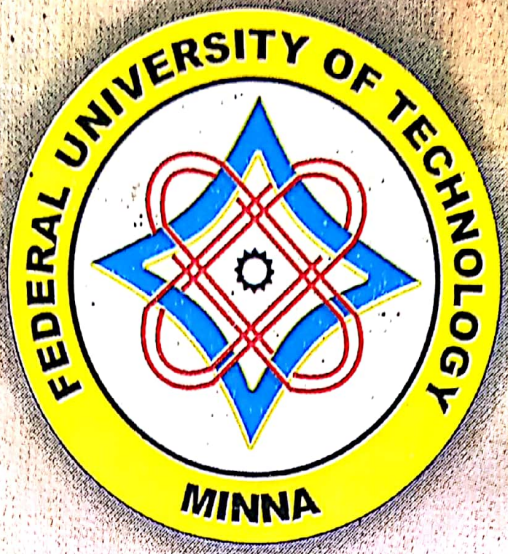
7TH ANNUAL ENGINEERING CONFERENCE

School of Engineering and Engineering Technology (SEET)

FEDERAL UNIVERSITY OF TECHNOLOGY (FUT), MINNA.



Book of Proceedings

Theme:

Health, Safety and Environment (HSE) in Engineering Practice

Date: 28th - 30th June, 2006

Edited By:

Dr. M.O. Edoga and Engr. Dr. M.G. Yisa

Generation of Random Numbers for Data Security Applications

Usman A. U. and Ajiboye J.A.
Dept. of Electrical & Computer Engineering,
Federal University of Technology, Minna.

Abstract

In this age of Electronic connectivity, the issue of data security is becoming more and more of great concern. The growth in computer systems and their interconnections via networks has increased the dependence of several organizations and individuals on information stored and communicated using these systems. Hence, there is need for data and resources to be well protected to guarantee its authenticity and to protect systems from network-based attacks. Cryptography and network security have matured, leading to the development of practical, readily available applications to enforce network security. This work covered a review of the concept of randomness with the stringent randomness requirement in data security systems giving particular attention to the Blum-Blum-Shub random number generator.

I INTRODUCTION

The requirements of information security within an Organization have undergone two major changes in the last several decades. Before the wide spread use of data processing equipment, the security of information felt to be valuable to an organization was provided primarily by physical and administrative means.

With the introduction of the computer, the need for automated tools for protecting files and other information stored on the computer become evident. This is especially the case of a shared system, such as a time-sharing system, and the need is even more acute for systems that can be assessed over a public telephone or data network. The generic name for collection of tools designed to protect data and to thwart hackers is computer security [Williams, 1995].

The second major change that affected security is the introduction of distributed systems and the use of network and communication facilities for carrying data between terminal user and computer, and between computer and computer. Clearly, security is required in any environment where information, data or items are not tended to be freely available to all during their transmission.

Today, security systems are built on increasingly strong cryptographic algorithms that foil pattern analysis attempts. However, at the heart of all cryptographic systems is the generation of secret, Unpredictable (i.e. random) numbers [Williams, 1995]. In other words good cryptography requires good random numbers. For example, random number generators are required to generate public/private key pairs for asymmetric (public key) algorithms including the Rivest-Shamir-Adelman (RSA), Digital signature Algorithm (DSA) and Diffie-Hellman [www.SSH-Tech-corner-cryptographic-Algorithms.htm and Knuth, 1982]. Keys for symmetric and hybrid cryptosystems are also generated randomly.

Random number generators are easily over looked and can thus become the weakest point of a cryptosystem. For any chain is only as strong as its weakest link [www.SSH-Tech-corner-cryptographic-Algorithms.htm]. Even a strong confusion generator can be made irrelevant if the system supports only a small number of keys, since simply trying all the keys (a key search attack), would be sufficient to penetrate such a cipher. Thus any real system based on keys must support enough keys to prevent this attack and that is an issue for the random number generator.

Actually Random numbers find frequent applications in several other fields of life. Researchers use random numbers for tackling a wide range of problems [Ivars, 1996, Knuth,1982, and Thierry,1997]. From modeling molecular behavior and sampling opinion to solving certain equations and testing the efficiency of algorithms. Such numbers also play crucial roles in a wide variety of games, including electronic versions of slot machines, lotteries, and other forms of gambling. [Ivars, 1996, Knuth,1982, and Thierry,1997] . Both Ivars, 1996 and Knuth,1982 present good materials on the History and advancement of such random numbers. But of special interest in this study is the random numbers suitable for cryptographic application.

Since security protocols rely on the unpredictability of the keys they use, random numbers for cryptographic applications must meet stringent requirements. The most important requirement is that attackers, including those who know the random number generator design, must not be able to make any meaningful predictions about the random number generator outputs. D. Eastlake etal, 1994 makes a good recommendation of other stringent requirements.

