RESEARCH ARTICLE OPEN ACCESS

Percon8 Algorithm for Random Number Generation

Dr. Mrs. Saylee Gharge*, Mr. Honey Brijwani**, Mr. Mohit Pugrani**, Mr. Girish Sukhwani**, Mr. Deepak Udherani**
*(Associate Professor, V. E. S. Institute of Technology, Mumbai)
** (Student, V. E. S. Institute of Technology, Mumbai)

ABSTRACT
In today's technology savvy world, computer security holds a prime importance. Most computer security algorithms require some amount of random data for generating public and private keys, session keys or for other purposes. Random numbers are those numbers that occur in a sequence such that the future value of the sequence cannot be predicted based on present or past values. Random numbers find application in statistical analysis and probability theory. The many applications of randomness have led to the development of random number generating algorithms. These algorithms generate a sequence of random numbers either computationally or physically. In our proposed technique, we have implemented a random number generation algorithm combining two existing random number generation techniques viz. Mid square method and Linear Congruential Generator

Keywords- Linear Congruential Generator, Mid Square Random Number Generation Technique, Permutation Matrix, Maurer's Universal Statistical Test

I. INTRODUCTION

A number of network security algorithms based on cryptography make use of random numbers. For example,

- **Key distribution and reciprocal authentication schemes**: In such schemes, two communicating parties cooperate by exchanging messages to distribute keys and/or authenticate each other. In many cases, nonces are used for handshaking to prevent replay attacks. The use of random numbers for the nonces frustrates an opponent's efforts to determine or guess the nonce.
- **Session key generation**: A secret key for symmetric encryption is generated for use for a short period of time. This key is generally called a session key.
- Generation of keys for the RSA public-key encryption algorithm
- Generation of a bit stream for symmetric stream encryption

These applications give rise to two distinct and not necessarily compatible requirements for a sequence of random numbers: randomness and unpredictability. [1]

1.1. TRNGs, PRNGs, PRFs

Cryptographic applications typically make use of algorithmic techniques for generation of random numbers. These algorithms are deterministic and therefore produce sequences of numbers that are not statistically random. However, if the algorithm is good, the resulting sequences will pass many reasonable tests of randomness. Such numbers are referred to as **pseudorandom numbers (PRNGs)**. The many applications of randomness have led to the development of random number generating algorithms, typically PRNGs.

A popular approach to Random Number Generation is Blum Blum Shub Generator[2] wherein two prime numbers p and q are chosen such that they give a remainder of 3 on division by 4. Then a Random Number is generated such that it is relatively prime to product of p and q. This is processed as long as we require the Random Numbers.

RC4 is a stream cipher designed by Ron Rivest for RSA security. Results show that the period is overwhelmingly more than 10^{100} [3]. RC4 is used in SSL/TLS for communication between web browsers and servers. It is also used in WEP and newer WiFi Protected Access(WPA).

Best treatment of PRNGs can be found in Knuth, D.[4]. An excellent survey of various PRNGS can be found at Ritter, T.[5]

Figure 1 compares a **true random number generator (TRNG)** with two forms of pseudorandom number generators. A TRNG takes as input an entropy source that is effectively random. In essence, the entropy source is drawn from the physical environment of the computer and could include things such as keystroke timing patterns , disk electrical activity and instantaneous values of the system clock.

In contrast, a PRNG takes as input a fixed value, called the seed, and produces a sequence of output bits using a deterministic algorithm. Typically, as shown, there is some feedback path by which some of the results of the algorithm are fed back as input. The important thing to note is that the output bit stream is determined solely by the input value, so that an adversary who is aware of the algorithm and the seed can reproduce the entire bit stream.

