

Random Numbers and Cryptographically Strong Pseudo Random Numbers Generators

Anyone who considers arithmetical methods of producing random digits is, of course, in a state of sin.

-- John Von Neumann, 1951

Random Number Generators

- A sequence of random variables U_1, U_2, U_3, \dots that are statistically independent and each of which has a uniform probability density function on the interval of real numbers $(0, 1)$ is called a sequence of random numbers.
- Random Numbers are used in:
 - ⊗ simulation whether discrete event, continuous, Monte Carlo, etc.
 - ⊗ Cryptographic protocols – keys, challenges, etc.

Early Random Number Generators

- Early Approaches: Methods were carried out by hand and used to create tables of numbers.
- Examples: Casting lots, rolling dice, drawing numbered balls from a well-stirred urn, successive digits of p or e , last 4-digits of a group of telephone numbers, etc.
- With 1,000,000 digits, you can make 125,000 random 8-digit decimal numbers --- Not enough for a simulation!

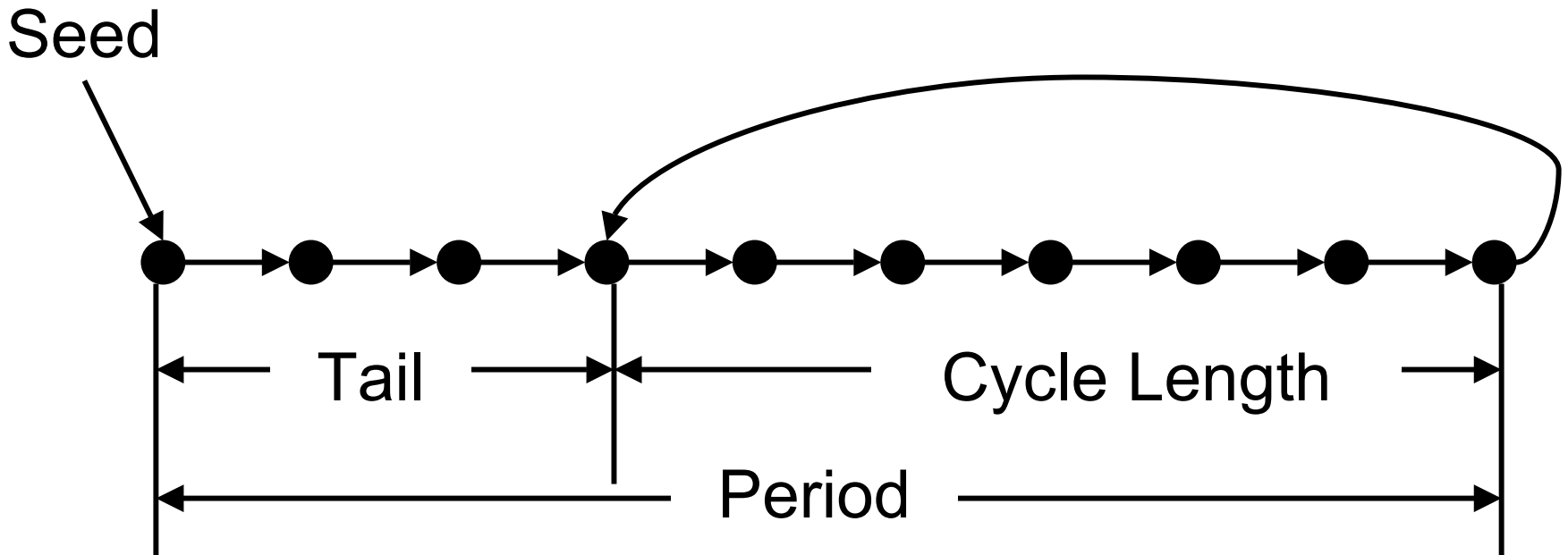
Computer Generated Random Numbers

- With the rise of computers, the focus shifted to numeric or arithmetic ways to generate random numbers.
 - ⊗ Not truly “random,” since they are not unpredictable. The whole sequence can be determined using the formula and the input values.
 - ⊗ Also called: pseudo random or “close-to” random
 - ⊗ The goal is to produce numbers that appear to be random.