Unfortunately, there are so many random number generator designs and so many claims for them, that it is difficult even to compare the claims, let alone the designs. The most common approach is to compute random numbers by means of an algorithm or a formula o. These normally result in what is commonly named pseudo-random numbers. As a matter of fact, in the past, the Random number generation was mostly done by software mostly based on those algorithms. The resulting sequences from such systems typically don't meet all the criteria that establish randomness. Patterns often still remain in the sequence. After all, the computer simply follows a set procedure to generate the numbers, and restarting the process produces the same sequence. Moreover, the sequences eventually begin by repeating themselves of course.

However, as digital systems becomes faster and denser, it is possible, and sometimes necessary, to implement the generator directly in Hardware. Serious research is carried out in increasing measure on a truly (real) random source of data for random (unpredictable) numbers, such as values based on radioactive decay, atmospheric noise, thermal electrical noise and a fast, free running oscillator etc.

Today, some computers have a hardware component that functions as real random value generator. However, most computers still don't have a hardware that generates random numbers that is sufficiently random that an adversary cannot predict.

II RANDOMNESS DEFINED.

The Dictionary meaning of the term random is that which is done, chosen, etc without method or conscious choice. This points to haphazardness i.e that which is pattern less.

According to Thierry, 1997 there are many subjective perceptions about the term "randomness" but the exact definition has been debated depending on the application. Actually, it appears there can never be a straightforward definition of a random process. This is because, if a finite set of random events could be defined, then at least in principle, it could be built (or chosen) such as to comply with the definition. If so, it would not be random, because it would obey a certain rule used to build it. Therefore, the definition of randomness is intrinsically not possible.

In practice, this result in pseudo-definitions which all suffer from various hidden logical and functional defects. Some of these are circular, that is they use the term "random" or something perceptive equivalent or similar in order to define the term "randomness". For example Ritter, 1991 define randomness as "an attribute of the process which generates or selects "random" number rather than the

number themselves. Other definitions merely emphasize one or more statistical property of true random numbers.

A. Classical Probabilistic Perspective of Randomness

From the perspective of classical probability, any sequence of equally probable events is equally likely and thus equally "random" [David, 1991 and Knuth, 1982]. The sequence 1001 and 0000 or 1111 are equally probable with the probability of any of four bit outcome being 1/16, because there is 16 possible combinations namely; (0000, 0001, 0010, ---, 1111). Thus if origin in a probabilistic event were made the sole criterion of randomness, then both series would have to be considered random; and indeed so would all others, since the same mechanism generate all possible series.

The classical definition above allows one to speak of a process, such as the tossing of coin as being random. It does not allow one to call a particular outcome or string or sequence of outcome like obtaining ten heads in a row with twenty tosses of a fair coin, random. Ashish,2002 gives the classical probability notions of randomness based on Shannon's concept of entropy.

Deviating a little from this classical probability definition of randomness, a more sensible definition, which focused on the individual outcomes rather than on the generating process of string; is considered. It established hierarchy of degrees of randomness. The limitation that the definition cannot help to determine, except in a very special cases whether or not a given sequence is random is closely related to Kurt Godel's incompleteness theorem devised and proved in 1931 [Ashi,2002, Gregory, 1975 and Ritter, 1991].

B. Algorithmic Perspective.

This new definition of randomness has its heritage in information theory dating black to World War II [Gregory, 1975]. More often situations arise of providing as input a program, or string of instructions to a computer so that it produces a desired string as output. Such instructions given to computer must be complete and step without requiring that it comprehend the result of any part of the operations it performs.

The possibility of reducing redundancy by compressing such input string will mean a shorter sequence can serve as a code to reproduce the original string. Both Gregory, 1975 and Ashis, 2002, provides a practical example of compressible and incompressible data to be transmitted to a friend in another galaxy. A result of this is the canonical definition of randomness based on incompressibility, which was proposed independently about 1965 by A.N. Kolmogorov of the Academy of science of the U.S.S.R and G.J Chaitin and stated thus;

"a series of number is random if the smallest algorithm capable of specifying it to a computer has about the same number of bits of information as the series itself". It is on the basis of this definition that the two series of digits presented below are examined;

1010101010101010

01100010011001110010

whereas based on the classical probabilistic notions the two series can be considered "random", it is observed that the first sequence consist of patterns of 10 repeated ten times and as such could be specified to a computer by a very simple algorithm, such as "Print 10 ten times" extending the series by following the same pattern, might just change the programmer to say, "Print 10 a thousand times." The number of bits in such an increased algorithm is a small fraction of the number of bits in the series it specifies, and as the series grows larger the size of the program increases at a slower rate.

