International Journal of Scientific Research in Computer Science, Engineering and Information Technology
© 2019 IJSRCSEIT | Volume 5 | Issue 2 | ISSN : 2456-3307
DOI : https://doi.org/10.32628/IJSRCSEIT

Software Psuedo Random Number Generators to Hardware True Random Number Generators: a transition in Data-Security

Shreenabh Agrawal
Class X A, The Chanda Devi Saraf School, Nagpur, Maharashtra, India

ABSTRACT

Objective of research: To understand the basic issues in data security during vital transactions related to money or critical data in e-business and to statistically test the efficiency of PRNG –Pseudo Random Number Generators versus TRNG- True Random Number Generators used for Cryptography.

Experimental set up: A prototype of TRNG was designed. The Schematic component architecture of the prototype, the number generation code and the C Inspired Algorithm have been presented. 5000 numbers were generated using it. Another set of 5000 numbers was generated using an online PRNG.

Statistical testing: The numbers thus generated were tested by Kolmogorov Smirnov Z test for uniformity and Runs Test for Median and Mean for randomness. The results were displayed in tabular format.

Results: The statistical analysis showed that the performance of TRNG was better than PRNG for both uniformity and randomness.

Keywords : Randomness, PRNG, TRNG, Cryptography, Kolmogorov Smirnov Z test, Runs Test

I. INTRODUCTION

With a major shift of traditional industry to the e-industry and major industry giants shifting online for business, the concern for data security is a major one (Buchanan, 2017). The entire business loop from the industry to the customer and back is at risk of data leakage and manipulation due to cyber attacks. To be able to survive such cyber attacks, the random numbers used for cryptographic functions have to be truly random such that they cannot be deciphered either by forward or backward tracking. There has been a lot of research in this area of random number generation since last few decades. Researchers have come a long way from mathematical sequence numbers to quantum random numbers. Earlier, most of the requirements were handled using software generated random numbers but due to their security concerns hardware random number generators came into being. They used either physical, chemical or environmental phenomena for generating their output. This was termed as 'Entropy Source'. As the seed based pseudo random numbers had a high risk of getting compromised, true random number generation became the need of the hour.

II. METHODS AND MATERIAL

PRNG (Pseudo Random Number Generator) vs TRNG (True Random Number Generator):

Random numbers used in Cryptography can be generated by two common methods: by using a Pseudo Random Number Generator (PRNG) or by a True Random Number Generator (TRNG). PRNGs are algorithm based random number generators and are software based while TRNGs are generally hardware based. PRNGs are useful for their speed while TRNGs are useful for their randomness. As discussed, PRNGs use an algorithm to generate

random numbers based on a seed. A seed is the first input given to an algorithm which the algorithm uses to generate random numbers which can be given manually or decided by the computer. Due to this, in a PRNG, after some numbers, the sequence starts repeating. Another drawback of PRNGs is that for a particular seed and particular algorithm, only certain numbers are generated while many numbers are never generated .

To overcome such discrepancies, another method of generating random numbers is by using TRNGs. They analyse some physical phenomenon or artificial disturbances called 'noise' and generate numbers accordingly. They are capable of generating high quality random numbers. A simple example of TRNG can be a dice rolling machine or a shuffled card reader machine. At times, it can be very difficult to construct a highly efficient TRNG and statisticians often resort to complex phenomena like radioactive decay to generate such numbers. Some commonly available TRNG's and their operating principles are mentioned in Table 1 below:

Table 1: True Random Number Generators

Model	Year	Operating principle
QRBG121	2005	Photoelectric effect
Quantis-USB	2006	Beam splitter
Entropy Key	2009	Avalanche noise
Quantis-PCIe-16M	2010	Beam splitter
PQ32MU	2013	Shot noise
Ivy Bridge-EP	2013	Johnson–Nyquist noise
Alea II	2014	Reverse biased semiconductor junction
TrueRNG v2	2014	Zener noise
Infinite Noise	2014	Johnson–Nyquist noise
BitBabbler White	2015	Electromagnetic interference
TRNG	2015	Registerless linear-feedback shift registers
OneRNG	2015	Optional atmospheric noise
TrueRNG Pro	2015	Reverse-biased semiconductor junction
ChaosKey 1.0	2016	Reverse biased semiconductor junction
PQ4000KS	2016	Shot noise
TrueRNG v3	2016	Reverse-biased semiconductor junction
SwiftRNG Pro	2018	Reverse biased Zener diodes