Computer Generated Random Numbers Basics

- We will be discussing methodologies that will ultimately yield $U(0,1)$ distributions.
- All generators require a seed to begin the random number sequence.
- The values are considered to be pseudo-random, since the generation of the values is deterministic and allows us to repeat results.
- $x_n = f(x_{n-1}, x_{n-2}, \dots)$
- We will be concerned with the statistical properties of randomness and period and cycle length.

Period and Cycle Length



Desirable Generator Properties

- The routine should be fast.
- Should not require a large amount of memory.
- It should have a long cycle (a period of 1 billion is considered good by most)
- The random values should be repeatable.
- The values should closely approximate the ideal statistical properties of uniformity and independence.

A Little History

- Observational methods (telephone numbers, Rand Corp.) -- Time consuming.
- Use of transcendental numbers (e , π). -- Generation process is long.
- Mid-Square method, proposed by Von Neumann in the 1940's. The technique starts with a seed, the seed is squared and the middle digits become the random number.

Mid-Square Method

- Mid-Square method, proposed by von Neumann in the 1940's. The technique starts with a seed, the seed is squared and the middle digits become the random number.
- **Example:**
 - ⊗ $X_0 = 5497$
 - ⊗ $X_0^2 = (5497)^2 = 30,217,009 \Rightarrow X_1 = 2170$
 - ⊗ $R_1 = 0.2170$
 - ⊗ $X_1^2 = (2170)^2 = 04,708,900 \Rightarrow X_2 = 7089$
 - ⊗ $R_2 = 0.7089$
- Problems: difficult to assure that the sequence will not degenerate over a long period of time, zeros once they appear are carried in subsequent numbers (try 5197 as a seed).

Midproduct Generators

- Similar to the midsquare technique, starts by selecting two seeds each containing the same number of digits. Period of this technique is typically longer than the midsquare technique.

- **Example:**

- ⊠ seeds ... $X_0' = 2938, X_0 = 7229$

- ⊠ $U_1 = X_0'X_0 = (2938)(7229) = 21,238,802 \Rightarrow X_1 = 2388$

- ⊠ $R_1 = 0.2388$

- ⊠ $U_2 = X_0X_1 = (7229)(2388) = 17,262,852 \Rightarrow X_2 = 2628$

- ⊠ $R_2 = 0.2628$

Constant Multiplier Technique

- Variation of the midproduct technique using a constant multiplier.

- **Example:**

- ⊗ seed = $X_0 = 7233$, constant = $K = 3987$

- ⊗ $V_1 = (K)(X_0)$

- ⊗ $V_1 = (3987)(7223) = 28,798,101 \Rightarrow X_1 = 7981$

- ⊗ $R_1 = 0.7981$

- ⊗ $V_2 = (K)(X_1)$

- ⊗ $V_2 = (3987)(7981) = 31,820,247 \Rightarrow X_2 = 8202$

- ⊗ $R_2 = 0.8202$

Additive Congruential Method

- Requires a sequence of n numbers X_1, X_2, \dots, X_n and produces an extension of the sequence using the relationship:

$$\boxtimes X_i = (X_{i-1} + X_{i-n}) \bmod m, i = n+1, \dots$$

- **Example:**

$$\boxtimes \text{original sequence} = 57, 34, 89, 92, 16, n = 5, m = 100$$

$$\boxtimes X_6 = (X_5 + X_1) \bmod 100 = 73 \bmod 100 = 73$$

$$\boxtimes R_1 = X_6/100 = 0.73$$

$$\boxtimes X_7 = (X_6 + X_2) \bmod 100 = 107 \bmod 100 = 7$$

$$\boxtimes R_2 = X_7/100 = 0.07$$

\boxtimes If $n = 2$, this method is called the Fibonacci series method.

Mixed Linear Congruential Generators

- Initially proposed in the 1950's, produces a sequence of integers between 0 and $m-1$ using the relationship:

$$\boxtimes X_{i+1} = (aX_i + c) \bmod m$$

- where X_0 is the seed, a is the multiplier, c is the increment and m is the modulus (all non-negative).
- The selection of a , c , and m and the seed drastically effect the properties of the generator. The X 's produced are integers between 0 and $m-1$.
- **Example:** (seed = 27, $a = 17$, $c = 43$, $m = 100$)