The second series of binary digits generated by flipping a coin 20 times and writing a "1" when the outcome was heads and a "0" when it was tails appears pattern less. There is no shortcut to reproduce it, as the shortest most economical algorithm for introducing the series into a computer would be "print 01100010011001110010." Which means necessarily the algorithm has to be expanded to the

corresponding size of a much longer pattern less series. This incompressibility is a property of all random numbers [Acho 2002, and Gregory, 1975]

III RANDOM NUMBER SOURCES

Most "random" number sources actually utilize a Pseudo Random Number Generator (PRNG). A PRNG is an algorithm with some state information that is the sole input. The generator is exercised by steps, and two things occur consistently during each step. There is a transformation of the state information, and the generator outputs a fixed size bit string. The generator seed is simply the initial state information. In other words it is a deterministic algorithm which, given a truly-random binary sequence of length n, outputs a binary sequence of length k(n) > n which appears to be random, k() being a polynomial.

With any PRNG, after a sufficient number of steps, the generator comes back to some sequence of states that was already visited before (a cycle). Then, the period of the generator is the number of steps required to do one full cycle through the visited states. The actual entropy (unpredictability) of the output can never be greater than the entropy of the seed.

Ari etal, 2000 referenced a large body of literature on the design and properties of PRNGS. Knuth, 1982 has a classic exposition on pseudo-random numbers. Applications he mentions include simulation of natural phenomena, sampling, and numerical analysis, testing computer programs, decision making aesthetics and recreation. None of these have the same characteristics as the sort of security uses in cryptography. Only in the last three could there be an adversary trying to find the random quantity in use.

Thierry,1997 classified random number generators based on their applications to include;

-The toy generators that are provided by most programming language, and many software pages, which he said, are to be considered suspicious for most serious application;

-The serious generators, that is generators with internal state information using at least 64 bits and with empirical and/or theoretical justification and;

-The truly random generators i.e. electronic circuits that employ measurements of a natural phenomenon to provide totally unpredictable draws.

A close examination of a few most popular pseudo-random number generators is what follows.

A Linear Congruential Generators (Lcg).

The Linear Congruential Generator (LCG) is by far the most, widely used technique for random number generation. The LCG is one of the oldest, and still the most common type of RNG implemented for programming language commands (e.g. the standard rand () function for C and ANSI C used by VAX C, RANDU introduced by IBM in 1963, MTHSRANDOM used by VAXFOTRAN, and VAX Basic etc). These are all built of LCG algorithm. David,1991 presents a good description of techniques for analyzing outputs of such RNGs.

The LCG algorithm is parameterized with four numbers, as follows:

m the modulus $m > 0$

a the multiplier $0 \leq a < m$

c the Inferement $0 \leq c < m$

Vo the starting value or seed $0 \leq Vo < m$

It is a modular arithmetic where the (n + 1) th value is obtained via the following iterative equation:

$$Vn + 1 = (a \cdot Vn + c) \bmod m \quad \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots (1)$$

If m, a, a and V0 are integers, then this technique will produce a sequence of integers with each integer in range $0 \leq Vo < m$.

The selection of values for a, c and m is critical in developing a good LCG random number generator. For example with m = c = 1, the sequence produced is obviously not satisfactory.

Now with a = 7, c = 0, m = 32 and Vo = 1, this generates the sequence (1,7,17,23,1,7,etc), which is also unsatisfactory. The reason is that of the 32 possible values only four is used; thus the sequence is said to have a period of 4.

A change in the value of "a" to 5, result in the following sequence; (1,5,25,29,17,21,9,13,1,5,...), which increases the period to 8.

A larger m results in an increased potential for producing a long series of distinct random numbers. A common criterion is that m is nearly equal to the maximum representable nonnegative integer for a given computer [David,1991 and Horwitz and Hill, 1989]. This value of m near to or equal to 2^{31} is typically chosen. The strength of the linear-congruential algorithm is that if the multiplier and modulus are properly chosen, the resulting sequence of numbers will be statistically indistinguishable from a sequence drawn at random (but without replacement) from the set 1,2……., m-1. but there is really nothing random at all about the algorithm, apart from the choice of the initial value V0. once that value is chosen, the remaining numbers in the sequence follow deterministically. This of course has implication for cryptanalysis.

The real problem with most LCGs is their tiny amount of internal state, simple step formula, and complete exposure. If an enemy knows that LCG is being used, and if the parameters are know (e.g) a = 1024, c ≠ 0, m= 2^{31}-1), then once a single number in the series is discovered, all subsequent numbers are known. Even if a few values from the sequence becomes available for analysis, the formula (i.e the parameters of the algorithm) can be deduced, the sequence reproduced, and the cryptosystem penetrated [Ritter, 1991].

To make the actual sequence used "non-reproducible", so that the knowledge of part of the sequence on the part of an opponent is insufficient to determine future elements of the sequence. Bright and Enison, 1979 suggest using an internal system clock to modify the random number stream. One way to use the clock would be to restart the sequence after every N numbers, using the current clock values (mod m) as the new seed. Another way would be to simply add the current clock value to each random number (mod m) [Williams, 1995].

