Secure Key Generation for Cryptography

S.Natarajan
Department of Computer Science and Engineering
National Engineering College
K.R.Nagar,Kovilpatti 628503
natarajans9493@gmail.com

S.Dheenathayalan
Assistant Professor
Department of Computer Science and Engineering
National Engineering College
K.R Nager,Kovilpatti 628503.
talk2dheena@gmail.com

Abstract----**Random numbers are needed in a variety of applications. But still finding good random numbers is a difficult task. Several deterministic algorithms also used for producing random numbers. The quality of the results of these methods critically depends on the quality of the random sequence. Computational efficiency is also an important aspect when very long sequences of random numbers have to be produced. Since real random numbers can be obtained from some physical phenomena only, they are difficult to be employed in real applications. In this research work, Cellular Automata are used to design a symmetric key cryptography system. CA are applied to generate a pseudo-random numbers sequence which is used during the encryption process. The mathematical approach for proposed methodology over the existing maximum length CA emphasizes on better flexibility to fault coverage. The randomness quality of the generated patterns for the proposed methodology has been verified using Diehard Tests**

Keywords-—PRNG; Life Like Cellular Automata; Diehard Tests.

I. INTRODUCTION

Production of random patterns, plays a fundamental role in the field of research work varying from Computer Science, Mathematics or Statistics to cutting edge VLSI Circuit testing. Random numbers are also used in cryptographic key generation and game playing. Mathematicians describe that this random numbers happen in a sequence where the values are homogeneously distributed over a well-defined interval, and it is unfeasible to predict the next values based on its past or present ones. In probabilistic approach, a random number is defined as a number, chosen as if by chance from some precise distribution such that selection of a large set of these numbers reproduces the fundamental distribution. Those numbers are also required to be independent to maintain no correlations between successive numbers. In practice, random number generator requires a specification of an initial number used as the starting point, which is known as a "seed". Random number generators are classified into several groups based on the difference in generation procedures of random numbers. Pseudo-random number and true-random number are most commonly used in scientific works. The classification of the random number is based upon the selection of seed, that describes how much randomized way has been adopted for the selection of that seed for generation of random pattern. The quality of any random number generator is being tested through a statistical test suite named Diehard Tests. Random numbers or patterns obtained by means of executing a computer program, is based on implementing a particular recursive algorithm are commonly referred to as pseudorandom numbers. “Pseudo”, emphasizes that the selection of seed is in deterministic way. The quality of randomness generated by any random number generator is needed to be verified. For this purpose, the diehard tests are a battery of statistical tests for measuring the quality of a random number generator.

II. RANDOM NUMBER GENERATOR

A random number generator (often abbreviated as RNG) is a device that generates a sequence of numbers without any discernible pattern. A RNG must pass certain statistical tests to be considered valid. There are many applications of randomness, such as gambling, game simulation and cryptography. The wide use for random numbers has led to many different methods for their generation, but they all fall under one of two categories: physical or computational. Both have advantages and disadvantages, depending on whether efficiency or security is the top priority.

A. Relevance to Information Security

A RNG that gives predictable output would make the system vulnerable to replay attacks. Since a system is only as secure as its weakest component, random number generators are a crucial part of information security.

B. Vulnerability to attack

The RNG process is usually a single isolated component easy to locate, making it an attractive target for attackers. If an attacker can predict the sequence of supposedly random numbers, data integrity and confidentiality are compromised. An attack on RNG is difficult to detect by any upstream test of the numbers. Furthermore, such attacks require only a single access to the system. No data need be sent back in contrast to, for example, a computer virus that steals keys and then e-mails them to some drop point. Microsoft uses the CryptGenRandom function to generate random values for its Windows operating system. This function is included in Microsoft's Cryptographic Application Programming

Interface, and is used whenever random numbers are needed. It is said to be cryptographically secure, but the specifics of the algorithm have not been officially published and verified. Researchers have used reverse engineering to test it and have discovered security concerns. An attacker needs only to steal the state bits once, and then they can persistently violate the security of a CryptGenRandom instance and even determine past random numbers generated. This is a serious problem, since the process can be run backwards once the state bits are known, and information already sent are compromised. For example, if the user has made online purchases from websites such as eBay, the random key used to encrypt credit card information can be recovered by an attacker. Moreover, the CryptGenRandom function runs in user mode, so anyone with access to a regular user account on a system can easily obtain access to important information such as the state bits.