$$\boxtimes X_0 = 27$$

$$\boxtimes X_1 = (17 \cdot 27 + 43) \bmod 100 = 502 \bmod 100 = 2$$

$$\boxtimes R_1 = 2/100 = 0.02$$

$$\boxtimes X_2 = (17 \cdot 2 + 43) \bmod 100 = 77 \bmod 100 = 77$$

$$\boxtimes R_2 = 77/100 = 0.77$$

Mixed Linear Congruential Generators Notes

- The modulus m should be large, since all X 's are between 0 and $m-1$, and the period can never be more than m .
- For the mod m computation to be efficient, m should be a power of 2.
- If c is non-zero, the maximum possible period m is obtained iff:
 1. integers m and c are relatively prime.
 2. every prime number that is a factor of m is also a factor of $a-1$.
 3. $a-1$ is a multiple of 4, if m is a multiple of 4.
- If $c = 0$, this generator is called a multiplicative congruential or power residue

Combined Generators

- It is possible to combine two or more generators to produce a “better” generator.
 - ⊗ Adding random numbers from two or more generators: $w_n = (x_n + y_n) \bmod m$
 - ⊗ Exclusive-or random numbers from two or more generators.
 - ⊗ Shuffling - uses one random number sequence as an index to decide which of several numbers generated by a second sequence should be returned.

The RANDU Generator

- Originally distributed with IBM mainframe computers in the 1960's as part of the scientific subroutine library.
- Was a multiplicative LCG:
- $x_n = (2^{16} + 3)x_{n-1} \bmod 2^{31}$
- When triplets of successive numbers were generated by the generator and plotted as points in three space, all the points were found to lie on a total of 15 planes. This specific generator has been shown to fail a number of the aforementioned tests.

Pitfalls to Avoid in Random-Number Generation

- A complex set of operations will not necessarily lead to random results - a sequence of operations where the final results are difficult does not necessarily mean the generator will pass a test for uniformity and independence.
- A single statistical test is not sufficient to show a generator is good. The generator may in fact fail other criteria.
- Avoid generators that have problems with certain seeds - for example, the generator works correctly for all seeds except $x_0 = 37911$, which will $x_n = (9806x_{n-1} + 1) \bmod(2^{17} - 1)$ make the generator stick at 37911 forever.

Pitfalls to Avoid in Random-Number Generation

- Accurate implementation is important - the period and randomness of generators are guaranteed only if the generation formula is accurately implemented without any overflow or truncation. Watch for overflows that may cause values to be represented as negative numbers.
- Bits of successive words generated by a random number generator are not randomly distributed - special techniques are required to generate random bits efficiently.
- It is better to use an established generator that has been tested than to invent a new one.

Need for Random bits in Cryptography

- One needs random bits (or values) for several cryptographic purposes, but the two most common are the generation of cryptographic keys (or passwords) and the blinding of values in certain protocols.
- Conventional random number generators available in most programming languages or programming environments are not suitable for use in cryptographic applications (they are designed for statistical randomness, not to resist prediction by cryptanalysts).

Cryptographic Random Number Generators

- In the optimal case, the Cryptographic Random Number Generators are based on true physical sources of randomness that cannot be predicted.
- Such sources may include the noise from a semiconductor device, the least significant bits of an audio input, or the intervals between device interrupts or user keystrokes. The noise obtained from a physical source is then "distilled" by a cryptographic hash function to make every bit depend on every other bit. Quite often a large pool (several thousand bits) is used to contain randomness, and every bit of the pool is made to depend on every bit of input noise and every other bit of the pool in a cryptographically strong way.

Cryptographic Random Number Generators

- When true physical randomness is not available, pseudorandom numbers must be used. This situation is undesirable, but often arises on general purpose computers.
- It is always desirable to obtain some environmental noise - even from device latencies, resource utilization statistics, network statistics, keyboard interrupts, or whatever. The point is that the data must be unpredictable for any external observer; to achieve this, the random pool must contain at least 128 bits of true entropy.
- Some machines may have special purpose hardware noise generators

Cryptographic Random Number Generators

- What one needs for cryptography is values which can not be guessed by an adversary any more easily than by trying all possibilities [that is, “brute force”].
- There are several ways to acquire or generate such values, but none of them is guaranteed.
- Therefore, selection of a random number source is a matter of art and assumptions, as indicated in the RFC on randomness by Eastlake, Crocker and Schiller[RFC1750].