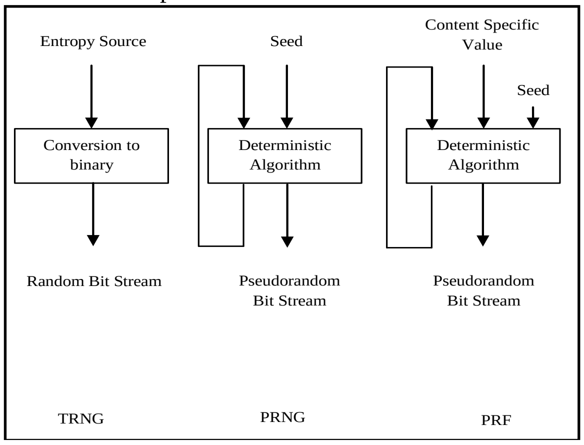


Fig 1. Random and Pseudo Random Number Generators

Fig 1 shows two different forms of PRNGs:

- **Pseudorandom number generator**: An algorithm that is used to produce an open ended sequence of bits is referred to as a PRNG.A common application for an open ended sequence of bits is as input to a symmetric stream cipher. The length of output bits is not fixed.
- **Pseudorandom function (PRF):** A PRF is used to produce a pseudorandom string of bits of some fixed length. Examples are symmetric encryption keys and nonces. Typically, the PRF takes as input a seed plus some context specific values, such as a user ID or an application ID.

Other than the number of bits produced, there is no difference between a PRNG and a PRF.

II. PRNG REQUIREMENTS

2.1. RANDOMNESS

In terms of randomness, the requirement for a PRNG is that the generated bit stream appear random even though it is deterministic. It should exhibit following characteristics:

- **Uniformity**: At any point in the generation of a sequence of random or pseudorandom bits, the probability of occurrence of a zero or one is 1/2.
- **Scalability**: Any test applicable to a sequence can also be applied to sub sequences extracted at random. Hence, any extracted subsequence should pass any test for randomness.
- **Consistency**: The behaviour of a generator must be consistent across starting values (seeds). It is inadequate to test a PRNG based on the output from a single seed or an TRNG on the basis of an output produced from a single physical output.

2.2. UNPREDICTABILITY

A stream of pseudorandom numbers should exhibit two forms of unpredictability:

- **Forward unpredictability**: If the seed is unknown, the next output bit in the sequence should be unpredictable irrespective of any knowledge of previous bits.
- **Backward unpredictability**: It should also not be feasible to determine the seed from knowledge of any generated values. No correlation between a seed and any value generated from that seed should be evident

2.3. SEED REQUIREMENTS

For cryptographic applications, the seed that serves as input to the PRNG must be secure. Because the PRNG is a deterministic algorithm, if the adversary can deduce the seed, then it is highly likely that the output can be determined. Therefore, the seed itself must be a random or pseudorandom number.

Typically, the seed is generated by a TRNG, as shown in Figure 2. One may wonder, if a TRNG is available, why it is necessary to use a PRNG. If the application is a stream cipher, then a TRNG is not practical. The sender would need to generate a keystream of bits as long as the plaintext and then transmit the keystream and the ciphertext securely to the receiver.

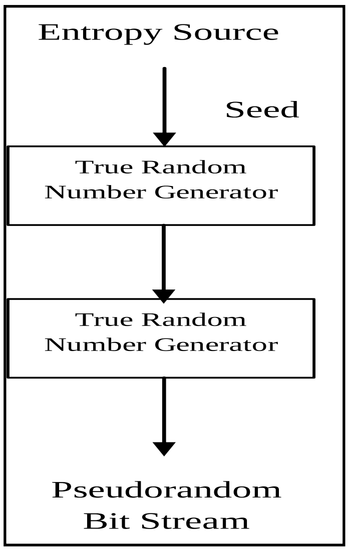


Fig 2. Generation of Seed Input to PRNG

III. MID SQUARE METHOD

The mid square method was proposed by Von-Newmann and Metropolis in 1946. In this method of random number generation, an initial seed is assumed and that number is squared. The middle four digits of the squared value are taken as the first random number.

Next, the random number which is generated most recently is again squared and the middle four digits of this squared value are assumed as the next random number. This is to be repeated to generate the required number of random numbers.