C. Methods of randomness

True randomness - Physical methods

The earliest methods for generating random numbers - dice, coin flipping, roulette wheels are still used today, mainly in games and gambling as they tend to be too slow for applications in statistics and cryptography. Some physical phenomena, such as thermal noise in Zener diodes, appear to be truly random and can be used as the basis for hardware random number generators. The generation of true random numbers through measuring physical phenomena is based on the theory of entropy. For example, Hot Bits is an online generator of true random numbers, using timing successive pairs of radioactive decays detected by a Geiger-Müller tube interfaced to a computer. This process is governed by the inherent uncertainty in the quantum mechanical laws of nature. RANDOM.ORG, on the other hand, looks at variations in the amplitude of atmospheric noise. Even lava lamps have been used to generate random numbers -- images of the lamps are converted into a unique video stream.

Pseudo randomness - Computational methods

Computational methods use mathematical formulae or simply precalculated tables to produce sequences of numbers that appear random. Since the computer follows a deterministic algorithm to generate these numbers, they are inherently predictable. For example, Blum Shub (BBS) is a pseudorandom number generator proposed in 1986 by Lenore Blum, Manuel Blum and Michael Shub. It generates sequences of random numbers with the following formula: $x_{n+1} = (x_n)^2 \bmod M$, where M = pq, p and q are both prime numbers. The developers of this method have proved that it is extremely secure. To crack this algorithm would essentially require factorization of large primes, which is assumed to be mathematically infeasible. With a large M, the output of BBS displays no nonrandom patterns that can be discovered with a reasonable amount of calculation.

III. CELLULAR AUTOMATA

A cellular automaton is a discrete model studied in computability theory, mathematics, physics, complexity science, theoretical biology and microstructure modeling. Cellular automata are also called cellular spaces, tessellation automata, homogeneous structures, cellular structures, tessellation structures, and iterative arrays. A cellular automaton consists of a regular grid of cells, each in one of a finite number of states, such as on and off (in contrast to a coupled map lattice). The grid can be in any finite number of dimensions. For each cell, a set of cells called its neighborhood is defined relative to the specified cell. An initial state (time t = 0) is selected by assigning a state for each cell. A new generation is created (advancing t by 1), according to some fixed rule (generally, a mathematical function) that determines the new state of each cell in terms of the current state of the cell and the states of the cells in its neighborhood. Typically, the rule for updating the state of cells is the same for each cell and does not change over time, and is applied to the whole grid simultaneously, though exceptions are known, such as the stochastic cellular automaton and asynchronous cellular automaton.

A. One Dimensional Cellular Automata

The CA structure investigated by Wolfram can be viewed as discrete lattice of sites (cells) where each cell can assume either the value 0 or 1. The next state of a cell is assumed to depend on itself and on its two neighboring cells for a 3-neighborhood dependency. The cells evolve in discrete time steps according to some deterministic rule that depends only on local neighborhood. In effect, each cell as shown in Fig 1, consists of a storage element (D- Flip Flop) and a combinational logic implementing the next state.