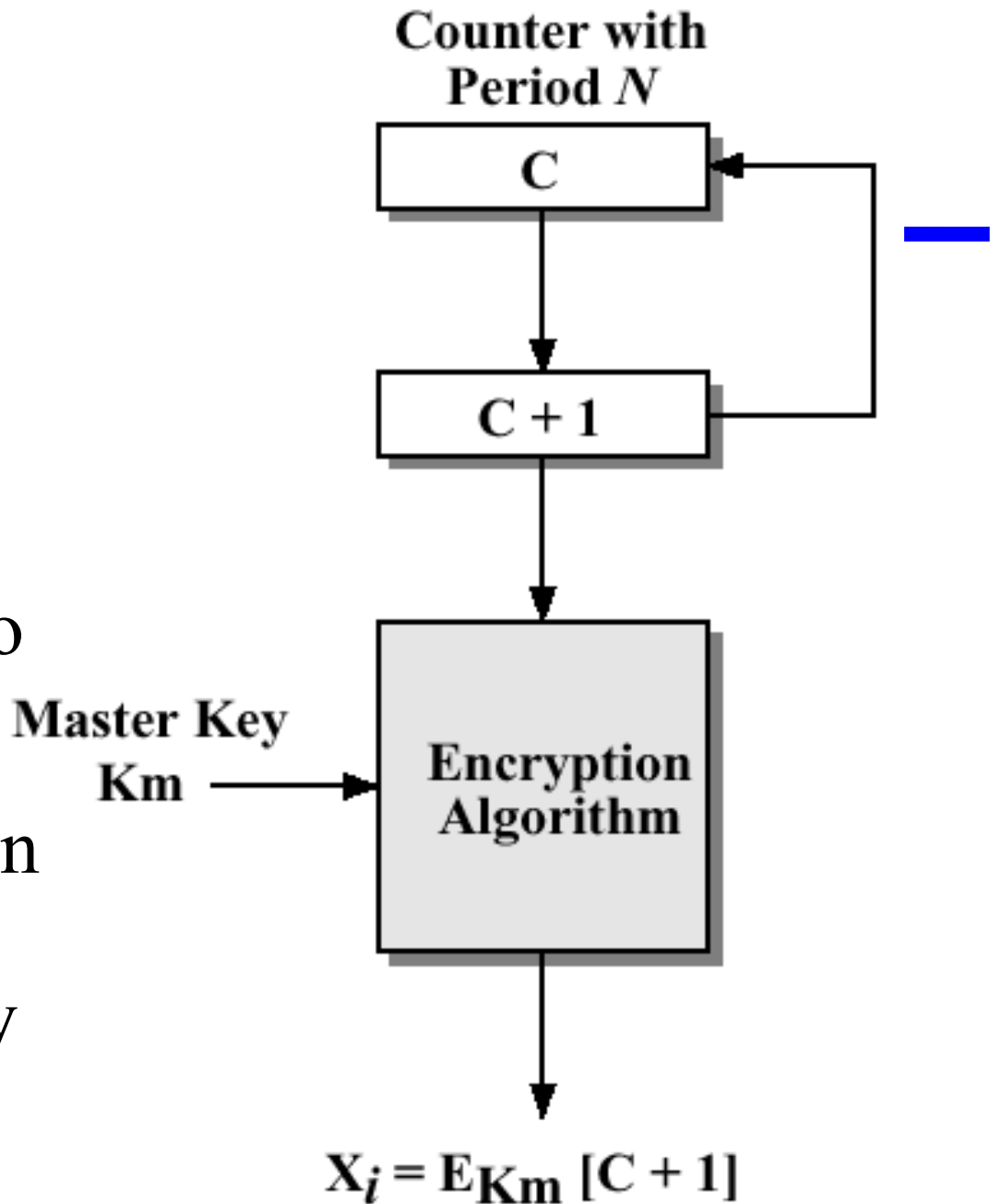
Criterion for a Random Source

- There are several definitions of randomness used by cryptographers, but in general there is only one criterion for a random source:
 - ⊗ *Any adversary with full knowledge of your software and hardware, the money to build a matching computer and run tests with it, the ability to plant bugs in your site, etc., must not know anything about the bits you are to use next even if he knows all the bits you have used so far.*