3.1. FLOWCHART

Fig 3 shows the flowchart for generation of Random Numbers using Mid Square Algorithm.
The Loop continues for as long as the Required Random Numbers are generated.

The process is relatively fast as all it takes is computing square of a number and truncation.

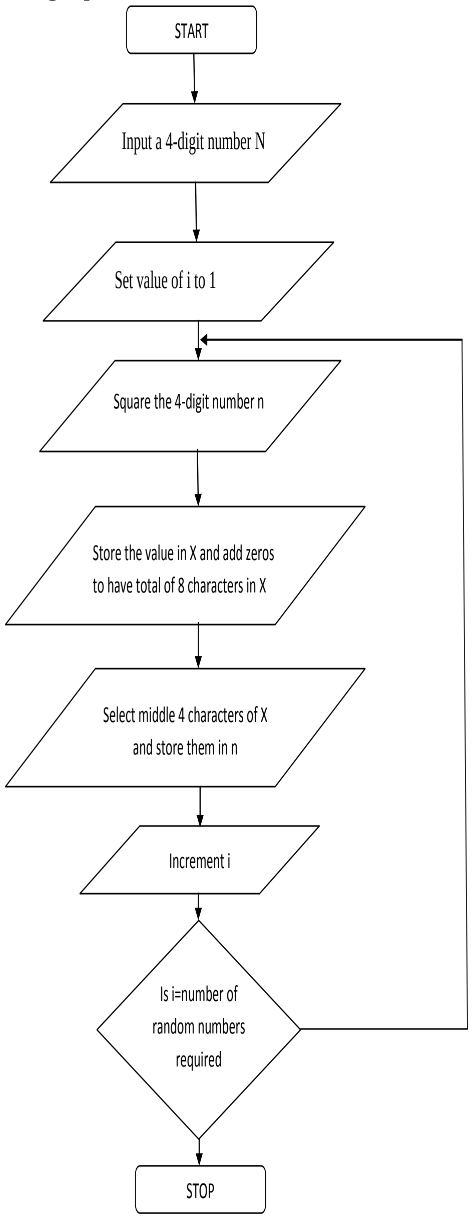

START
Input a 4-digit number N
Set value of i to 1
Square the 4-digit number n
Store the value in X and add zeros to have total of 8 characters in X
Select middle 4 characters of X and store them in n
Increment i
Is i=number of random numbers required
STOP

Fig 3. Flowchart for Mid Square Algorithm

Consider the following example

Table 1. Sample Random Numbers Generated using Mid Square Method

Serial No.	n (4-digit)	n^2
1	8765	76825225
2	8252	68095504
3	0955	00912025
4	9120	83174400
5	1744	03041536
6	0415	00172225
7	1722	02965284
8	9652	93161104

3.2. LIMITATIONS

- Relatively slow
- Statistically unsatisfactory
- Sample of random numbers may be too short
- For a generator of *n*-digit numbers, the period can be no longer than 8^n
- If the middle 4 digits are all zeroes, the generator then outputs zeroes forever. If the first half of a number in the sequence is zeroes, the subsequent numbers will be decreasing to zero

IV. LINEAR CONGRUENTIAL GENERATOR[6]

A widely used technique for pseudorandom number generation is an algorithm first proposed by Lehmer[7], which is known as the linear congruential method.

The algorithm is parameterized with four numbers, as follows:

m	the modulus	$m > 0$
a	the multiplier	$0 < a < m$
c	the increment	$0 \leq c < m$
X_0	the starting value, or seed	$0 \leq X_0 < m$

The sequence of random numbers is obtained via the following iterative equation:

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

If m, a, c, X_0 and are integers, then this technique will produce a sequence of integers with each integer in the range $0 \leq Xn < m$

The selection of values for a, c and m is critical in developing a good random number generator. For example, consider $a = c = 1$. The sequence produced is obviously not satisfactory. Now consider the values $a = 7$ $c = 0$ $m = 32$ and $X_0 = 1$. This generates the sequence {7, 17, 23, 1, 7, etc.} , which is also clearly unsatisfactory. Of the 32 possible values, only four are used; thus, the sequence is said to have a period of 4. If, instead, we change the value of a to 5, then the sequence is {5, 25, 29, 17, 21, 9, 13, 1, 5, etc.} , which increases the period to 8.

