New Design of Crypto-Based Pseudo random number generator (CBPRNG) using BLOW FISH cipher

T.Chalama Reddy 1, Dr.R.Seshadri2

1. Asso proffessor, Department of CSE, Narayana College of Engineering, Nellore, INDIA.
2Professor & Director, Computer Center, Sri Venkateswara University, Tirupathi, INDIA
[1]chalamareddy.t@gmail.com, [2] ravalaseshadri@gmail.com

Abstract:-

Random Number Generators (RNGs) are an important building block for algorithms and protocols in cryptography. Random number generation is used in a wide variety of cryptographic operations, such as key generation and challenge/response protocols. A random number generator outputs a sequence of 0s and 1s such that at any position, the next bit cannot be expected on the previous bits. However, true random number produces non- deterministic output since if the same random generator is run twice, identical results are not received. Thus we go for pseudo random number generator that is deterministic device because if this random number generator is run twice or more, it gives same results. Our paper presents new crypto based pseudo random number generator. It uses BLOW FISH ciphers and the Cipher-Block chaining (CBC) mode that uses three stages of the block chaining. The plain text for each stage comes from the output of the first BLOW FISH, which uses the 64-Bit date and time as the plain text. CBPRNG creates three 64-bit random numbers, the first and the second are concatenated to create a 128-random number and the third is used as next initial vector (IV) for CBPRNG. Our Crypto-Based Pseudo-random Number Generator produces a sequence of bits that has a random looking distribution. This new generator helps to develop huge range cryptographic applications to increase the system security. A number of applications such as financial security applications and Pretty Good Privacy (PGP) use this technique.

Keywords:- PRNG, TRNG, Cryptographic Random numbers, seed, BLOW FISH, 3DES ciphers

1. INTRODUCTION

Random Number Generators (RNGs) used for cryptographic applications typically produce a sequence of zero and one bits that may be combined into sub-sequences or blocks of random numbers. There are classified into: *deterministic* and *nondeterministic*. A deterministic RNG consists of an algorithm that produces a sequence of bits from an initial value called a seed. A nondeterministic RNG generates output that is dependent on some irregular physical source that is outside human control.

Good cryptography requires good random numbers. The following criteria are used to validate that a sequence of numbers is random.

Uniform distribution: The uniform of numbers in the sequence should be uniform; that is, the frequency of occurrence of each of the numbers should be approximately the same.

Independence: No one value in the sequence can be inferred from the others.

This paper evaluates the new crypto based Pseudo Random Number Generator (CBPRNG) for use in cryptographic applications. Almost all cryptographic protocols require the generation and use of secret values that must be unpredictable to attackers. For example, random number generators are required to generate public/private key pairs for asymmetric (public key) algorithms such as RSA, DSA, and Diffie-Hellman. The Keys for symmetric and hybrid cryptosystems are also generated randomly. The most important is that attackers, who know the RNG design, must not be able to create any useful predictions about the RNG outputs.

In practice, there are two major types of random number generators (RNGs): true-random (TRNGs) and pseudo-random (PRNGs) both of which have been employed in various commercial applications.

True Random Number Generators (TRNGs), on the other hand, capitalize on naturally occurring random phenomena and generate nearly perfect statistical randomness without the need for seed initialization. For this reason, system designers should strongly consider the use of TRNGs for any current or future application that depends on randomness. However, there are several draw backs to this approach. The process is normally slow, and the same random stream cannot be repeated if needed,

Pseudo-random number generators are designed using algorithms that generate numbers or bit streams that appear to be random. In most cases the output from these RNGs are random enough to pass basic statistical testing, but given that this method employs a deterministic approach that is initialized with a seed, it is debatable whether this category can be considered genuinely random. However, one of the most important uses of random numbers comes from cryptographic computer security protocols and algorithms.

Cryptographic applications use random numbers to generate encryption keys, create initial parameter values, and to introduce random nonces into protocols and padding schemes. In most cases these numbers come from a Pseudorandom Number Generator (PRNG) which is a deterministic software algorithm which *imitates* randomness. A PRNG takes an input string of bits, or a bit vector, and generates a longer output bit-vector which "appears" random.

One-way function can be used directly as pseudorandom number generator that is cryptographically sound. Therefore, algorithms such as SHA-1 [5] or RSA are commonly used as a PRNG. Unfortunately, PRNGs suffer from two major security disadvantages. First, PRNGs require some input which deterministically governs the output. To securely use a PRNG, this input (the "seed") must be kept *secret*. Second, PRNGs can only generate fixed number bits before they cycle and repeat themselves.

