Random Number Generator

Naga Bharat Reddy Dasari

October 2 2015

1 Introduction

Random Number Generator was classified into two types, non-deterministic random bit generator (NRBG) and Deterministic Random Bit Generators (DRBG). Random Bits produced by involving some physical process which are unpredictable, fall into class of NRBG's. The rest of the random bit generators which were developed using some strong mathematical logic and algorithms, fall into the class of DRBG's.

In this paper we would see the different DRBG algorithms which were specifically designed to accommodate the properties of cryptography. These algorithm would fall into the category of cryptographically secure pseudo-random number generator (CSPRNG).An algorithm is said to be CSPRNG if they pass statistical randomness test and also hold up well under serious attack, even when part of their initial or running state becomes available to an attacker[1].

2 History

There are many pseudo random number generators which have been proposed and implemented throughout the decades. The first pseudo random number generator algorithm for the use electronic computer was proposed in 1951 by Jon von Neumann [2] and this algorithm implements middle square method. The disadvantage of this method was choosing the initial seed.

Some more pseudo random number generators with high importance in the history include Linear congruential generator, Mersenne twister, Blum Blum Shub , Wichmann-Hill etc., But to be a CSPRNG any algorithm need to fulfill the requirements mentioned in section 1. There are many examples in cryptographic ciphers which are excellent but the random choices were not random enough and security was lost as a direct consequence. For an example, the Purple cipher machine used in World War II by Japanese for communication was consistently broken because the randomness of the key for encryption was not sufficient. Hence to avoid this kind encryption several CSPRNG are standardized.

2.1 Attacks on random number generator

Early versions of Netscape's Secure Socket Layer (SSL) encryption protocol used pseudo-random quantities derived from a PRNG seeded with three variable values: the time of day, the process ID, and the parent process ID. These quantities are often relatively predictable, and so have little entropy and are less than random, and so that version of SSL was found to be insecure as a result. That RNG was fixed in later releases by more robust seeding.

Microsoft uses an unpublished algorithm to generate random values for its Windows operating system. These random quantities are made available to users via the CryptGenRandom utility. In

November 2007 a paper titled Cryptanalysis of the Random Number Generator of the Windows Operating System[3] was published which presented serious weaknesses in Microsoft's approach at the time. Microsoft has stated that the problems described in the paper have been addressed in subsequent releases of Windows, which use a different RNG implementation.

To avoid attacks the National Institute of standards and technology has published NIST SP 800-900A recommending the standards to design an DRGB which included four different techniques to implement DRBG but one among them Dual_EC_DRBG is a PRNG which was a standardized in NIST SP 800-90A but National Security Agency (NSA) has shown that it is cryptographically insecure. The other mechanisms Hash_DRBG, HMAC_DRBG, CTR_DRBG are uncontroversial till now and hence assumed to provide required standard security for Cryptography.

3 Description

Any Random Number Generator which is DRBG is deterministic i.e the random number generator would repeat the sequence of the numbers generated after certain period length. The seed for the RNG is the initial value with which first random number is calculated and when the period length is reached then the first random number is repeated and so the rest of the numbers in the same order. One of simple and easiest RNG which can help us to understand this theory is Linear Congruential Generators.

3.1 Linear Congruential Generator

Linear Congruential generator (LCG) is one of the oldest and most important Pseudo Random Number Generator. They are fast to implement on the computer which can provide modulo arithmetic by storage-bit truncation. All the Linear Congruential Generators follow the following recursive formula to generate random numbers.

$$X_{n+1} = (aX_n + c) \quad mod \quad m$$

Where X is the sequence of pseudo random values, and
$m, 0 < m$ -The Modulus
$a, 0 < a < m$ -The multiplier
$c, 0 \leq c < m$ -The increment
$X_0, X_0 \leq 0 < m$ -The "seed" or "start value"

For an LCG the period length can be at most m. The LCG will have a full period for all seed values if and only if:

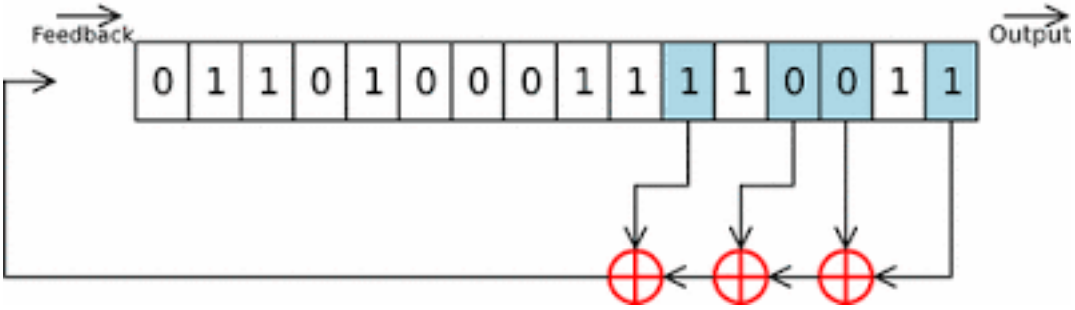
1. m and the offset c are relatively prime,
2. $a - 1$is divisible by all prime factors of m,
3. $a - 1$is divisible by 4 if m is divisible by 4.