We would like m to be very large, so that there is the potential for producing a long series of distinct random numbers. A common criterion is that m be nearly equal to the maximum representable nonnegative integer for a given computer.

Thus, a value of m near to or equal to 2^{31} is typically chosen.

Park, S. and Miller, K proposed three tests to be used in evaluating a random number generator:

T1: The function should be a full-period generating function. That is, the function should generate all the numbers between 0 and m before repeating.

T2: The generated sequence should appear random.

T3: The function should implement efficiently with 32-bit arithmetic.

With appropriate values of a, c, and m, these three tests can be passed. With respect to T1, it can be shown that if m is prime and c= 0 , then for certain values of a the period of the generating function is m-1, with only the value 0 missing.

For 32-bit arithmetic, a convenient prime value of m is $2^{31} - 1$. Thus, the generating function becomes

$X_{n+1} = (aX_n + c) \bmod (2^{31} - 1)$

Of the more than 2 billion possible choices for a, only a handful of multipliers pass all three tests. One such value is $a = 7^5=16807$, which was originally selected for use in the IBM 360 family of computers[8]. This generator is widely used and has been subjected to a more thorough testing than any other PRNG. It is frequently recommended for statistical and simulation work.

4.1. FULL PERIOD

The period of a general LCG is at most m, and for some choices of factor a much less than that. Provided that the offset c is nonzero, the LCG will have a full period for all seed values if and only if:

- c and m are relatively prime,
- (a-1) is divisible by all prime factors of m,
- (a-1) is a multiple of 4 if m is a multiple of 4.

These three requirements are referred to as the Hull-Dobell Theorem. While LCGs are capable of producing pseudorandom numbers which can pass formal tests for randomness, this is extremely sensitive to the choice of the parameters c, m, and a.

4.2. STRENGTHS

- 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
- LCGs are fast and require minimal memory (typically 32 or 64 bits) to retain state. This makes them valuable for simulating multiple independent streams.

4.3. LIMITATIONS

- If an opponent knows that the linear congruential algorithm is being used and if the parameters are known (e.g., $a = 7^5$, $c = 0$, $m = 2^{31} - 1$), then once a single number is discovered, all subsequent numbers are known.
- Even if the opponent knows only that a linear congruential algorithm is being used, knowledge of a small part of the sequence is sufficient to determine the parameters of the algorithm.

Suppose that the opponent is able to determine values for X_0, X_1, X_2, and X_3.

Then;

$X_1 = (aX_0 + c) \bmod m$
$X_2 = (aX_1 + c) \bmod m$
$X_3 = (aX_2 + c) \bmod m$

These equations can be solved for a, c and m

V. PERCON8 ALGORITHM

PERCON8 is named so as its uses Permutation-Concatenation to generate 8 Random Numbers in one Round.PERCON8 Algorithm combines the advantages of Mid Square Random Number Generation technique (4 digit input) and Linear Congruential Generator (m=32) while exploiting their limitations to its use. It is a multi-round algorithm that can be used for generation of Random Numbers. The number of rounds are not fixed, that is, they depend on the values of seed that are used as inputs in every round, until the seed becomes zero. Each round generates a sequence of eight 8- digit Random Numbers. As the technique employs Linear Congruential Generator which produces serially correlated Random Numbers, a Permutation Matrix is used to decorrelate the sequence thereby rendering the estimation nearly impossible.