Present paper is as follows. Section 2, describes the related researches. In section 3, we present our Proposed Cryptosystem-based random number generator. In section 4, describe Implementation and Settings. In section 5 present Implementation Results and analysis and in section 6, we present application *of* random numbers. Finally, conclusion is summed up in section 7.

2. Related researches

Random numbers plays an important role in the use of encryption for various network security applications. Sources of use true random numbers are hard to come by. The physical noise generators, such as pulse detectors of ionizing radiation events, gas discharge tubes, and leaky capacitors, are one potential source. However, such devices are of limited utility in network security applications.

The most widely used technique for pseudorandom number generation is an algorithm first proposed by lehmer, 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	$0<=c<m$
X_0	the starting value or seed	$0<=X_0<m$

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

$$Xn+1=(aXn+c) \bmod m$$

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

A cryptosystem such as an encryption cipher or a hash function can also be use to generate a random stream of bits. We briefly mention a system that used 3DES encryption algorithm.

The PRNG proposed in ANSI X9.17 defines a cryptographically strong pseudorandom number generator. A number of applications employ this technique, including financial security applications and PGP (Pretty Good Privacy). This generator uses three 3DES with two keys (encryption-decryption-encryption). The PRNG proposed in ANSI X9.17 have the property that one the key has been compromised, an attacker is forever after able to predicate their outputs. But our

CBPRNG can recover from a key compromise.

It was concluded in [6] that BLOW FISH is faster and more efficient than other common encryption algorithms. So, in our proposed CBPRNG we use BLOW FISH cipher for instead of other ciphers.

Blowfish was designed in 1993 by Bruce Scheier as a fast, alternative to general existing encryption algorithms such AES, DES and 3DES etc. Since Blowfish has not any known security weak points so far, this makes it an excellent candidate to be considered as a standard encryption algorithm. Blowfish is unpatented, and will remain so in all countries. The algorithm is hereby placed in the public domain, and can be freely used by anyone.

Blowfish is a symmetric block encryption algorithm designed in consideration with,

Fast: It encrypts data on large 32-bit microprocessors at a rate of 26 clock cycles per byte.

Compact: It can run in less than 5K of memory.

Simple: It uses addition, XOR, lookup table with 32-bit operands.

Secure: The key length is variable, it can be in the range of 32~448 bits: default 128 bits key length.

3. Proposed Cryptosystem-Based Pseudo Random Number Generator (CBPRNG)

A cryptosystem such as an encryption cipher or a hash function can also be use to generate a random stream of bits. Our CBPRNG generator uses encryption algorithm. This generator makes use of four BLOW FISH encryptions modules. All four make use of same pair of variable 32- 448-bit Keys, which must kept secrete, and 64 bit seed value as the initial vector (IV).

The CBC requires more processing time than ECB because of its key-chaining nature. It was concluded in [6] that the extra time added is not significant for many applications, knowing that CBC is much better than ECB in terms of protection. The difference between the two modes is hard to see by the naked eye, the results showed that the average difference between ECB and CBC is 0.059896 second, which is relatively small. ECB uses simplest way to break plaintext into 64-bit blocks and encrypt each of them with same key. This can be attacked by replace one encrypted block with another encrypted block if structure of document is known. So, in this paper we propose the Cipher-Block chaining (CBC) mode that uses three stages of the block chaining as shown in fig2. The plain text for each stage comes from the output of the first BLOW FISH (1), which uses the 64-Bit date and time as the plain text. CBPRNG creates three 64-bit random numbers, the first and the second are concatenated to create a 128-random number and the third is used as next initial vector (IV).