Source: https://en.wikipedia.org/wiki/Comparison_of_hardware_random_number_generators

The following basic differences are reported related to PRNGs and TRNGs:

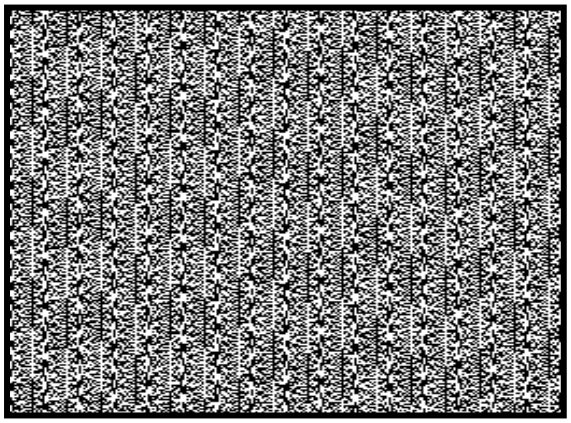
Table 2: PRNG vs TRNG

Description	PRNG	TRNG
True Randomness	No	Yes
Equal Probability of All Numbers	No	Yes
Non-degradable Randomness	Yes	No
Cost-effective	Yes	No
Portable	Yes	No
Decentralized	Yes	No
Easy to use	Yes	No
Randomness on demand service	Yes	Yes
Problem of artificial whitening	Yes	No
Insecure delivery of numbers	Yes	No
Risk of breach of secrecy	Yes	No

Data security issues related to PRNGs:

The Guardian reports," 'Computer whiz' rigged Vegas lottery number generator to produce predictable numbers a couple of times a year thus earning 16.5 million USD!" The root cause of incidents like these is the inability of PRNGs to produce Truly Random Numbers. Another such incident as reported by Dr. Dobb's Journal dates back to 1996 when two researchers, broke Netscape's SSL Encryption keys thus giving a major setback to millions of users who relied on Netscape and RSA Security (a major agency which used this service) for encryption. This

loophole was discovered by them after discovering that the time of the day and the process IDs were used to generate the random keys. In November 2007, Dorrendorf et al. from the Hebrew University of Jerusalem and University of Haifa published a paper which presented serious weaknesses in Microsoft's approach at the time. The paper's conclusions were based on disassembly of the code in Windows 2000, but according to Microsoft applied to Windows XP as well. A discrepancy was observed in the safety feature of the random numbers by Allen (2008) in his study. The picture (1) below is a bitmap image of the random numbers which are produced by the PHP rand () function in Microsoft Windows. It shows a clear connection and repetition of the numbers generated by the PRNG.



Picture 1: Bitmap image of Random Numbers

A similar flaw was also discovered in Play Station 3. In December 2010, a group calling itself 'fail0verflow' announced recovery of the private key used by Sony to sign software for the PlayStation 3 game console. The attack was made possible because Sony failed to generate a new random nonce for each signature (Borza, 2011). A more recent setback was uncovered involving Bitcoins- the e-currency. In August 2013, it was revealed that bugs in the Java class SecureRandom had flaws in implementations of Bitcoin on Android. When this occurred the private key could be recovered, in turn allowing stealing Bitcoins from the containing wallet (Chirgwin,2013).

Experimental Method to empirically test PRNG vs TRNG:

To test the propositions of researchers in support of TRNG, a prototype of a TRNG was designed. The prototype comprised of 3 components -an entropy source that produces random inputs from a non deterministic hardware process at around three 3-digit inputs per second , an Arduino board that uses a C inspired program to distil the triplet of 3 digit random inputs into a triplet of two digit high quality random inputs combining them algebraically into a 6 digit high quality nondeterministic random number and a serial monitor that displays the output for the end user.