5.1. SEED GENERATION

This algorithm takes the initial seed generated by a TRNG, that is, an entropy source, and uses the Mid Square Algorithm to generate the seeds required in the following rounds. The seed generated by TRNG is squared and appended with extra zeros so that we have 8 characters. The middle 4 digits serve as the seed for the following round and the process continues until the seed becomes zero after which the algorithm desires another seed form an entropy source. Each round generates a sequence of eight 8-digit Random Numbers.

5.1.1. FLOWCHART

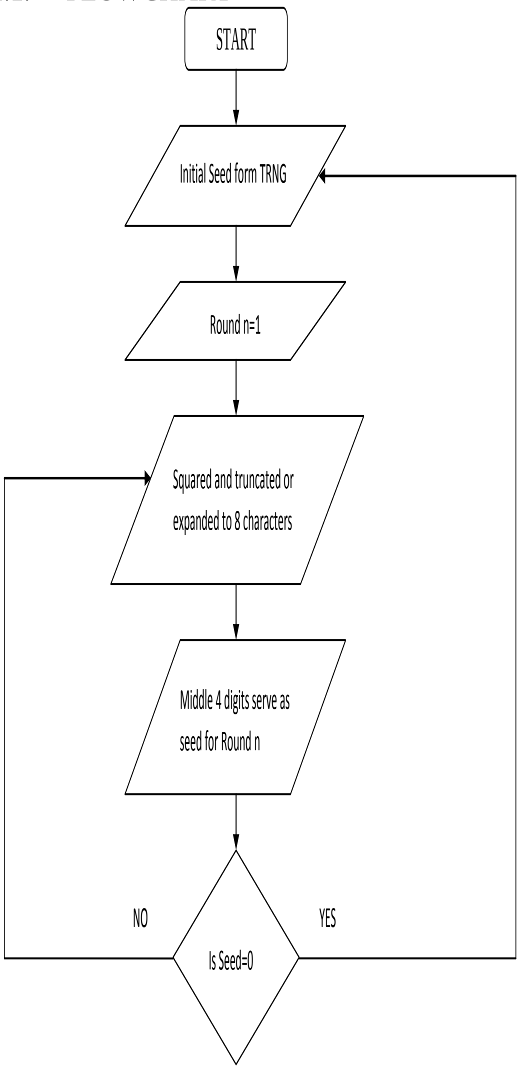

START
Initial Seed form TRNG
Round n=1
Squared and truncated or expanded to 8 characters
Middle 4 digits serve as seed for Round n
Is Seed=0
NO
YES

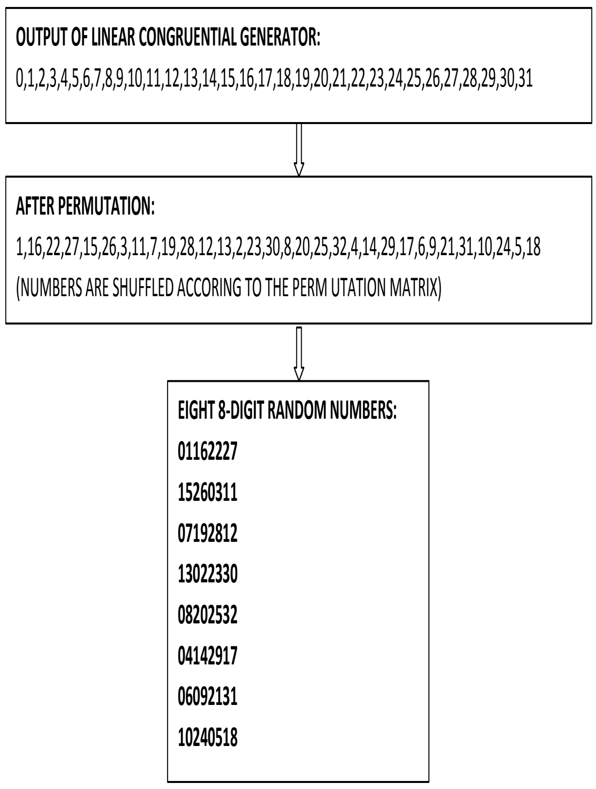
Fig 4. Flowchart for Seed Generation

5.2. ROUND DESCRIPTION

Each round uses Linear Congruential Generator as its function.
Suppose m=32 and values of a and c, as mentioned in section 4, are taken such that maximum period of 32 is achieved.
$X_{n+1} = (aX_n + c)\text{mod } m$