But anyone who is recording the sequence can easily determine the next occurring random number. Therefore to avoid this kind of attack we need to have large period length. The built-in function rand() in most of the compilers use LCG with modulus m = 232 to 264 hence makes the LCG to be unpredictable as the range is more and thus considered to be more efficient. LCG are used in the applications that require a RNG to be fast and uses minimum memory. But the applications which require high quality of randomness cannot use LCG like in cryptography. If a linear congruential generator is seeded with a character and then iterated once, the result is a

simple classical cipher which can be easily broken by standard frequency analysis [4]. Hence to avoid classical ciphers the RNG which can generate stream ciphers has been introduced into the world of cryptography.

3.2 Linear feedback shift registers

A linear-feedback shift register (LFSR) is a shift register whose input bit is a linear function of its previous state [5]. An LFSR mostly uses XOR circuit as linear function and as the stream values produced depend on the previous state of the registers and the initial seed we can say an LFSR is deterministic. LFSR has a finite number of states therefore it ends up into repeating cycle if a proper feedback function is not used to generate random numbers.

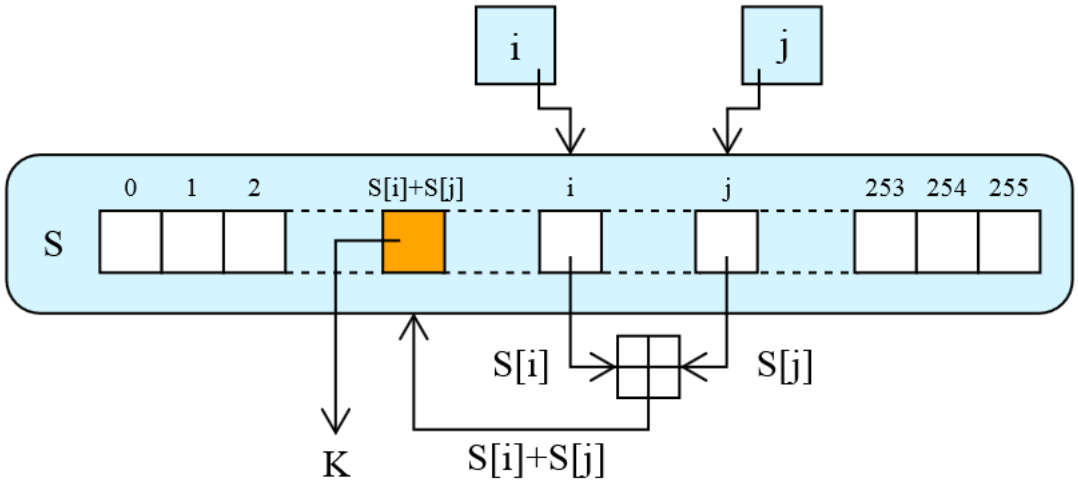


LFSRs are used as PRNG in the cryptography for military operations as they are simple to construct with the hardware and output uniform distribution of stream ciphers. But LFSR is a linear system it can be analyzed using cryptanalysis and can be intercepted using Berlekamp–Massey algorithm [6]. To avoid this kind of attack due to linearity the LFSR is implemented using non linear combination of bits or the output bits of multiple LFSR or irregular clocking of LFSR.Important LFSR-based stream ciphers include A5/1 and A5/2, used in GSM cell phones, E0, used in Bluetooth, and the shrinking generator. The A5/2 cipher has been broken and both A5/1 and E0 have serious weaknesses. LFSR is most likely a hardware Implementation than a software implementation. Another form of cipher is considered in cryptography for the software implementation.

3.3 RC4 PRNG

RC4 is a stream ciphers which remarkable for its simplicity and speed in software. For as many iterations as are needed, the PRGA modifies the state and outputs a byte of the keystream. In each iteration, the PRGA increments i, looks up the ith element of S, S[i], and adds that to j, exchanges the values of S[i] and S[j], and then uses the sum S[i] + S[j] (modulo 256) as an index to fetch a third element of S, (the keystream value K below) which is XORed with the next byte of the message to produce the next byte of either ciphertext or plaintext. Each element of S is swapped with another element at least once every 256 iterations.

Several operating systems like FreeBSD, Linux's libbsd, and Mac OS X access to a random number generator originally based on RC4 and proposed new random number generators are often compared to the RC4 random number generator. Unfortunately several attacks on RC4 were able to distinguish its output from random sequence hence the standardization for the CSPRNG to avoid further attacks has become more important.