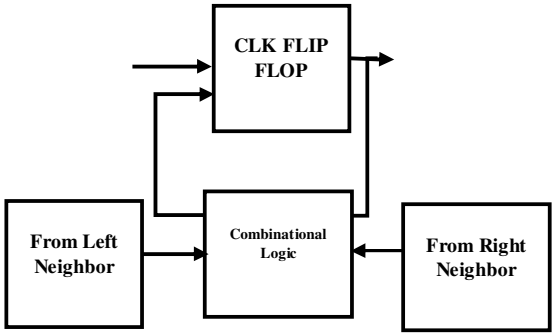

Fig.1. Basic CA Block

If the next state function of a cell is expressed in the form of a truth table, then the decimal equivalent of the output is conventionally called the rule number for the cell. Thus, for a 3 neighborhood CA, the next state function for cell i is represented as follows for each of 256 rules. From table 1 the top rows give all the possible states of the neighborhood cells at the time instant (t), while the 2nd and 3rd rows give the corresponding states of the i^{th} cell at the time instant (t+1) for two illustrative CA rules. The Second Row taken as binary number and converted into decimal representation is the rule no. 90. Similarly, the third row corresponds to rule no. 2. The

expression for a rule can be obtained from its truth table. The minimized expression for rule 90 & rule 2 is

Neighbor State	111	110	101	100	011	010	001	000	RULE
Next State	0	1	0	1	1	0	1	0	90
Next State	1	0	0	1	0	1	1	0	2

TABLE 1. 1D CA EXAMPLE

$$q_i\,(t+1) = q_{i-1}(t) \oplus q_{t+1}(t) \quad \text{- (1)}$$
$$q_i\,(t+1) = q_{i-1}(t) \oplus q_i(t) \oplus q_{t+1}(t) \quad \text{- (2)}$$

Definitions:

1. If same rule is applied to all the cells in a CA, then the CA is said to be Uniform or Regular CA.
2. If different rules are applied to different cells in a CA, then the CA is said to be Hybrid CA.
3. The CA is said to be a Periodic CA if the extreme cells are adjacent to each other.
4. The CA is said to be a Null boundary CA if the extreme cells are connected to logic 0-state.
5. If in CA the neighborhood dependence is on XOR or XNOR only, then the CA is called additive CA, specifically, a linear CA employs XOR rules only.
6. A CA whose transformation is invertible (i.e. all the states in the state transition diagram lie in some cycle) is called a Group CA, otherwise it is called a Non-Group CA.

B. Two Dimensional Cellular Automata

In 2D cellular automata the cells are arranged in two dimensional grids with connections among the neighborhood cells. The state of CA at any time instant can be represented by an m x n binary matrix. The neighborhood function specifying the next state of a particular cell of the CA is affected by the current state of itself and eight cells in its nearest neighborhood. Mathematically, the next state q of the $(i, j)^{th}$ cell of a 2D CA is given by

$$qij\,(t+1) = f\,[qi-1, j-1\ t \oplus qi-1, j\ t, qi-1, j+1\ t, qi, j-1\ t, qi, j+1\ t, qi+1, j-1\ t, qi+1, j\ t, qi+1, j+1\ t] \quad \text{- (3)}$$

Where f is the Boolean function of 9 variables. To express a transition rule of 2D CA, a specific rule convention is used.

0	1	0
1	0	0
1	1	1

Fig.2. 2D CA Block

The central box represents the current cell (that is the cell being considered) and all other boxes represent the eight nearest neighbors of that cell. The number within each box represents the rule number associated with that particular neighbor of the current cell – that is, if the next state of a cell is dependent only on its present state, it is referred to as rule 1. If the next state depends on the present state of itself and its right neighbor, it is referred to as rule 3(=1+2). If the next state depends on the present state of itself and its right neighbor, it is referred to as rule 3(=1+2). If the next state depends on the present state of itself and its right bottom, left, and top neighbors, it is referred to as rule 171 (=1+2+8+32+128) and so on.

IV. RELATED WORK

Several methods have been implemented to generate good quality random numbers. Some novel efforts have been established to generate good quality random numbers. Some important efforts have also been made to generate pseudo-random numbers using CA. It has been already established that the pseudo-random numbers generated using maximum length CA [4] exhibits a high degree of randomness. And Some CA rules have been performed by using the 1D cellular automata. The most common way to generate pseudo-random number is to use a combination of "randomize" and "rand" functions. Random patterns can be achieved based on the following recursive PRNG equation.

$$X_{n+1} = P_1 X_n + P_2 (\text{Mod } N) \quad \text{- (4)}$$

Here P1, P2 are prime numbers, N is range for random numbers. Xn is calculated recursively using the base value X0. X0 is termed as seed and it is a prime number. If X0 (seed) is same all time or selected in any deterministic way, then it yields pseudo-random number.