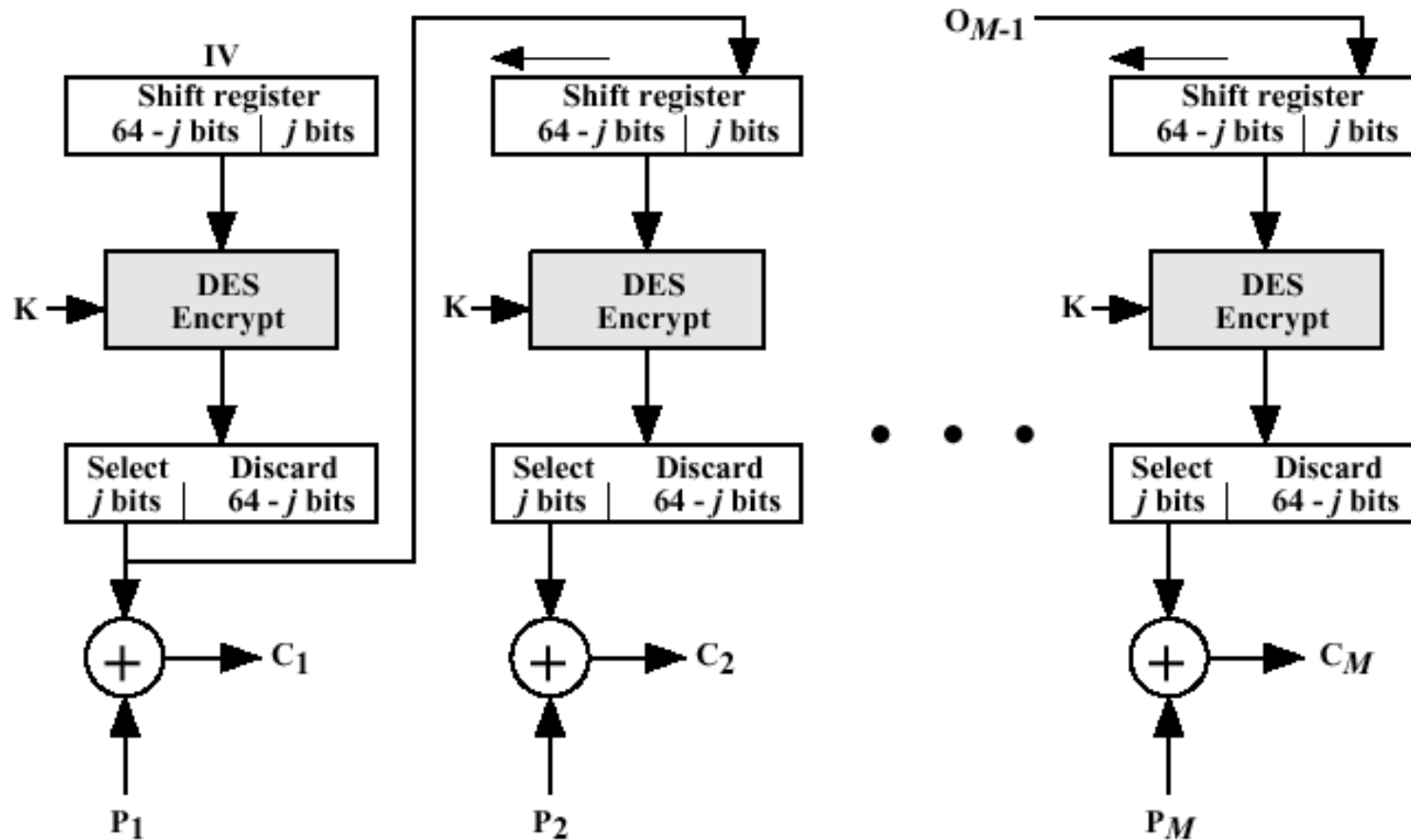
Using Encryption with a Counter

K_m is a master key which is not known to the adversary.

Even better results can be obtained if the counter is replaced by a PRNG with a full period.