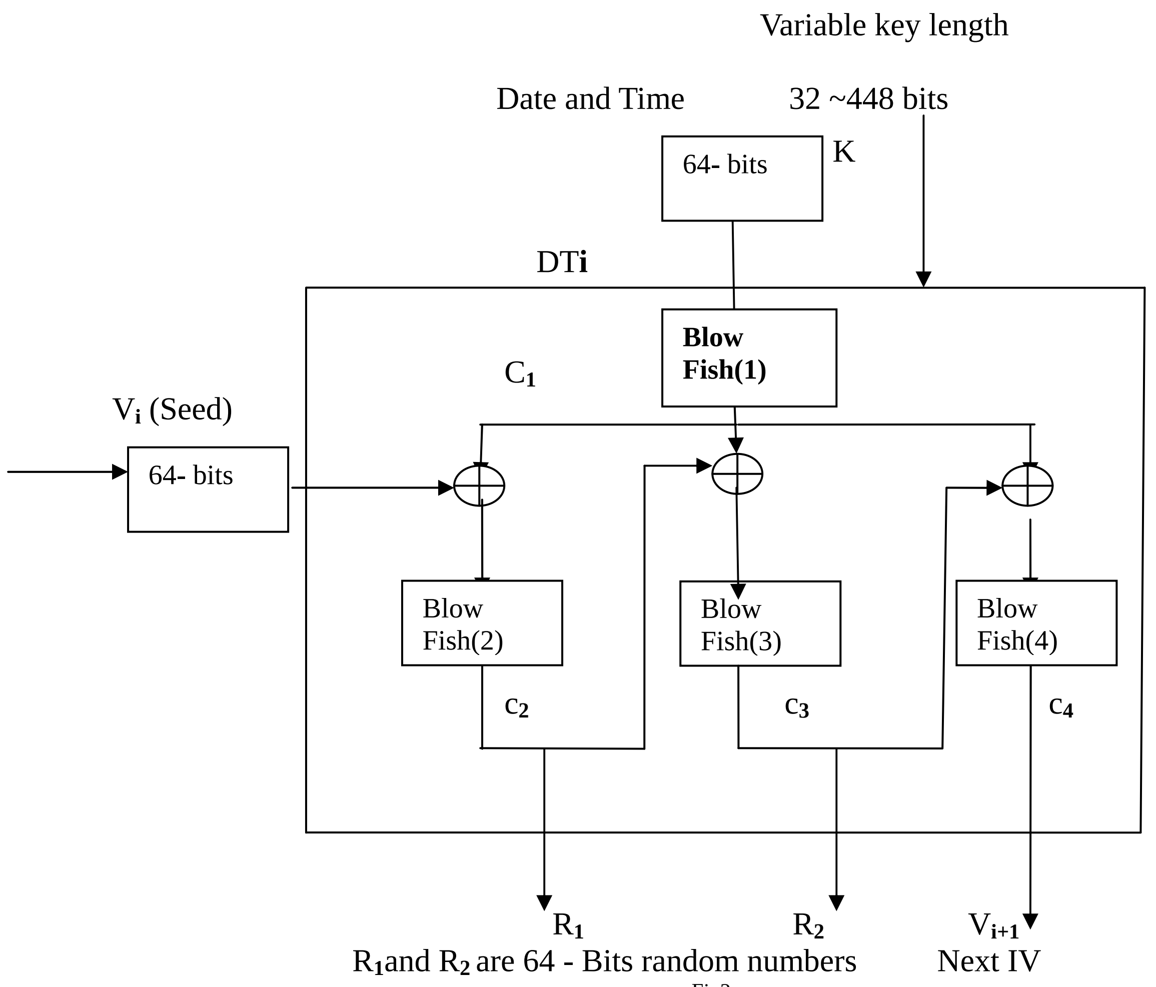


Fig2:

The fig2 shows the steps in New Crypto-based pseudorandom number generator. It uses three Blow Fish ciphers.

Input to DTi is Date and time string which is obtained in the program itself by getting the current time and date which varies according to date and time. The other a 64-bit seed value; this is initialized to some arbitrary value and is updated during the generation process. From f igure2, we define the following quantities:

-- C1,C2,C3, and C4 are outputs (cipher texts) of BLOW FISH(1), BLOW FISH(2), BLOW FISH(3), and BLOW FISH(4) respectively.

-- DT_i denotes 64- bit date/time value at the beginning of i^{th} stage,

-- V_i is 64-bit seed value at the beginning of i^{th} generation stage,

---Values of R_1 and R_2 are C_2 and C_3 respectively for each generation stage.

-- Ri is the concatenation of R_1 and R_2 that results 128- bit pseudorandom number generated by the i^{th} generation stage and,

--Same key K upto 448 bits is used for each stage.

Then encryption process is as follows:

$\mathbf{c_1}$= BlowFish$_{\mathbf{K}}$ [DTi]

$\mathbf{c_2}$=BlowFish $_{\mathbf{K}}$ [Vi XOR $\mathbf{c_1}$]

$\mathbf{c_3}$=BlowFish$_{\mathbf{k}}$**[**c1 XOR $\mathbf{c_2}$**]**

R1=c2, R2=c3

Ri (128-bit Pseudo Random number)=$R_1 \| R_2$

$\mathbf{c_4}$= BlowFish$_{\mathbf{k}}$[c1 XOR $\mathbf{c_3}$]

$V_{i+1} = \mathbf{c_4}$ **is** next IV for (i+1) $^{\mathbf{th}}$ generation stage, i.e. Vi=Vi+1.

Several factors contribute to the cryptographic strength of this method. The technique involves a variable bit length key and four BLOW FISH encryptions. The scheme is driven by two pseudorandom inputs, the date and time value, and a seed produced by the any other generator. Even if a pseudorandom number R_1 or/ and R_2 is compromised, it will be impossible to deduce the next IV (Vi+1) from the R_1 or/and R_2 because an additional BLOW FISH operation is used to produce the next IV.