But designing such portable application code to generate unpredictable numbers based on such system clocks is particularly challenging because the system designer does not always know the properties of the system clocks that the code will execute on. So if the code is to be employed across a variety of computer platforms and systems. It becomes a problem. Thus, because it is difficult to find a good set of LCG parameters, LCGs are normally difficult to customize – Not only have linear congruential generators been broken, but all polynomial congruent generators, such as quadratic generators and cubic generators, have also been broken [Eastlake etal, 1994].

B Linear Feedback Shift Register (Lfsr).

A feedback shift register consist of an ordinary shift register made up of m flip-flops and a logic circuit that are interconnected to form a multiloop feedback circuit. The flip-flops in the shift register are synchronously clocked. At each pulse of the clock, the state of each flip-flop is shifted to the next one down the line. With each clock pulse that logic circuit computes a Boolean function of the states of the flip-flops. The result is thereby feedback as the input to the first flip-flop, there by preventing the shift register from emptying. The sequence so generated is determined by the length m of the shift register, its internal state, and the feed back logic [Simon,1994]. With a total of m flip-flops, the number of possible state of the shift register is at most 2m.

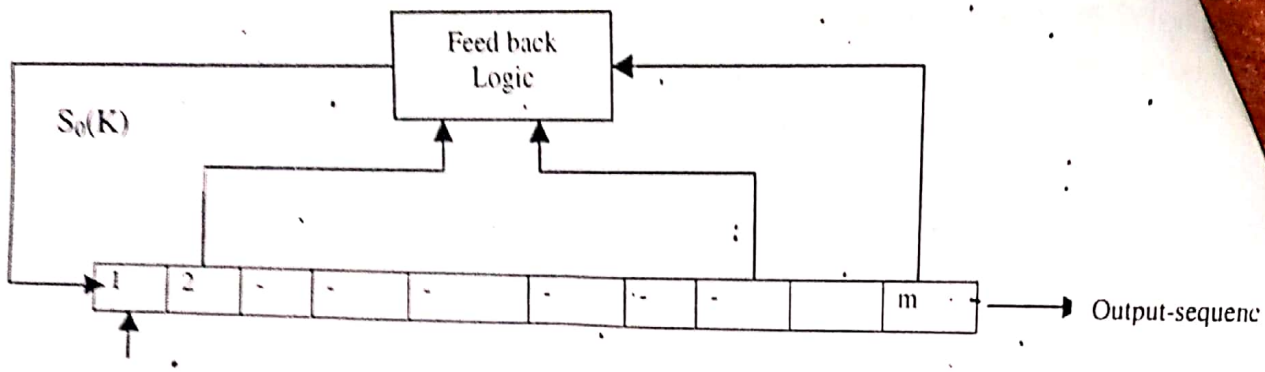


Figure 3.0 LFSR Pseudorandom number

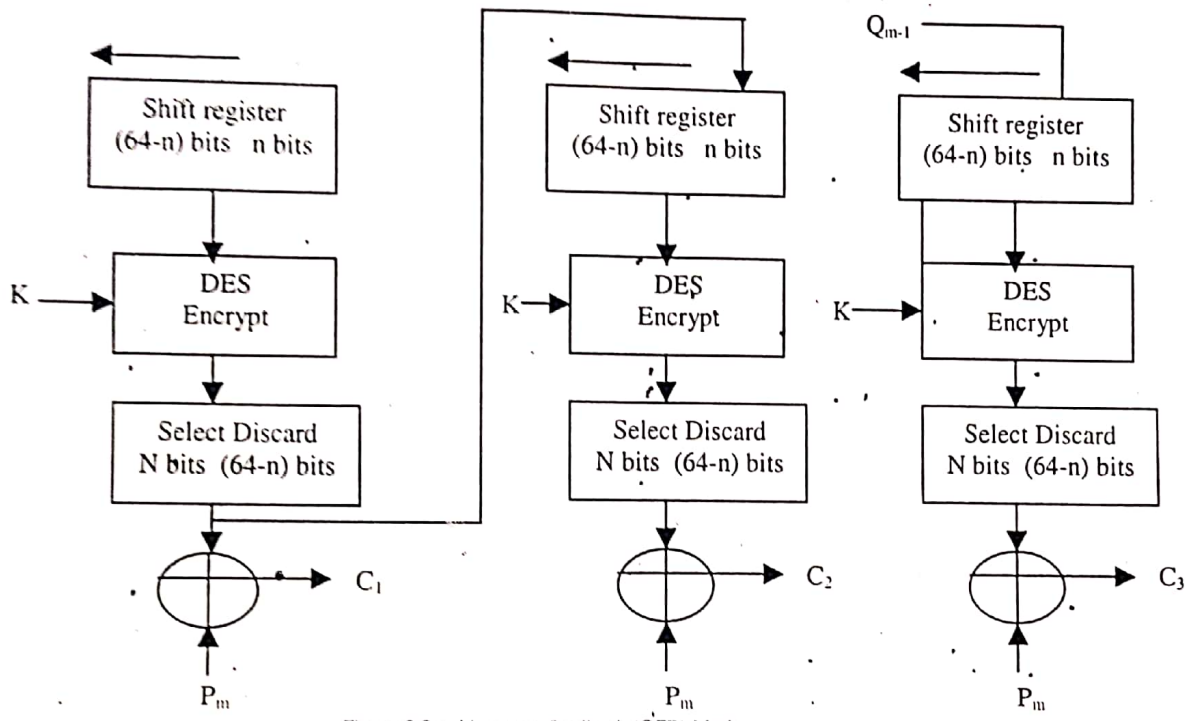


Figure 3.2: n bit output feedback (OFB) Mode.