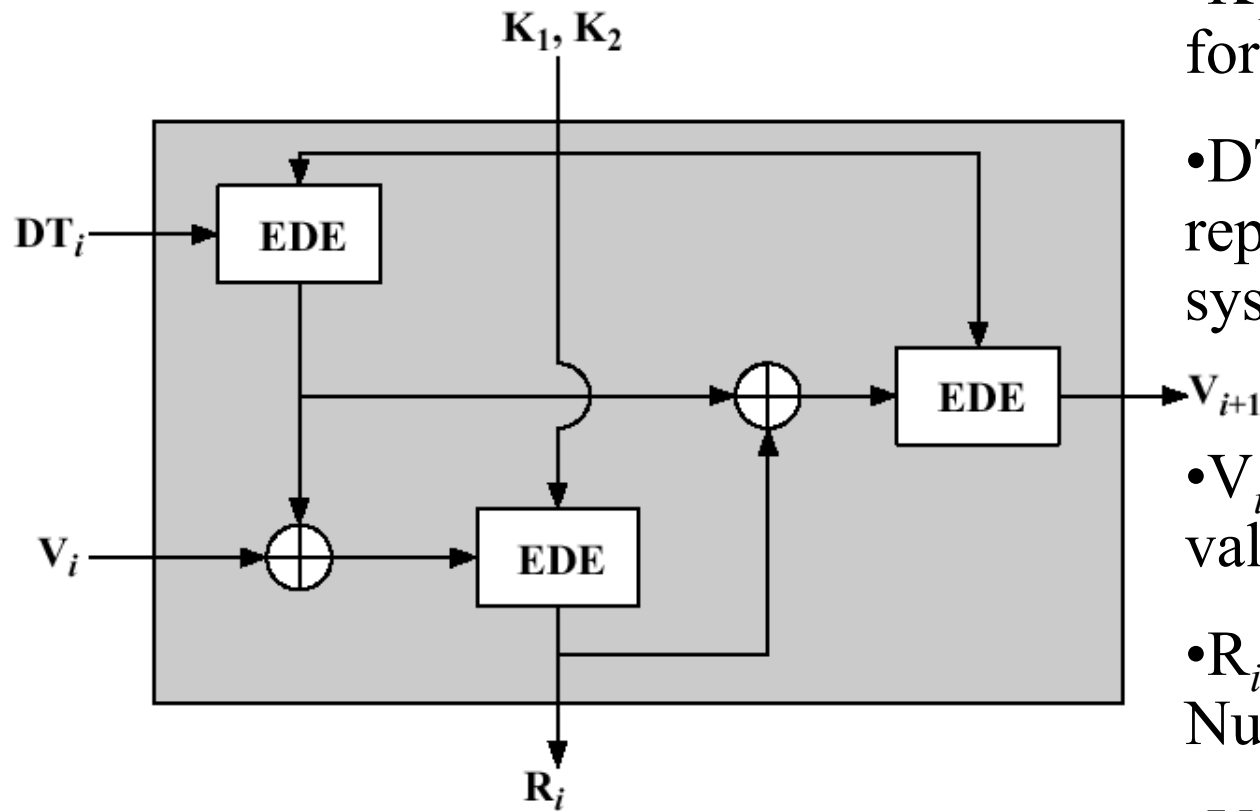


DES Output Feedback Mode - OFB



(a) Encryption

ANSI X9.17 Pseudorandom Number Generator



- K_1 and K_2 are two keys for 3DES

- DT_i is a 64 bit representation of current system date and time

V_{i+1}

- V_i is an initialization value (seed)

- R_i is the Random Number generated

- V_{i+1} is the initialization value for the next iteration

Blum Blum Shub - BBS

CSPRBG

- $p \equiv q \equiv 3 \pmod{4}$
- $n = p \times q$
- s is chosen to be relatively prime to n
 - ⊗ $X_0 = s^2 \pmod{n}$
 - ⊗ for $i = 1$ to ∞
 - $X_i = (X_{i-1})^2 \pmod{n}$
 - $B_i = X_i \pmod{2}$

Blum Blum Shub - BBS CSPRNG

Table 5.2 Example Operation of BBS Generator

s	X_i	B_i
0	20749	
1	143135	1
2	177671	1
3	97048	0
4	89992	0
5	174051	1
6	80649	1
7	45663	1
8	69442	0
9	186894	0
10	177046	0

s	X_i	B_i
11	137922	0
12	123175	1
13	8630	0
14	114386	0
15	14863	1
16	133015	1
17	106065	1
18	45870	0
19	137171	1
20	48060	0

The Last Word

- Even though cryptographically strong random number generators are not very difficult to build if designed properly, they are often overlooked.
- The importance of the random number generator must thus be emphasized - if done badly, it will easily become the weakest point of the system.
- Look at the Intel RNG paper.

