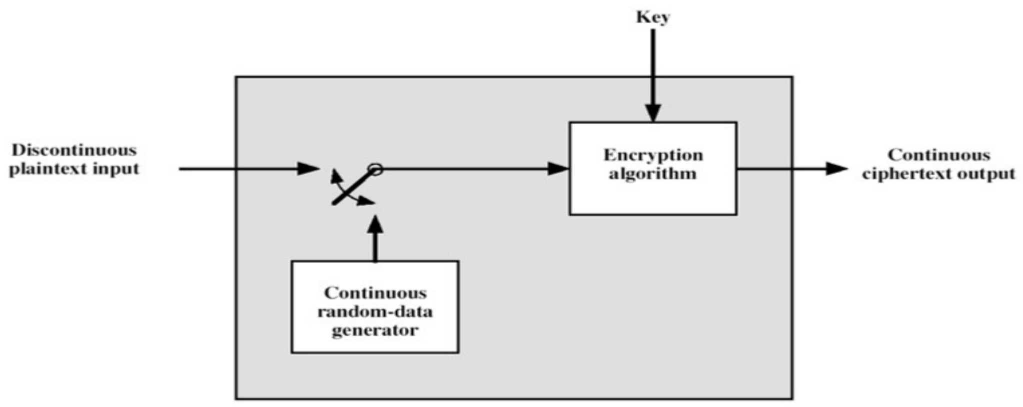
Key
Discontinuous plaintext input
Encryption algorithm
Continuous ciphertext output
Continuous random-data generator

Figure 4.3: Traffic padding encryption device.