The value X_n is the value of seed generated in round 1, as mentioned in section 5.1. This produces a sequence of 32 (maximum period assumed) serially correlated Random Numbers. To decorrelate the sequence, the generated sequence is permuted using the Permutation Matrix thereby producing 32 decorrelated Random Numbers. These 32 numbers are arranged in set of 4 numbers (2,3,17,19 becomes 02031719) to produce an 8 digit Random Number using modulo 32 arithmetic. Thus, we obtain eight 8-digit Random Numbers.

Consider the following example:



OUTPUT OF LINEAR CONGRUENTIAL GENERATOR:
0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,22,23,24,25,26,27,28,29,30,31

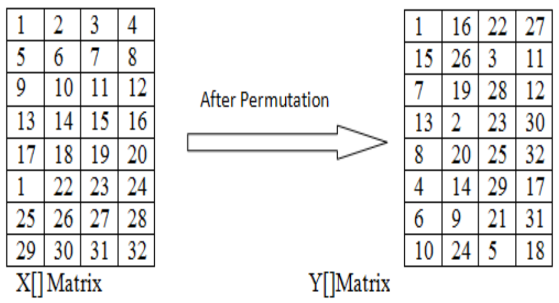
AFTER PERMUTATION:
1,16,22,27,15,26,3,11,7,19,28,12,13,2,23,30,8,20,25,32,4,14,29,17,6,9,21,31,10,24,5,18
(NUMBERS ARE SHUFFLED ACCORING TO THE PERM UTATION MATRIX)

EIGHT 8-DIGIT RANDOM NUMBERS:
01162227
15260311
07192812
13022330
08202532
04142917
06092131
10240518

Fig 5. Example of PERCON8 Algorithm

5.2.1. PERMUTATION MATRIX

The output bits obtained from the Linear Congruential generator are serially correlated as the next number is generated using the previous output as one of the function input. To decorrelate the Output Data, a Permutation Matrix is used, which shuffles the output values in a random order, thereby producing statistically random and independent output. Decorrelation , as a property, is used to eliminate the dependence of output values on each other which makes the randomness vulnerable.
If X[] is the output matrix which stores the output (32 values, assuming maximum period) of the Linear Congruential Generator, then on Permutation the output matrix Y[] has the following value.



1	2	3	4
5	6	7	8
9	10	11	12
13	14	15	16
17	18	19	20
1	22	23	24
25	26	27	28
29	30	31	32

X[]Matrix

After Permutation

1	16	22	27
15	26	3	11
7	19	28	12
13	2	23	30
8	20	25	32
4	14	29	17
6	9	21	31
10	24	5	18