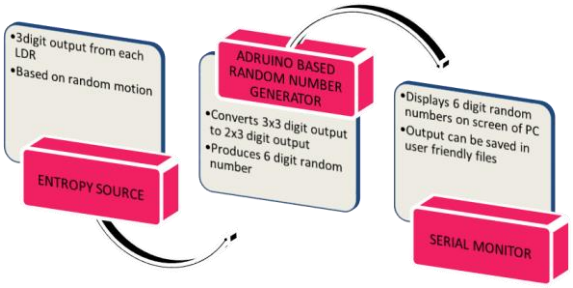


Figure 1: Schematic Component Architecture and Process

Entropy source provides a serial stream of entropic data in the form of 0 - 1024 voltage level sequences. The entropy source runs on an independent circuit and uses artificial noise generated by random interruption in light intensity due to moving soft balls. The entropy source is powered by an external power supply to run on to ensure there is no electronic connection between it and other core logic. Random inputs are passed to the Arduino board for further processing. The C inspired program housed in an Arduino Nano board deletes the first digit of the three digit random input from entropy source and converts them into high quality two digits by removing the digit which was susceptible to

prediction due to its narrow range of possible values. The triplets of two digit raw entropy samples generated by the entropy source are thus reduced to a single 6 digit high quality nondeterministic random number.

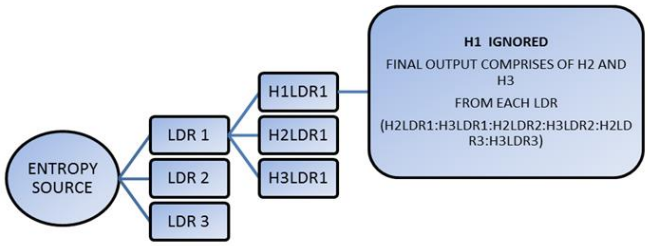


Figure 2: Number Generation Code

Algorithm designed using C inspired program on Arduino Board :

```
int LDR1 = A0;
int LDR2 = A1;
int LDR3 = A2;

int LDR1Value = 0;
int LDR2Value = 0;
int LDR3Value = 0;

int OTP1;
int OTP2;
int OTP3;

long int FinallOTP;

void setup() {
  Serial.begin(9600);
  pinMode(LDR1, INPUT);
  pinMode(LDR2, INPUT);
  pinMode(LDR3, INPUT);
}
void loop() {

  LDR1Value = analogRead(LDR1);
  LDR2Value = analogRead(LDR2);
  LDR3Value = analogRead(LDR3);
  DigitfromOTP1();
  DigitfromOTP2();
  DigitfromOTP3();

  FinalOTP();

  Serial.println(FinallOTP);
  delay(1000);

}
void DigitfromOTP1() {// Deletes hundred's digit from LDR1 input

  if(LDR1Value<100) {
    OTP1=LDR1Value;
  }
  else if(LDR1Value>100 && LDR1Value<200)
{
    OTP1=LDR1Value-100;
  }
  else if(LDR1Value>200 && LDR1Value<300)
{
    OTP1=LDR1Value-200;
  }
  else if(LDR1Value>300 && LDR1Value<400)
{
    OTP1=LDR1Value-300;
  }
  else if(LDR1Value>400 && LDR1Value<500)
{
    OTP1=LDR1Value-400;
  }
  else if(LDR1Value>500 && LDR1Value<600)
{
    OTP1=LDR1Value-500;
  }
  else if(LDR1Value>600 && LDR1Value<700)
{
    OTP1=LDR1Value-600;
  }
  else if(LDR1Value>700 && LDR1Value<800)
{
    OTP1=LDR1Value-700;
  }
  else if(LDR1Value>800 && LDR1Value<900)
{
    OTP1=LDR1Value-800;
  }
  else if(LDR1Value>900 && LDR1Value<1000)
{
    OTP1=LDR1Value-900;
  }
  else if(LDR1Value>1000 &&
LDR1Value<1024) {
    OTP1=LDR1Value-1000;
  }
}
void DigitfromOTP2() {// Deletes hundred's digit from input of LDR2

  if(LDR2Value<100) {
    OTP2=LDR2Value;
  }
  else if(LDR2Value>100 && LDR2Value<200)
{
    OTP2=LDR2Value-100;
  }
  else if(LDR2Value>200 && LDR2Value<300)
{
    OTP2=LDR2Value-200;
  }
```

```
  else if(LDR2Value>300 && LDR2Value<400)
{
    OTP2=LDR2Value-300;
  }
  else if(LDR2Value>400 && LDR2Value<500)
{
    OTP2=LDR2Value-400;
  }
  else if(LDR2Value>500 && LDR2Value<600)
{
    OTP2=LDR2Value-500;
  }
  else if(LDR2Value>600 && LDR2Value<700)
{
    OTP2=LDR2Value-600;
  }
  else if(LDR2Value>700 && LDR2Value<800)
{
    OTP2=LDR2Value-700;
  }
  else if(LDR2Value>800 && LDR2Value<900)
{
    OTP2=LDR2Value-800;
  }
  else if(LDR2Value>900 && LDR2Value<1000)
{
    OTP2=LDR2Value-900;
  }
  else        if(LDR2Value>1000        &&
LDR2Value<1024) {
    OTP2=LDR2Value-1000;
  }
}
void    DigitfromOTP3()    {//    Deletes
hundred's digit from input of LDR3

  if(LDR3Value<100) {
    OTP3=LDR3Value;
  }
  else if(LDR3Value>100 && LDR3Value<200)
{
    OTP3=LDR3Value-100;
  }
  else if(LDR3Value>200 && LDR3Value<300)
{
    OTP3=LDR3Value-200;
  }
  else if(LDR3Value>300 && LDR3Value<400)
{
    OTP3=LDR3Value-300;
  }
  else if(LDR3Value>400 && LDR3Value<500)
{
    OTP3=LDR3Value-400;
  }
  else if(LDR3Value>500 && LDR3Value<600)
{
    OTP3=LDR3Value-500;
  }
  else if(LDR3Value>600 && LDR3Value<700)
{
    OTP3=LDR3Value-600;
  }
  else if(LDR3Value>700 && LDR3Value<800)
{
    OTP3=LDR3Value-700;
  }
  else if(LDR3Value>800 && LDR3Value<900)
{
    OTP3=LDR3Value-800;
  }
  else if(LDR3Value>900 && LDR3Value<1000)
{
    OTP3=LDR3Value-900;
  }
  else        if(LDR3Value>1000        &&
LDR3Value<1024) {
    OTP3=LDR3Value-1000;
  }
}

void FinalOTP() {

  int finalunit;
  finalunit = OTP1;

  int tth=100;
  int finaltth;
  finaltth=OTP2*tth;

  long int htl = 10000;
  long int finalhtl;
  finalhtl=OTP3*htl;
 FinalOTP=finalhtl+finaltth+finalunit;
  }
```