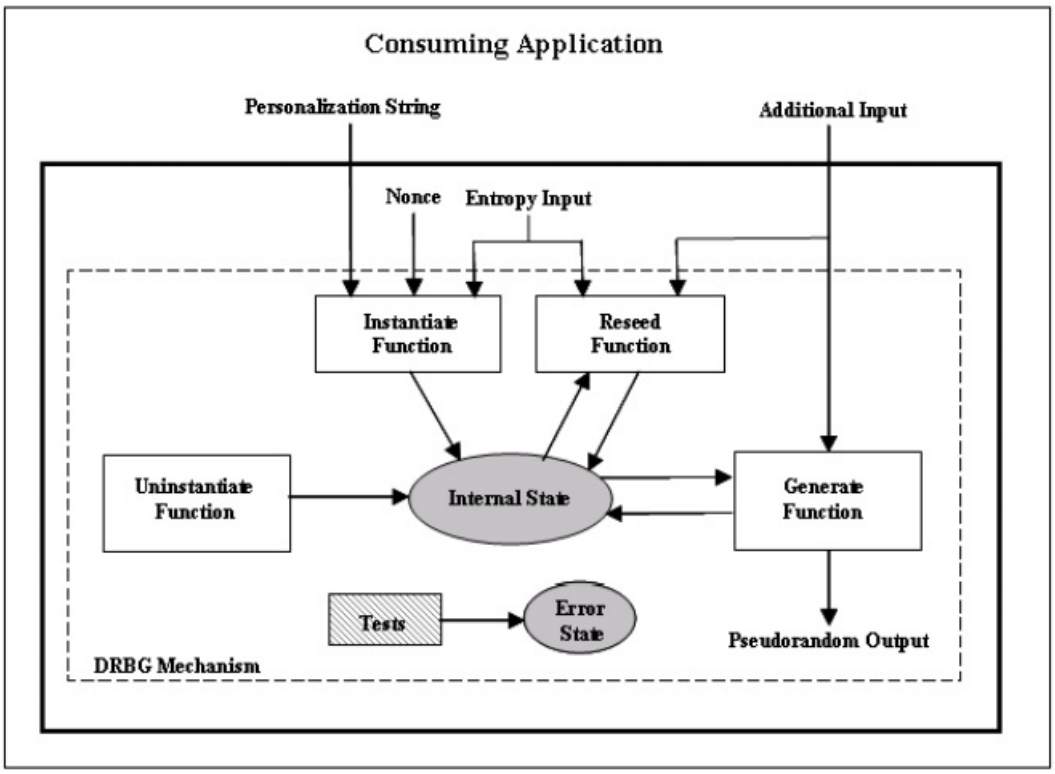



```
i := 0 ;
j := 0 ;
while GeneratingOutput: do
    i := (i + 1) mod 256;
    j := (j + S[i]) mod 256;
    swap values of S[i] and S[j];
    K := S[(S[i] + S[j]) mod 256];
    output K;
end
```

Algorithm 1: Pseudo Code for RC4

3.4 Hash_DRBG

According to the NIST 800-900A recommendation any DRBG model would include instantiate function, generate function, and reseed function, uninstantiate function and a health test function. The following diagram explains you functional model of a DRBG.



According to NIST SP 800-90A [7] the hash-based DRBG is designed using hash function that may be used by consuming applications requiring various security strengths. This algorithm produces a sequence of bits from an initial value that is determined by a seed which is determined from the input from the randomness source. Once the seed is provided and the initial value is determined, the algorithm is said to be instantiated and may be used to produce output. The seed here is kept secret

3.4.1 Hash Function

A cryptographic hash function is a hash function which is considered practically impossible to invert, that is, to recreate the input data from its hash value alone. These one-way hash functions have been called "the workhorses of modern cryptography". The input data is often called the message, and the hash value is often called the message digest or simply the digest. The ideal cryptographic hash function has four main properties:

- It is easy to compute the hash value for any given message
- It is infeasible to generate a message from its hash
- It is infeasible to modify a message without changing the hash
- It is infeasible to find two different messages with the same hash.

SHA-1 is one of the important hash functions that had been used. SHA-1 produces a 160-bit hash value known as a message digest. A SHA-1 hash value is typically rendered as a hexadecimal number, 40 digits long.

References

[1] https://en.wikipedia.org/wiki/Cryptographically_secure_pseudorandom_number_generator

[2] NIST SP 800-90A, Recommendation for Random Number Generation Using Deterministic Random Bit Generators,

[3] Dorrendorf, Leo; Gutterman, Zvi; Pinkas, Benny (1 October 2009). "Cryptanalysis of the random number generator of the Windows operating system"

[4] https://en.wikipedia.org/wiki/Pseudorandomness

[5] https://en.wikipedia.org/wiki/Frequency_analysis

[6] https://en.wikipedia.org/wiki/Linear_feedback_shift_register

[7] https://en.wikipedia.org/wiki/Berlekamp%E2%80%93Massey_algorith

[8] https://en.wikipedia.org/wiki/Pseudorandomness