4. Implementation and Settings:

The implementation of CBPRNG provides 64 Bit Random Numbers output. The Random number output is derived from the byte values of C_2, C_3, and C_4 (Cipher Text). The inputs supplied to the system are a private key whose length can be up to 448 Bits (56 Bytes), Initial Vector (seed) whose length is 64 Bit (8 Byte) and Dti (Date Time Input) whose length is 64 Bit(8 Byte).

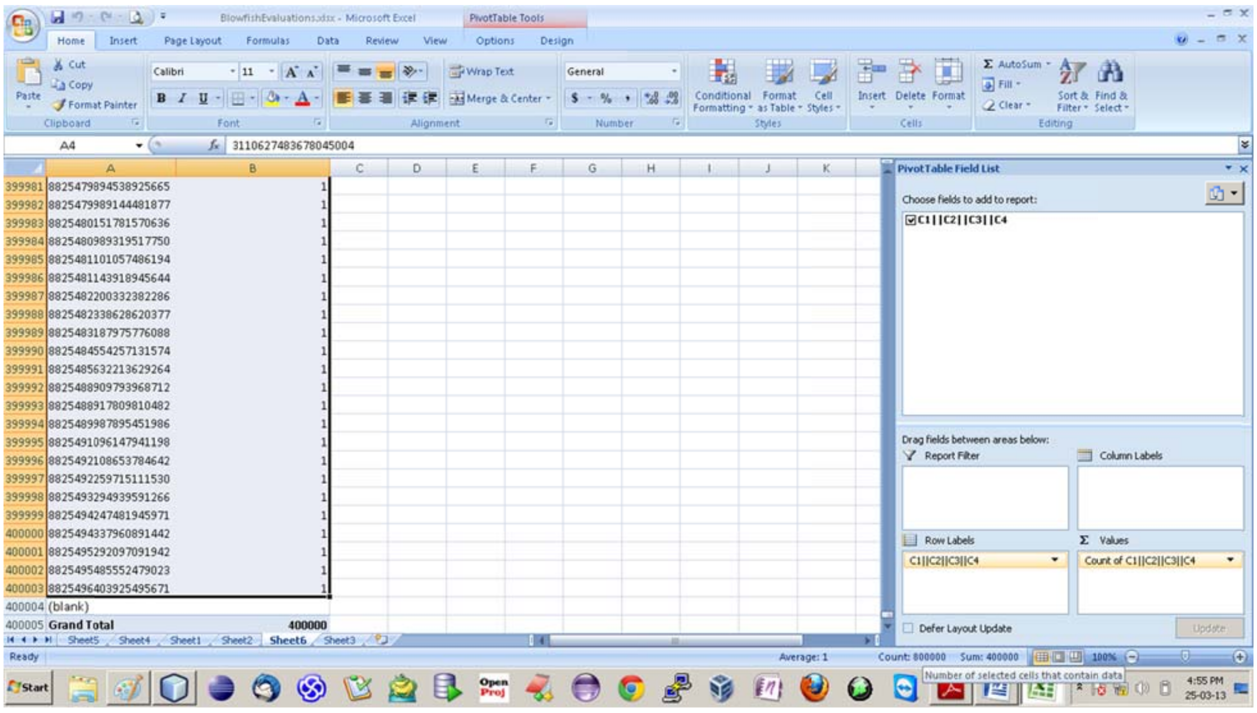
The GUI used to evaluate the proposed CBPRNG using Blowfish algorithm, was developed using Java SDK 1.6.26 version along with NetBeans 7.1 Vesion IDE.

We have used getTime() method from java.util.Date class to set the Dti value as a long positive integer. We have used javax.crypto package, javax.security package, Open Source Bouncy Castle API and javax.swing packages.

The system used to develop the CBPRNG implementation has the following specifications. Intel(R) Core(TM) i3 CPU M370 @ 2.40GHz 2.40GHz with 4.00 GB RAM, Windows 7 Home Premium 32-Bit OS.