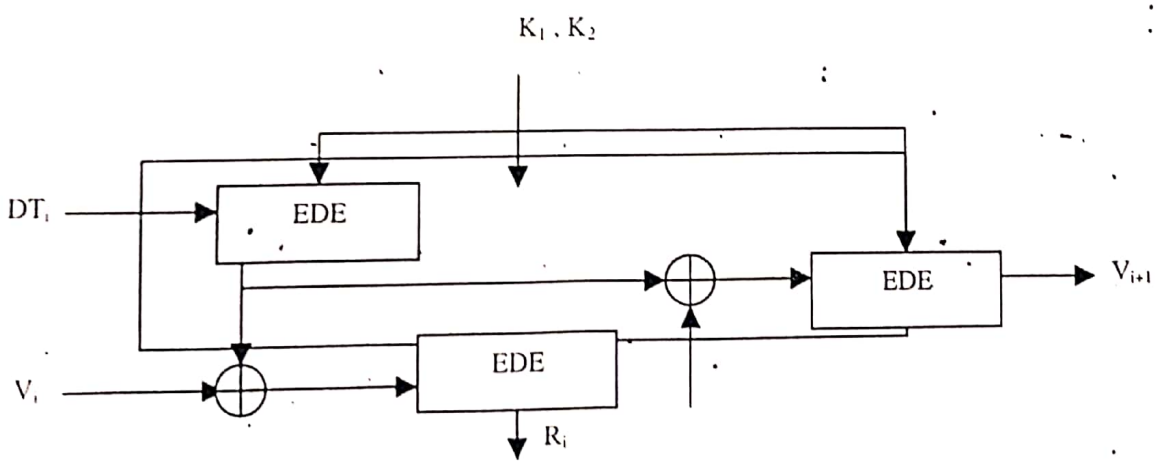


Figure 3.3.: Ansi X9.17 Pseudo-Random Number Generator.

A feedback shift register is said to be linear when the feedback logic consists entirely of modulo – 2 adders. (Modulo-2 addition is explained in [Ritter,1995-2003]. In such a case the zero state (e.g. the state for which all the flip-flops are in state 0) is not permitted. (The all zero's state will lock up in a degenerate cycle). As a result a maximum-length sequence of 2m – 1 is produced by a LFSR.

Maximal-length sequences have many of the properties possessed by a truly random binary sequence. But a maximal length sequence will occur if the shift-register "taps" from a polynomial that is primitive [Ritter, 1991]. (A primitive is a special kind of irreducible prime of a given finite field). The statistical performance of an LFSR with a primitive feedback polynomial really is quite interesting. Observation of bits produced by the feedback presented by Horwitz and Hill, 1989 shows that;

i. In each period of a sequence, the number of 1s is always one more than the number of 0s, which implies 1s or 0s are almost equally likely.

ii. Among the runs of 1s and of 0s in each period of a sequence, one half the runs of each kind are of length one, one-fourth are of length two, one-eight are of length three, and so on as long as these fractions represent meaningful numbers of runs. By a "run" we mean a subsequence of identical symbols (1s or 0s) within one period of the sequence. The length of this subsequence is the length of the run. So each of these sequence is also equally likely: and,

iii. There is only a period of length (2m-1) steps. The all – zeros state is an isolated and degenerate cycle.

With this understanding, the design of a maximal-length sequence generators reduces to finding the feedback logic for a desired period. The task is made particularly easy by virtue of the extensive tables of the necessary feedback connection for various shift-register length that have been computed in the literature. [Horwitz and Hill, 1989 and Simon, 1994].

As the length of shift register m, or equivalently, the period of the maximal sequence is increased, the sequence becomes increasingly similar to the random binary sequence [Simon, 1994]. Indeed in the limit, the two sequences become identical when m is made infinitely large. However, the price paid for making m large is an increasing storage requirement, which imposes a practical limit on how large m can actually be made. But of course, the LFSR can be made arbitrarily large (and thus more difficult to solve), and is also easily customized. Yet Ritter,1991 recommends additional isolation in cryptographic use.

The mathematical basis for these sequence was described by Tauworthe and is cited by Ritter,1991 as being a "linear recursion relation".