In existing systems, the random numbers are generated using maximum length cellular automata and 1D cellular automata [1]. When generating random numbers using maximum length CA, the time consumption and performance was lagging due to the cycle decomposition. It makes the system slower because of its recursive manner. Randomness quality is low and minimal number of possibilities for generating random numbers when using 1D cellular automata. And some of the system, there are used the non-uniform transition rules for generating random numbers. The non-uniform transition rules make the poor system efficiency in time and performance, because the transition rule is to be changed for each generation of random numbers.

This proposed research work is intended to

- Study concentrates on generating pseudo-random sequences by using cellular automata (CA)
- Design and develop 2D Cellular Automata based Pseudo Random Number Generator (CA-PRNG)
- Test the generated random numbers by using test suite for the purpose of identifying the quality of them

V. PROPOSED WORK

This generator actually returns 8*8 two-dimensional array of blocks instead of line of cells, each of which is either solid or empty.

A. Configuring CA

First, instead of a line of cells, we now have a two-dimensional matrix of cells. As with the elementary CA, the possible states are 0 or 1. Only in this case, since we're talking about "life," 0 means dead and 1 means alive.

B. Counting Neighbors

The cell's neighborhood has also expanded. If a neighbor is an adjacent cell, a neighborhood is now nine cells instead of three. Before generation, each cell's neighborhood has to be found.

C. . Applying Rules

- **Death.** If a cell is alive (state = 1) it will die (state becomes 0) under the following circumstances.
  Overpopulation: If the cell has four or more alive neighbors, it dies.
  Loneliness: If the cell has one or fewer alive neighbors, it dies.
- **Birth.** If a cell is dead (state = 0) it will come to life (state becomes 1) if it has exactly three alive neighbors (no more, no less).
- **Stasis.** In all other cases, the cell state does not change. To be thorough, let's describe those scenarios.
  Staying Alive: If a cell is alive and has exactly two or three live neighbors, it stays alive.
  Staying Dead: If a cell is dead and has anything other than three live neighbors, it stays dead.

D. . Generating Random Numbers

The random number generator takes the basic CA configuration as the initial seed in the form of 8*8 matrix. After that, count the neighbors for each element of the matrix and transit the elements based on the rules. Thus, the new 2D array will be formed that takes it as the next seed to iterate.

E. . Keeping in File

The generated random numbers are stored in a file on each generation of a number for the purpose of testing the randomness of them.

F. Proposed Algorithm

- Configuring 2D CA with random state: 0(Dead) or 1(Alive) and taking it as the seed.
- Count the neighbors for each cell.
- Generate the next state of CA by using the following rules.
  - If a living cell has less than two living neighbors, it dies.
  - If a living cell has two or three living neighbors, it stays alive.
  - If a living cell has more than three living neighbors, it dies.
  - If a dead cell has exactly three living neighbors, it becomes alive.
- Consider the new state as a seed for the next iteration.

VI. TESTING RANDOMNESS

After the generation phase, the system has to perform the testing on the generated random numbers for ensuring their randomness. The following tests are executed as enlisted below to measure the quality of the randomness of random pattern generator.

Most of the tests return a p-value, which should be uniform on [0,1) if the input file contains truly independent random bits. Those p-values are obtained by p = F(X), where F is the assumed distribution of the sample random variable X – often normal. But that assumed F is just an asymptotic approximation, for which the fit will be worst in the tails. Thus, you should not be surprised with occasional p-values near 0 or 1, such as 0.0012 or 0.9983. When a bit stream really FAILS BIG, you will get ps of 0 or 1 to six or more places. By all means, do not, as a statistician might, think that a $p < 0.025$ or $p > 0.975$ means that the RNG has "failed the test at the 0.05 level". Such ps happen among the hundreds that DIEHARD produces, even with good RNG's. So, keep in mind that "p happens".