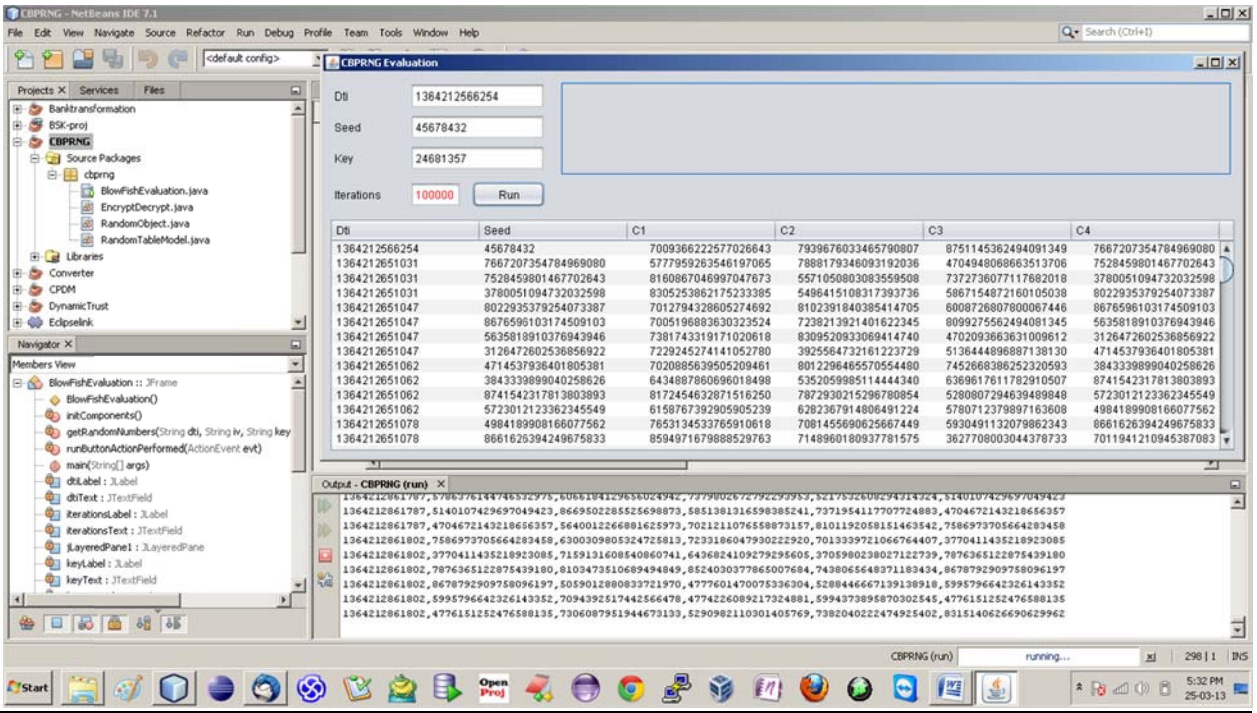
5. Implementation Results and analysis:

A Set of Iterations were conducted using the CBRNG Evaluation Tool to Identify the Repetitions of Random Number output in the Byte value Converted Cipher Texts C1,C2,C3,C4. We have used Microsoft Excel 2007 Pivot Table to find the Repetitions of Random values that are generated in C1,C2,C3,C4. We repeated them for many set of Iterations and cross checked in Pivot Table. The Output Screen of Pivot Table indicates no repetitions of C1,C2,C3,C4 in a set of 4 Lakh Random Number Set.