Y[]Matrix

Fig 6. Permutation Matrix

VI. FLOWCHART

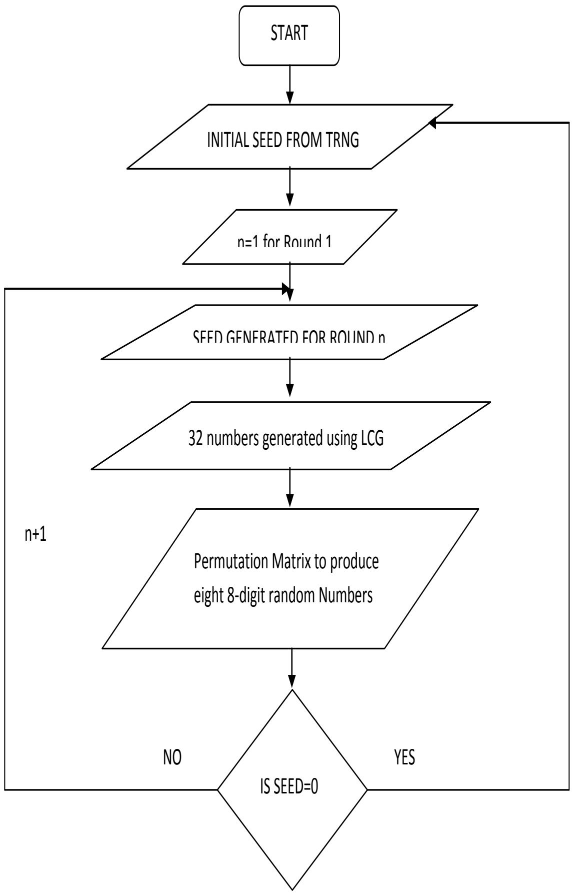
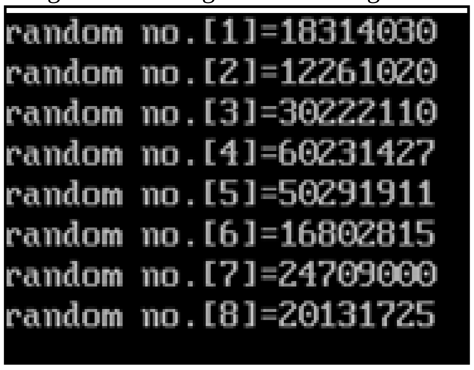

START
INITIAL SEED FROM TRNG
n=1 for Round 1
SEED GENERATED FOR ROUND n
32 numbers generated using LCG
Permutation Matrix to produce eight 8-digit random Numbers
n+1
NO
YES
IS SEED=0

Fig. 7. Flowchart for PERCON8 Algorithm

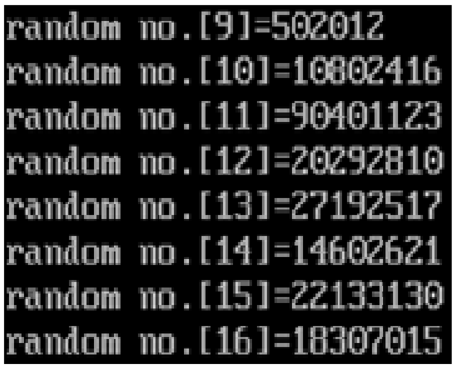
VII. RESULTS

Turbo C implementation of the Program Code with m=32, a=5 and c=1 and initial seed=5811 is shown below. Fig 7,8,9,10 show 32 8-digit random numbers generated using PERCON8 algorithm



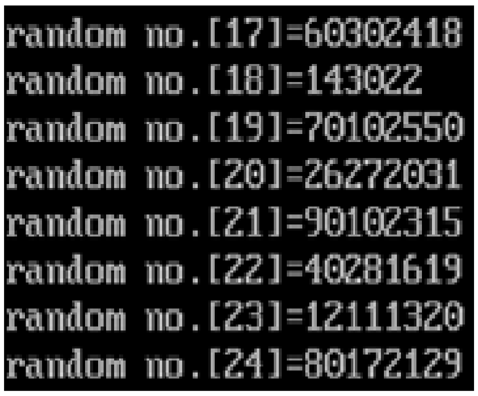
```
random no.[1]=18314030
random no.[2]=12261020
random no.[3]=30222110
random no.[4]=60231427
random no.[5]=50291911
random no.[6]=16802815
random no.[7]=24709000
random no.[8]=20131725
```

Fig 8. Random Numbers 1 through 8[9]