The following tests Diehard test are performed on the random numbers for identifying their randomness

Test No.	Name of the Test
1	Birthday Spacings
2	Monte Carlo Test
3	1 Bit Test
4	2 Bit Test
5	Runs Up1 Test
6	Runs Up2 Test
7	Runs Down1 Test
8	Runs Down2 Test

Table 2. Diehard Tests

The comparison result in table 2 shows the degree of randomness achieved by different random number generators in diehard tests. The table shows the pvalue, chi-square value and pi-value that given by diehard tests

Test No.	Existing System	Proposed System	Status
1	p = 0.1700	P = 0.2500	Pass
2	pi = 0.9250	pi = 1.7230	Pass
3	p = 0.3146	p = 0.5549	Pass
4	Chi – square = 0.2465	Chi – square = 0.3169	Pass
5	p = 0.128	p = 0.3951	Pass
6	p = 0.399	p = 0.481	Pass
7	p = 0.412	p = 0.513	Pass
8	p = 0.376	p = 0.8522	Pass

Table 3. Performance result through Diehard for existing and proposed system

VII. CONCLUSION

The random number generator for 2D CA produces the random numbers with the maximum probabilities. The Diehard Test results show the quality of randomness achieved from the samples of random data sets. The number of passes shows the quality of randomness achieved by particular method. Hence this methodology is suitable for generating random sequences with the 2D CA. Also, the probability values are increased in this methodology comparer than previous methodologies. In future, tune the algorithm for getting much better optimal results and have to develop with the optimization techniques for improving the system efficiency such as space, time and cost and have to be implemented the Self Programmable CA as the Controllable CA to achieve the more quality of the random number generator.

REFRENCES

[1] "Cost Optimized Design Technique for Pseudo Random Numbers in Cellular Automata", Arnab Mitra and Anirban Kundu. IJAIT Vol. 2 No.3 June,2012

[2] "Generating High-Quality Random Numbers By Cellular Automata with PSO", Qianfeng, Songanian Yu, Wang Ding and Ming Leng, Fourth International Conference on Natural Computation

[3] "Generating parallel random number generators in Parallel by cellular programming", M. Sipper, M. Tomassini, International Journal of Modern Physics C 7 (2) (1996) 181-190.

[4] "Theory and applications of cellular automata in cryptography," S. Nandi, B. K. Kar, and P. P. Chaudhuri, IEEE Trans. Comput., vol. 43, pp.1346–1357, 1994

[5] "Good random number generators are (not so) easy to find," P. Hellekalek, in Math. Comput. Simulation, 1998, vol. 46, pp. 485–505.

[6] "Simple cellular automata as pseudorandom m-sequence generators for built-in self-test," M. Matsumoto, ACM Trans. Modeling Comput. Simulation, vol. 8, no. 1, pp. 31–42, 1998.

[7] "Application of cellular automata in cryptography", Master Thesis, Warsaw University of Technology, 2002 (in Polish)

[8] "Cryptography with cellular automata, in: Advances in Cryptology: Crypto'85 Proceedings", LNCS 218, Springer, S. Wolfram, 1986, pp. 429–432.

[9] "Symmetric Key Encryption Technique: A Cellular Automata based Approach in WSN", Sathyabrata Roy, Jyothirmoy Karjee, U.S. Rawat, Dayama Pratik N and Nilanjan Dey

[10] http://natureofcode.com/book/chapter-7-cellular-automata/

[11] https://en.wikipedia.org/wiki/Rule_30

[12] https://gamedevelopment.tutsplus.com/tutorials/generate-random-cave-levels-using-cellular-automata--gamedev-9664

[13] http://mathworld.wolfram.com/CellularAutomaton.html