A4 | 3110627483678045004

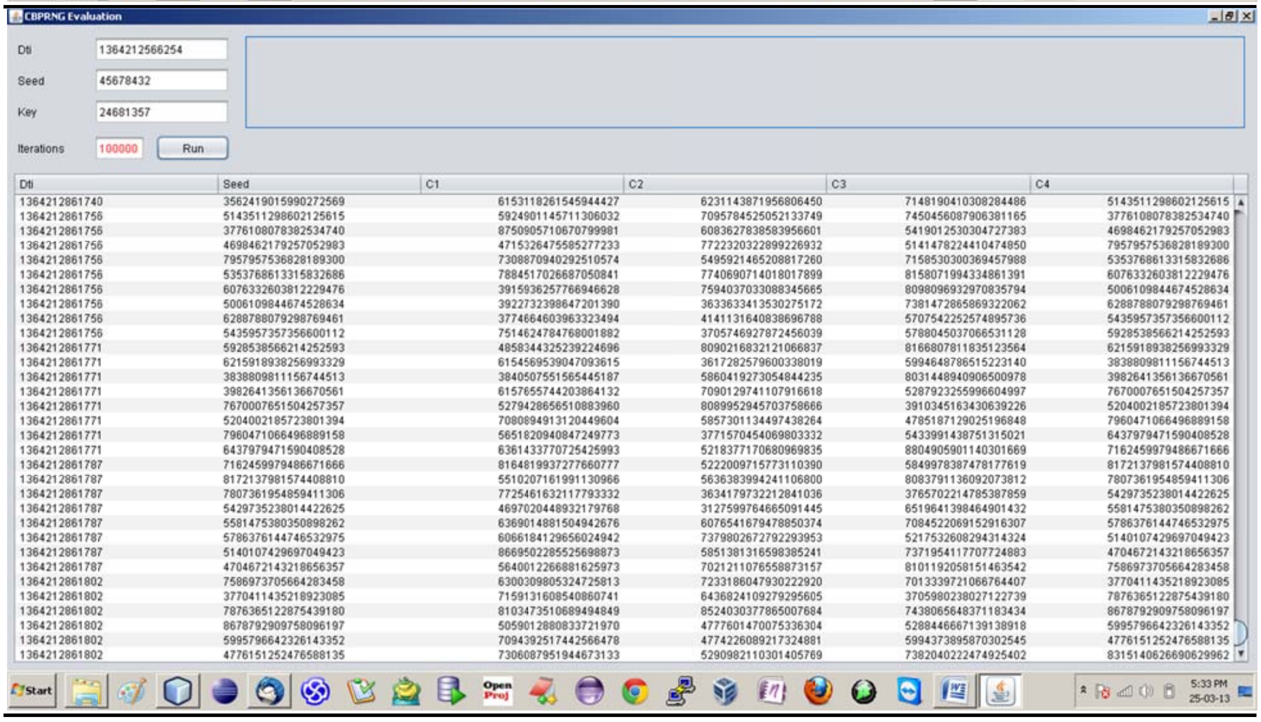
	A	B
399981	8825479894538925665	1
399982	8825479989144481877	1
399983	8825480151781570636	1
399984	8825480989319517750	1
399985	8825481101057486194	1
399986	8825481143918945644	1
399987	8825482200332382286	1
399988	8825482338628620377	1
399989	8825483187975776088	1
399990	8825484554257131574	1
399991	8825485632213629264	1
399992	8825488909793968712	1
399993	8825488917809810482	1
399994	8825489987895451986	1
399995	8825491096147941198	1
399996	8825492108653784642	1
399997	8825492259715111530	1
399998	8825493294939591266	1
399999	8825494247481945971	1
400000	8825494337960891442	1
400001	8825495292097091942	1
400002	8825495485552479023	1
400003	8825496403925495671	1
400004	(blank)	
400005	**Grand Total**	**400000**



Dti 1364212566254
Seed 45678432
Key 24681357
Iterations 100000

Dti	Seed	C1	C2	C3	C4
1364212566254	45678432	7009366222577026643	7939676033465790807	8751145362494091349	7667207354784969080
1364212651031	7667207354784969080	5777959263546197065	7888179346093192036	4704948068663513706	7528459801467702643
1364212651031	7528459801467702643	8160867046997047673	5571050803083559508	7372736077117682018	3780051094732032598
1364212651031	3780051094732032598	8305253862175233385	5496415108317393736	5867154872160105038	8022935379254073387
1364212651047	8022935379254073387	7012794328605274692	8102391840385414705	6008726807800067446	8676596103174509103
1364212651047	8676596103174509103	7005196883630323524	7238213921401622345	8099275562494081345	5635818910376943946
1364212651047	5635818910376943946	7381743319171020618	8309520933069414740	4702093663631009612	3126472602536856922
1364212651047	3126472602536856922	7229245274141052780	3925564732161223729	5136444896887138130	4714537936401805381
1364212651062	4714537936401805381	7020885639505209461	8012296465570554480	7452668386252320593	3843339899040258626
1364212651062	3843339899040258626	6434887860696018498	5352059985114444340	6369617611782910507	8741542317813803893
1364212651062	8741542317813803893	8172454632871516250	7872930215296780854	5280807294639489848	5723012123362345549
1364212651062	5723012123362345549	6158767392905905239	6282367914806491224	5780712379897163608	4984189908166077562
1364212651078	4984189908166077562	7653134533765910618	7081455690625667449	5930491132079862343	8661626394249675833
1364212651078	8661626394249675833	8594971679888529763	7148960180937781575	3627708003044378733	7011941210945387083

```
1364212861787,5786376144746532975,6066184129656024942,7379802672792293953,5217532608294314324,5140107429697049423
1364212861787,5140107429697049423,8669502285525698873,5851381316598385241,7371954117707724883,4704672143218656357
1364212861787,4704672143218656357,5640012266881625973,7021211076558873157,8101192058151463542,7586973705664283458
1364212861802,7586973705664283458,6300309805324725813,7233186047930222920,7013339721066764407,3770411435218923085
1364212861802,3770411435218923085,7159131608540860741,6436824109279295605,3705980238027122739,7876365122875439180
1364212861802,7876365122875439180,8103473510689494849,8524030377865007684,7438065648371183434,8678792909758096197
1364212861802,8678792909758096197,5059012880833721970,4777601470075336304,5288446667139138918,5995796642326143352
1364212861802,5995796642326143352,7094392517442566478,4774226089217324881,5994373895870302545,4776151252476588135
1364212861802,4776151252476588135,7306087951944673133,5290982110301405769,7382040222474925402,8315140626690629962
```



