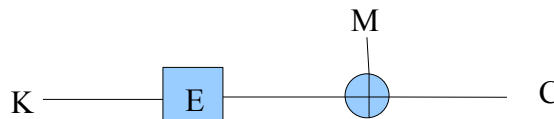


Stream Cipher

Stream Cipher, also known as state cipher for its dependence on the current state, is a symmetric key cipher where plaintext bits are combined with a pseudo-random cipher bit stream, typically by an XOR operation. A typical stream cipher encrypts plaintext one byte at a time, although it may be designed to operate on one bit at a time or on units larger than a byte at a time.

The stream cipher is inspired from the one-time pad (OTP). The difference is that a one-time pad uses a genuine random number stream as the key, whereas a stream cipher uses a random looking (pseudo-random) number stream. While OTP requires the key to be as long as the plaintext, the stream cipher reduces the requirement to just a seed (key) and the algorithm is used to generate a random looking stream (keystream) of the length of the plaintext.



Desirable Properties of Stream Ciphers:

1. The encryption sequence should have a large period i.e. the keystream should be large.
2. The keystream should approximate the properties of a true random number stream as close as possible.
3. Key(seed) should be sufficiently long to guard against brute-force attacks on the key.
4. It should not be possible to infer the seed from the keystream.
5. The algorithm should be efficient.
6. It should not be possible to predict the keystream from the known part of it (known as Next Bit Test)

Next Bit Test:

A Pseudo-random number generator is said to pass the next-bit test if there is no polynomial time algorithm that, on input of the first k bits of an output sequence, can predict the $(k+1)$ st bit with probability significantly greater than $\frac{1}{2}$.

Primary advantage of Stream Cipher over Block Cipher

1. Faster than block ciphers
2. Uses far less code than block cipher

Disadvantage:

1. Less secure than block cipher, but with a properly designed PRNG, a stream cipher can be as secure as block cipher of comparable key length.
2. Key can be reused in a block cipher but not in a stream cipher

Random Numbers:

Random numbers are extensively used in cryptographic protocols.

Examples:

1. Used as nonces to prevent replay attacks
2. Used in session key generation
3. Generation of keys for the RSA public key encryption algorithm

These application want the random numbers to have two, not necessarily compatible, properties:

1. **Randomness:** Randomness is based on the following two criteria:
 1. Uniform distribution: Numbers in the sequence should be uniformly distributed
 2. Independence: No value in the sequence can be inferred from the other.
2. **Unpredictability:** Successive numbers of the sequence should be unpredictable.

All true random sequences are unpredictable due to independence property. Cryptographic applications use sequences that are not truly random but appear to be random and satisfy unpredictability property. These Pseudo-random sequences are generated by [Pseudo-random Number Generators \(PRNGs\)](#). A pseudo-random number generator uses a function that produces a deterministic stream of bits that eventually repeats.

Examples of PRNGs:

Linear Congruential Generators

Proposed by Lehmer, the algorithm is parametrized with four numbers:

m	the modulus	$m > 0$
a	the multiplier	$0 < a < m$
c	the increment	$1 \leq c < m$
X0	seed	$0 \leq X_0 < m$

The following equation is used to generate the sequence of random numbers:

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

Example:

```
unsigned int state;
int E(void)
{
    state = 322349 *state + 45656749;
    return state % 2;
}
```

The selection of values for a, c and m is critical in developing a good random number generator. If they are properly chosen, then the generator will be a maximal period generator and have period of m.

If the multiplier and modulus are properly chosen, the resulting sequence of numbers will be [statistically indistinguishable](#) from a sequence drawn at random but the algorithm is deterministic and once the seed is chosen, all subsequent numbers follow deterministically.

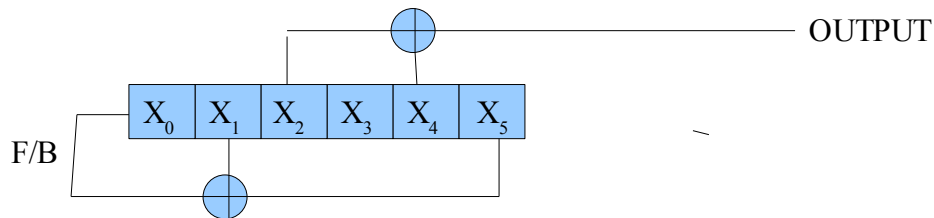
In its basic form, the LCG is not much useful cryptographically. If the parameters are known then once a number is discovered, all future numbers are known. Even if the parameters are not known,

knowledge of a few numbers in the sequence is sufficient to predict all the future numbers. A solution is to use the internal system clock to modify either the seed or the output random number, another is to combine many LCGs. Still LCGs are mainly useful for non-cryptographic applications.

Linear Feedback Shift Register

A feedback shift register(FSR) is made up of two parts: a shift register and a feedback function. In the LFSR, the feedback function is simply the XOR of certain bits in the register. LFSRs are the most common type of shift registers used in cryptography. LFSRs that cycle through all 2^n-1 internal states (where n is the bit-length of the shift register) are called maximal period LFSRs. The resulting output sequence is called an m-sequence.