$a(k) = c(1) * a(k-1) + c(2) * a(k-2) + \ldots + c(m) * a(k-m) \pmod 2$ where a(k) is the latest bit in the sequence, c the coefficients or binary numerical factors a mod 2 polynomial, and m the degree of the polynomial and thus the required number of storage elements. This formula is particularly interesting, for relatively minor generalization of the same formula produced the Generalized Feedback Shift Register (GFSR) and additive generators described in [Ritter,1991].

C Cryptographycally Generated Random Numbers.

For cryptographic applications, it makes sense to take advantage of the encryptions logic available to produce random numbers. A number of means have been used, and a few representative examples are considered below.

D Cyclic Encryption.

Figure 3.1. Illustrates an approach suggested in [Meyer and Matyas, 1982]. In this case, the procedure is used to generate session keys from a master key. A counter with period N provides input to the encryption logic. For example, if 56-bit DES keys are to be produced, then a counter with period 256 is used. After each key is produced, the counter is incremented by one. Thus the pseudo-random numbers produced by this scheme cycle through a full

period: Each output V_0, V_1,.....V_{N-1} is based on a different counter value, and therefore $V_0 \neq V_1 \neq \ldots\ldots \neq V_{N-1}$.

Since the Master key is protected, it is not computationally feasible to deduce any of the secret keys through knowledge of one earlier key.

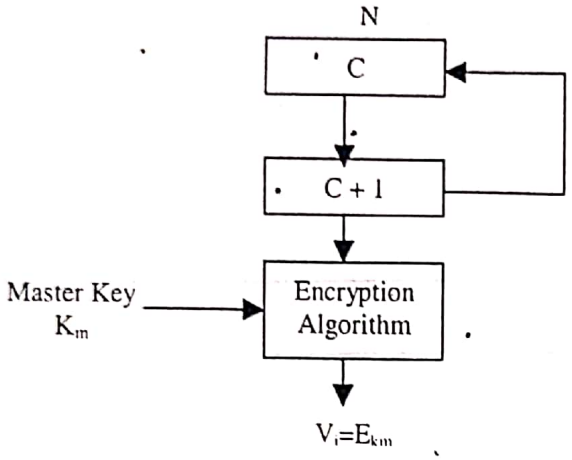


Figure 3.1: pseudorandom number generation from a counter.

A further way to strengthen the algorithm will be to ake the input from the output of a full period pseudorandom umber generator, rather than a simple counter.

Des Output Feedback Mode.

During 1968 – 1975, IBM developed a ryptographic procedure that enciphers a 64 – bit block of laintext into a 64 – bit block of cipher text under the ontrol of a 56-bit key. The National Bureau of Standards now the National Institute of standards and Technology NIST) accepted this algorithm as a Federal Information Processing Standard 46 (FIPS PUB 46) and it became ffective on July, 15 1977 [15,18].

The algorithm for this most widely used encryption cheme; the Data Encryption Standard (DES) is described in etail in [6,15,16,and 18]. Stallings [18] presents the ermutation table for the scheme. The four operation modes efined for different applications of DES are: Electronic odebook (EBC), Cipher Block chaining (CBC), Cipher eedback (CFB) and the output feedback (OFB) modes.

The output feedback (OFB) mode of DES illustrated in Figure 3.2 below can be used for key generation as well as for stream encryption. In the figure, it is assumed that the unit of transmission is n bits; a common value is n = 8 [Ritter, 1995]. The input to the encryption function is a 64-bit shift register that is initially set to some initialization vector (IV).

It is seen that the output of each stage of operation is a 64-bit value, of which the n leftmost bits are fed back for encryption. Successive 64-bit outputs constitute a sequence of pseudo-random numbers with good statistical properties [Ritter, 1995 and Simon, 1994]. Here also, the use of a protected master key protects the generated session keys.

A particular implementation of the cipher feedback (CFB) mode recommended by the national bureau of standards for the generation of cryptographic bit stream is shown in [Ritter, 1995]. David, 1991 claims the implementation of this scheme in software is too slow.
(C1 is the result of X) Ring the left most (most significant) n bit of the output of the encryption with the first unit of plaintext.)

F ANSI X9.17 Pseudo-Random Number Generator.

Specified in the ANSI financial institution key management standard ANSI X9.17, is one of the strongest (cryptographically speaking) pseudo-random number generator [Eastlake, 1994]. A number of applications employ this technique, including financial security applications and Pretty Good Privacy (PGP). PGP is the effort of Phil Zimmer man described in [Williams, 1995]. Figure 3.3 illustrates the algorithm, which makes use of triple DES for encryption. The scheme involves:
-Input: Two pseudorandom inputs that drive the generator. One is a 64-bit representation of the current date and time, which is updated on each number generation. The other is a 64-bit seed value. This is initialized to some arbitrary value and is updated during the generation process.

-Keys: The generator makes use of three triple DES encryption modules. All the three make use of the same pair of 56-bit keys, which must be kept secret and are used only for pseudo-random number generation.