Dti 1364212566254
Seed 45678432
Key 24681357
Iterations 100000

Dti	Seed	C1	C2	C3	C4
1364212861740	3562419015990272569	6153118261545944427	6231143871956806450	7148190410308284486	5143511298602125615
1364212861756	5143511298602125615	5924901145711306032	7095784525052133749	7450456087906381165	3776108078382534740
1364212861756	3776108078382534740	8750905710670799981	6083627838583956601	5419012530304727383	4698462179257052983
1364212861756	4698462179257052983	4715326475585277233	7722320322899226932	5141478224410474850	7957957536828189300
1364212861756	7957957536828189300	7308870940292510574	5495921465208817260	7158530300369457988	5353768613315832686
1364212861756	5353768613315832686	7884517026687050841	7740690714018017899	8158071994334861391	6076332603812229476
1364212861756	6076332603812229476	3915936257766946628	7594037033088345665	8098096932970835794	5006109844674528634
1364212861756	5006109844674528634	3922732398647201390	3633633413530275172	7381472865869322062	6288788079298769461
1364212861756	6288788079298769461	3774664603963323494	4141131640838696788	5707542252574895736	5435957357356600112
1364212861756	5435957357356600112	7514624784768001882	3705746927872456039	5788045037066531128	5928538566214252593
1364212861771	5928538566214252593	4858344325239224696	8090216832121066837	8166807811835123564	6215918938256993329
1364212861771	6215918938256993329	6154569539047093615	3617282579600338019	5994648786515223140	3838809811156744513
1364212861771	3838809811156744513	3840507551565445187	5860419273054844235	8031448940906500978	3982641356136670561
1364212861771	3982641356136670561	6157655744203864132	7090129741107916618	5287923255996604997	7670007651504257357
1364212861771	7670007651504257357	5279428656510883960	8089952945703758666	3910345163430639226	5204002185723801394
1364212861771	5204002185723801394	7080894913120449604	5857301134497438264	4785187129025196848	7960471066496889158
1364212861771	7960471066496889158	5651820940847249773	3771570454069803332	5433991438751315021	6437979471590408528
1364212861771	6437979471590408528	6361433770725425993	5218377170680969835	8804905901140301669	7162459979486671666
1364212861787	7162459979486671666	8164819937277660777	5222009715773110390	5849978387478177619	8172137981574408810
1364212861787	8172137981574408810	5510207161991130966	5636383994241106800	8083791136092073812	7807361954859411306
1364212861787	7807361954859411306	7725461632117793332	3634179732212841036	3765702214785387859	5429735238014422625
1364212861787	5429735238014422625	4697020448932179768	3127599764665091445	6519641398464901432	5581475380350898262
1364212861787	5581475380350898262	6369014881504942676	6076541679478850374	7084522069152916307	5786376144746532975
1364212861787	5786376144746532975	6066184129656024942	7379802672792293953	5217532608294314324	5140107429697049423
1364212861787	5140107429697049423	8669502285525698873	5851381316598385241	7371954117707724883	4704672143218656357
1364212861787	4704672143218656357	5640012266881625973	7021211076558873157	8101192058151463542	7586973705664283458
1364212861802	7586973705664283458	6300309805324725813	7233186047930222920	7013339721066764407	3770411435218923085
1364212861802	3770411435218923085	7159131608540860741	6436824109279295605	3705980238027122739	7876365122875439180
1364212861802	7876365122875439180	8103473510689494849	8524030377865007684	7438065648371183434	8678792909758096197
1364212861802	8678792909758096197	5059012880833721970	4777601470075336304	5288446667139138918	5995796642326143352
1364212861802	5995796642326143352	7094392517442566478	4774226089217324881	5994373895870302545	4776151252476588135
1364212861802	4776151252476588135	7306087951944673133	5290982110301405769	7382040222474925402	8315140626690629962

6. Applications of Random Numbers

Random numbers are playing vital role in all aspects of cryptography. Cryptographic algorithms and protocols, such as RC4, DSA, SET and SSL require random numbers. Digital documents such as email or sending a small amount of Cash over internet, we are in need of random numbers. Particularly, random numbers are used in the following applications.

session keys and message keys are used in symmetric algorithms such as blow fish or triple DES. The session key is a one-time random key and used only once, or duration of session. For example, Email security system uses 128-bit session key for encrypting the email message.

one- time- pad is a one-time random key and used only once. Each message requires a new key of the same length as the new message. It produces random output that bears no statistical relationship to the plaintext. The Vernam cipher uses a one- time pad, which is discarded after a single use, and therefore is suitable only for short messages. For example, if the president of a country needs to send a completely secrete message to the president of another country, she can send a trusted envoy with random key before sending the message. The one time pad offers complete security.