Example:



In the example LFSR above, the XOR of X_2 and X_4 is considered as the output of the generator while the XOR of X_1 and X_5 is considered as the feedback. At each operation, the feedback bit becomes X_0 and each bit from X_0 to X_4 shifts one place to the right while X_5 falls off and is discarded.

LFSRs are used in cryptography mainly because they are easy to implement in hardware and are fast. Various techniques are used to make them cryptographically strong. They are, however, not so fast in software.

LFSRs have some non-random properties and so they have various successful cryptographic attacks on them. For an LFSR of length n , the internal state is the next n output bits of the generator. Even if the feedback scheme is unknown, it can be determined from only $2n$ bits of the generator.

Blum-Blum-Shub (BBS) Generator

[BBS](#) is referred to as a cryptographically secure pseudo-random bit generator (CSPRNG). A CSPRNG is defined as one that passes the [next-bit test](#).

Pick large primes p and q , compute $N=p*q$. So N is a composite number with only p and q as factors except universal factors N and 1 .

$$\text{Pick } X_0 \in \{2, 3, \dots, N-1\}$$

$$\text{Let } X_i = (X_{i-1})^2 \bmod N$$

$$\text{Output} = X_i \bmod 2 \text{ (least significant bit of } X_i)$$

Breaking BBS is equal to factoring N hence considering factoring of large primes to be a difficult problem, BBS is cryptographically secure. It has limited utility in applications due to its slow speed.

Probability Distribution

A probability distribution is a function that assigns a probability to each possible value in some set.

Example:

Set $\{0, 1\}$

$D(0) = 0.7$

$D(1) = 0.3$

Statement 'Draw X randomly from D ' is represented as $X \leftarrow D$.

Statistical Indistinguishability

Two distributions D & D' are ϵ – statistically indistinguishable if for all algorithms A ,

$$\text{Adv } A = | \Pr[A(X) = 1 \mid X \leftarrow D] - \Pr[A(X) = 1 \mid X \leftarrow D'] | \leq \epsilon$$

Statistical Indistinguishability puts no limit on the time taken or the computational power of the computer running algorithm A .

Computational Indistinguishability

Distribution D & D' are t, ϵ -computationally indistinguishable if for all algorithms A running in time $\leq t$,

$$\text{Adv } A \leq \epsilon$$

Computational indistinguishability puts a condition on statistical indistinguishability hence each statistically indistinguishable pair is also computationally indistinguishable but the opposite is not true.

Theorem 1: If D_1 and D_2 are t, ϵ_1 computationally indistinguishable and D_1 and D_3 are t, ϵ_2 computationally indistinguishable then D_1 and D_3 are $t, \epsilon_1 + \epsilon_2$ computationally indistinguishable.

Proof:

Let A be any algorithm running in time t , then:

$$\begin{aligned} & | \Pr[A(X) = 1 \mid X \leftarrow D_1] - \Pr[A(X) = 1 \mid X \leftarrow D_3] | \\ = & | \Pr[A(X) = 1 \mid X \leftarrow D_1] - \Pr[A(X) = 1 \mid X \leftarrow D_2] \\ & + \Pr[A(X) = 1 \mid X \leftarrow D_2] - \Pr[A(X) = 1 \mid X \leftarrow D_3] | \\ \leq & | \Pr[A(X) = 1 \mid X \leftarrow D_1] - \Pr[A(X) = 1 \mid X \leftarrow D_2] | \\ & + | \Pr[A(X) = 1 \mid X \leftarrow D_2] - \Pr[A(X) = 1 \mid X \leftarrow D_3] | \\ \leq & \epsilon_1 + \epsilon_2 \end{aligned}$$

Theorem 2 (Data Processing Inequality): If D and D' are t, ϵ computationally indistinguishable and f is any function computable in time t' then $f(D)$ and $f(D')$ are $t-t', \epsilon$ computationally indistinguishable.

Proof: (by contrapositive)

Suppose $\exists A$ running in time $t-t'$ such that $\text{Adv}_{f(D), f(D')} A > \epsilon$

$$\begin{aligned} & \Pr [A'(X)=1 \mid X \leftarrow D] \\ &= \Pr [A(f(X))=1 \mid X \leftarrow D] \\ &= \Pr [A(X)=1 \mid X \leftarrow f(D)] \dots\dots\dots (1) \end{aligned}$$

$$\begin{aligned} & \Pr [A'(X)=1 \mid X \leftarrow D'] \\ &= \Pr [A(f(X))=1 \mid X \leftarrow D'] \\ &= \Pr [A(X)=1 \mid X \leftarrow f(D')] \dots\dots\dots (2) \end{aligned}$$

From (1) and (2)

$$\begin{aligned} \text{Adv}_{D, D'} A' &= \left| \Pr [A'(X)=1 \mid X \leftarrow D] - \Pr [A'(X)=1 \mid X \leftarrow D'] \right| \\ &= \left| \Pr [A(X)=1 \mid X \leftarrow f(D)] - \Pr [A(X)=1 \mid X \leftarrow f(D')] \right| \\ &= \text{Adv}_A \\ &> \epsilon \end{aligned}$$

Which is contrary to our basic condition that $\text{Adv}_{D, D'} A \leq \epsilon$. Hence our assumption is wrong.