-Outputs: The output consists of a 64-bit pseudorandom number and a 64-bit seed value.

By definition;

DT1 is the Date & Time value at the beginning of ith generation stage;

Vi is the seed value at the beginning of ith generation stage;

Ri the pseudorandom number produced by the ith generation stage;

K1i K2 are the DES keys used for each stage.

Therefore;

$Ri = EDE\ k1, k2\ (EDEk1,k2\ (DT1) \oplus V1)$

$Vi+1 = EDEk1,k2\ (EDEk1,k2\ (DT1) \oplus R1)$

The cryptographic strength of this method depends on several factors. The technique involves a 112-bit key and 3 Encrypt-Decrypt-Encrypt (EDE) encryptions for a total of 9 DES encryptions [Horwitz and Hill, 1989]. The scheme is driven by 2 pseudo-random inputs, the Date and time value, and a seed produced by the generator that is distinct from the pseudorandom number produced by the generator because an additional EDE operation is used to produce the Vi+1 the knowledge of Ri is still not enough to produce the Vi+1 from Ri. Thus the amount of material that must be compromised by an opponent is overwhelming.

What follows is the design and implementation of the Blum, Blum Shub RNG which is claimed to be suitable for most cryptographic applications.

4.0 THE BLUM, BLUM SHUB RANDOM NUMBER GENERATOR

The Blum, Blum Shub (BBS) generator is a random number generator named after its inventors, which is based on quadratic residues. The initial seed for the generator S (O) and the method for calculating subsequent values are based on the following iterations;

$S(O) = (X^2) \bmod N$

$S(i+1) = S(i)^2 \bmod N$

Where N is the product of two large primes. X is chosen at random to be relatively prime to N and the output is the least significant bit of S (i) or the parity of S(i). Or, the output can be several of the least significant bits of S (i) up to $\log_2 (\log_2 N)$ bits [Eastlake,1994, Ritter, 1991 and Wikipedia free encyclopedia].

This generator seems unique in that it is claimed to be "polynomial-time unpredictable" and cryptographically strong though not suitable for use in simulations because of its slow nature [Ritter, 1991 and Wikipedia free encyclopedia]. The BBS generator is not a permutation generator like some digital computer RNGs discussed earlier.

All digital computer RNGs including the BBS necessarily repeat eventually, and may well include many short or degenerate cycles. Thus the generator requires some fairly-complex design procedures, which are apparently intended to assure long cycle operation. It has an unusually strong security proof, which relates the quality of the generator to the difficulty of integer factorization [Eastlake,1994, Ritter, 1991 and Wikipedia free encyclopedia].

Given two large prime numbers, it is easy to multiply them together. However, given their product, it appears to be difficult to find the nontrivial factors of a large integer. This is relevant for many modern systems in cryptography. If a fast method were found for solving the integer factorization problem, then several important cryptographic systems would be broken, including the RSA public-key algorithm, and the Blum Blum Shub random number generator.

Although fast factoring is one way to break these systems, there may be other ways to break them that don't involve factoring. So it is possible that the integer

factorization problem is truly hard, yet these systems can still be broken quickly. A rare exception is the Blum Blum Shub generator. It has been proved [Pascal, 1999] to be exactly as hard as integer factorization. There is no way to break it without also solving integer factorization quickly.

If a large, n-bit number is the product of two primes that are roughly the same size, then no algorithm is known that can factor the number in polynomial time. When the primes are chosen appropriately and care is taken that only a few lowest order bits of each S (i) are output, then in the limit as N grows large, distinguishing the output bits from random will be at least as difficult as factoring N.

A Design Requirements For The Bbs Generator

The basic BBS requirement for N = P * Q is that P and Q each be primes congruent to 3 mod 4 (this guarantees that each quadratic residue has one square root which is also a quadratic residue) [Wekipedia free encyclopedia]. This looks exceeding easy. But to guarantee a particular cycle length, there are two more conditions:

Condition 1 defines that a prime P is "special" if:

P = 2 P1 + 1 and

P1 = 2 * P2 + 1 where P1 and P2 are odd primes. Both P and Q are required to be "special".

The original BBS paper as cited by Ritter, 1991 gives 2879, 1439, 719, 179, and 89 as examples of special primes (but 179 and 89 appear not to be special while 167, 47 and 23 should be). Namely;

If P = 179 from the above condition

179 = 2 * P1 + 1

thus P1 = 178/2 = 89

89 = 2*P2+ 1 hence P2 = 88/2 = 44 Thus whereas 89 is an odd prime, 44 is not.

Similarly for P = 89 = 2* P1 + 1 implies P1 = 88/2 = 44 which is not an odd prime.

Appendix two contains a module that generate primes that meet these requirements of P and Q being primes congruent to 3 mod 4 and being "special" primes. Figure 4.0 is the result of running the program.

Condition 2 says that, only one of P1, Q1 may have 2 as a quadratic residue (P1, Q1 are the intermediate values computed during the "special" certification). Accordingly, N = 719 * 47 fails this additional condition, because the intermediate values of both special primes have 2 as a quadratic residue.