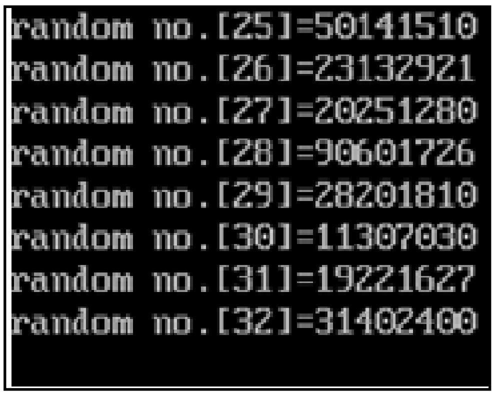
```
random no.[9]=502012
random no.[10]=10802416
random no.[11]=90401123
random no.[12]=20292810
random no.[13]=27192517
random no.[14]=14602621
random no.[15]=22133130
random no.[16]=18307015
```

Fig 9. Random Numbers 9 through 16[9]



```
random no.[17]=60302418
random no.[18]=143022
random no.[19]=70102550
random no.[20]=26272031
random no.[21]=90102315
random no.[22]=40281619
random no.[23]=12111320
random no.[24]=80172129
```

Fig 10. Random Numbers 17 through 24[9]



```
random no.[25]=50141510
random no.[26]=23132921
random no.[27]=20251280
random no.[28]=90601726
random no.[29]=28201810
random no.[30]=11307030
random no.[31]=19221627
random no.[32]=31402400
```

Fig 11. Random Numbers 25 through 32[9]

Following two tests determine the level of randomness of random numbers:

- **Frequency test:** To determine whether the number of 1s and 0s in binary representation of output is nearly same. The frequency test on our results yield a value of 0.47 which is approximately equal to 0.5 thereby ensuring that the number of 1s and 0s are nearly equal.
- **Maurer's Universal Statistical Test:** To detect whether or not the sequence can be significantly compressed. The Maurer's test on our results yield that approximately 42% of bits of the preceding value are repeated thereby making compression possible.[10]

VIII. CONCLUSION

PERCON8 Algorithm clears the Randomness Tests viz. Frequency Test and Maurer's Universal Statistical Test with expected results. Also, generation of 8-digit Random numbers using 32 modulo arithmetic can be achieved using PERCON8 algorithm. The criteria of unpredictability, that is, inability to predict the seed from known output or prediction of output from previous results, is clearly met as the seed is generated afresh at the beginning of every round and the output is decorrelated using Permutation Matrix. This algorithm proves to be faster than its competitors as it uses modulo 32 arithmetic. Thus, PERCON8 Algorithm proves to be efficient as far as Random Number Generation is considered. It can find applications in cryptographic algorithms requiring Random Numbers viz. RSA algorithm.

REFERENCES

[1] William Stallings, "Cryptography and Network Security", 5th edition, Pearson Education India

[2] Blum, L,; Blum, M.; and Shub, M. "A Simple Unpredictable Random Number Generator", SIAM Journal on Computing, No. 2, 1986

[3] Robshaw, M. *Stream Ciphers*. RSA Laboratories Technical Report TR-701, July 1995

[4] Knuth, D. *The Art of Computer Programming, Volume 2: Seminumerical Algorithms*. Reading, MA: Addison-Wesley, 1998

[5] Ritter, T. "The Efficient Generation of Cryptographic Confusion Sequences" *Cryptologia*, vol. 15 no. 2,1991. www.ciphersbyritter.com/ARTS/CRNG2ART.HTM

[6] Jerry Banks, John S. Carson II, Barry L. Nelson, "Discrete-Event System Simulation", Pearson Education India.

[7] Lehmer, D. "Mathematical Methods in Large-Scale Computing" *Proceedings, 2nd Symposium on Large-Scale Digital Calculating Machinery*, Cambridge: Harvard University Press, 1951

[8] Lewis, P.; Goodman,A.; and Miller,J. "A Pseudo- Random Number Generator for the System/360." *IBM Systems Journal No. 2*, 1968

[9] Turbo C IDE to implement Random Number Generation Using PERCON8 Algorithm

[10] Rukhin,A., et al. A Statistical Test Suite for Random and PseudoRandom Number Generators for Cryptographic Applications, NIST Sp 800-22,August 2008