Cipher block chaining modes requires initialization vectors that are random numbers.

Authentication protocols and Kerberos uses random challenges that are random numbers.

The seeds for routines that produce mathematical values, such as huge prime numbers for RSA or EIGmal crypto systems.

7.CONCLUSION:

Good cryptography has need of good random numbers. This paper evaluate the new crypto based Pseudorandom Number Generator (CBPRNG) for use in cryptographic applications. Almost all cryptographic protocols require the generation and use of secret values that must be unknown to attackers. The ANSI X9.17 defines a cryptographically strong pseudorandom number generator. This generator uses three 3DES with two keys (encryption-decryption-encryption). In ANSI X9.17 one the key has been compromised, an attacker is forever after able to predicate their outputs. To avoid from a key compromise, in our proposed design of CBPRNG we used BLOW FISH ciphers for instead of other ciphers because BLOW FISH is faster and more efficient than other encryption algorithms. By using Micro- soft Excel, we tested a large set of Random numbers generated by Our **CBPRNG**. Finally we concluded that there are no repetitions and no statistical relation ship between numbers. Pseudorandom number generators can imitate randomness sufficiently well for most applications; however, they still rely on some secret seed input.

REFERENCES

[1] R. B. P. Dept. The Evaluation of Randomness of RPG100 by Using NIST and DIEHARD Tests.Technical report, FDK Corporation, 2003.

[2] B. Jun and P. Kocher. The Intel Random Number Generator.Cryptography Research Inc. white paper, Apr. 1999.

[3] P. Kohlbrenner and K. Gaj.An embedded true random number generator for fpgas. In FPGA '04: Proceeding of the 2004 ACM/SIGDA 12th international symposium on Field programmable gate arrays, pages 71–78. ACM Press, 2004.

[4] C. Petrie and J. Connelly. A Noise-based IC Random Number Generator for Applications in Cryptography. IEEE TCAS II, 46(1):56–62, Jan. 2000.

[5] Singhal, Nidhi and Raina, J P S. “Comparative Analysis of AES and RC4 Algorithms for Better Utilization”, *International Journal of Computer Trends and Technology*, ISSN: 2231-280, July to Aug Issue 2011, pp. 177-181.

[6] Jawahar Thakur, Nagesh Kumar DES ,AES and BLOW FISH: Symmetric key cryptography algorithm simulation based performance analysis International Journal of Emerging Technology and Advanced Engineering : pages 6-12,Volume 1, Issue 2, December 2011)

[7] Singh, S Preet and Maini, Raman. “Comparison of Data Encryption Algorithms”, International Journal of Computer Science and Communication, vol. 2, No. 1, January-June 2011, pp. 125-127.

[8] William Stallings “Cryptography and Network Security: Principles and Practices” 3rd Edition, PHI Ltd , 2001.

[9] Behrouz A. Forouzan “ Cryptography and Network Security” special Indian Edition, Tata Mc GRAW HILL.

[10] Elminaam, D S Abd; Kader H M Abdual and Hadhoud, M Mohamed. “Evaluating the Performance of Sysmmetric Encryption Algorithms”, International Journal of Network Security, Vol. 10, No. 3, pp. 216-222, May 2010.

[11] Hamdan.o.Alanazi, B.B Zaidan, Hamid A jalab, M.shabbir and Y. Al-Nabhani ”New Comparative study between DES,3DES and AES within Nine Factors”, journal of computing vol 2 , issue 3, march 2010.

Dr.R.Seshadri Working as Professor and Director, University Computer Centre, Sri Venkateswara University, Tirupati. He completed his PhD in S.V.University in 1998 in the field of “ Simulation Modeling & Compression of E.C.G. Data Signals (Data compression Techniques) Electronics & Communication Engg.”. He has richest knowledge in Research field. He is guiding 10 Ph.D in Fulltime as well as Part time. He has vast experience in teaching of 26 years. He has attended several national and international conferences and published number of technical papers in different national and international Journals.

T.Chalama Reddy Working as asso Professor in Narayana Engineering College, Nellore . He completed his M.Tech in J.N.T.University in 2000 in the Specialization of “Software Engineering”. His interested area is Networks Security and Cryptography .He has vast experience in teaching of 16 years. He published 3 national and 1 international conferences and 3 papers published in different international Journals.