Each of the special conditions provides additional structure in the generator, which presumably was needed for some aspect of mathematical proof. Currently, it is thought that the BBS special primes construction is sufficient, provided X is chosen not on a degenerate cycle, which is easily checked. On whether the special primes guarantee that the RNG will not have short cycles, Ritter, 2001 points out that with public key size special primes, the "short" cycles will either be "long enough" to use, or degenerate (i.e single – cycle loops).

As an illustration, the BBS system of P = 23, Q = 47 (N = 1081) is considered, a system which according to Ritter, 1991 was specifically given as an example of the prescribed form" in the original BBS paper. Starting with X = 46,

$S(0) = (46)^2 \bmod 1081 = 1035$

$S(1) = (1035)^2 \bmod 1081 = 1035$ ... a degenerate cycle.

With X = 47, 47 is repeated which is also a degenerate cycle.

Starting with X = 48, the cycle begin to repeat after the tenth cycle.

Because BBS generators generally define multiple cycles with various numbers of states, the initial value X must be specially selected as well. According to Eastlake etal, 1994 it must be relatively prime to N.

The random bit returned is obtained as;

$F[S_{(0)}] = (Z_0, Z_1, \ldots, Z_t)$, where $Z_i = S_i \text{ MOD } 2$, (i=0,1,2...t)

5.0 CONCLUSION.

Typical RNG's found in most computer libraries are not sufficiently strong for security purposes. Whereas hardware RNG's produce truly (real) random numbers, they

are not easy to come by. So in most applications Pseudo-random number generators are used.

A description of what randomness entails is given and the stringent randomness requirement in cryptography explored. A few random number generators were reviewed and a particular attention was given to Blum Blum Shub generator.

The BBS construction is not a maximal length RNG, but instead defines a system with multiple cycles, including degenerate, short and long cycles. With large integer factors, state values on short cycles are very rare, but do exist. Short cycles are dangerous with any RNG, because when a RNG sequence begins to repeat, it has just become predictable, despite any theorems to the contrary. Consequently, if the BBS is keyed by choosing X at random, unknowingly a short cycle (a week key) may be selected, which would make the sequence predictable as soon as the cycle starts to repeat. That of course is a direct contradiction to the idea that BBS is proven to be universally secure. Just knowing the length of a cycle (by finding sequence repetition) should be enough to expose the factors. This is evidence that the assumption that factoring is hard is not universally true. Of course, factoring is not hard when the factors are given away. With the special prime's construction, apparently all "short" (but not degenerate) cycles are "long enough" for use. The BBS is very slow in comparison to other RNG's, thus selecting BBS RNG clearly implies a decision to pay a heavy price with the expectation of getting an RNG which is "proven secure" in practice.

REFERENCES

Ari , Marcus I , Elizabeth S , and Bruce K. H , How to turn loaded dice into fair coins, IEEE Transaction on information theory, 46(3), May 2000

Ashis P Information, uncertainty and Randomness Algorithmic Perspectives, IEEE Potentials, Oct/Nov 2002

Bright H , and Enison R , Quasi-Random Number Sequences from Long-period TLP Generator with Remarks on Application to Cryptography, Computing Surveys, De , 1979

Cryptography A-Z, www.SSH Tech.com/ cryptographic Algorithms.htm

David W D , Computer Generated Random Numbers, 1991

D. Eastlake, S. Croker, J.Schiller, Randomness recommended for security, Dec , 1994

Gregory J.C., Randomness and Mathematical proof, Scientific American 232(5), May 1975, Pgs 47-52

Horwitz and Hill, The Art of Electronics, 2nd Edition, Pgs 655 664

Ivars Peterson's Mathland, A catalogue of Random bits, April, 22, 1996

Knuth D., The art of computer programming, Semi numeric algorithms Addison Wesley Publishing Company, 2nd Edition, Chapter 3, 1982.

Meyer C., and Matyas S., Cryptography: A new Dimension in Computer Data Security, New York: Wiley, 1982

Pascal J., Cryptographic Secure Pseudo-Random Number Generation The Blum-Blum-Shub Generator, August 1999

Ritter T., The efficient generation of Cryptographic confusion sequences Cryptologia, 15(2): 1991, Pgs 81-139

Ritter T., Simple Seed Selection in BB&S, June 2001

Ritter T., Ritter's CrytpoGlossary and Dictionary of Technical Cryptography, 1995, 2003.

Simon H., Communication Systems, 3rd Edition, John Wiley and Sons Inc , 1994, Pgs 578-586; 815-836.

Thierry M., Pseudo-Random Generators, a high level survey in progress, CONNOTECH Experts-Conseils Inc , March 1997

Williams S., Network and internetwork security (Principles and Practice) prentice Hall Inc., 1995.

Wikipedia, The Free Encyclopedia.