5000 numbers were generated using this TRNG prototype. Another 5000 numbers were generated with the help of an online PRNG. The numbers were tested statistically using Kolmogorov Smirnov Z test for uniformity and Runs test of Means and Median for randomness.

III. RESULTS AND DISCUSSION

Statistical Analysis:

a) **Kolmogorov Smirnov Z test for testing uniform distribution of random numbers generated by PRNG as compared to TRNG:**

Table 3 : Kolmogorov-Smirnov Z Test for testing Uniform Distribution

	PRNG	TRNG
Kolmogorov-Smirnov Z	0.709	**0.516**
Asymp. Sig. (2-tailed)	0.696	**0.953**

Interpretation: From the table above it is seen that random numbers generated using the TRNG are highly significant (asymptotic significance =0.953) thus proving that they follow a uniform distribution with equal probability of presence of every number.

a) Runs Test based on Median

Table 4: Runs Test based on Median

	PRNG	TRNG
Z	1.698	**0.255**
Asymp. Sig. (2-tailed)	0.090	**0.799**

Interpretation: From the table above it is seen that on testing with Runs Test based on median, random numbers generated using the TRNG are highly significant (asymptotic significance =0.799) thus proving that they are statistically significant.

a) Runs Test based on Mean

Table 5: Runs Test based on Mean

	PRNG	TRNG
Z	1.641	**0.368**
Asymp. Sig. (2-tailed)	0.101	**0.713**

Interpretation: From the table above it is seen that on testing with Runs Test based on mean, random numbers generated using the TRNG are highly significant (asymptotic significance =0.713) thus proving that they are statistically significant.

IV.CONCLUSION

To prevent cyber attacks and keep data safe in crucial industries such as banking, defence, education (online exams/ evaluation), fundamental research, infrastructure (simulation), etc, the industry needs to shift from PRNGs to TRNGs as they provide a reliable solution to the issues related to randomness of numbers for data security.

V. REFERENCES

[1]. Allen B, “Are the numbers really random?”, https://www.random.org/analysis/ (2008)

[2]. Borza M, “The Sony PlayStation 3 hack deciphered: what consumer-electronics designers can learn from the failure to protect a billion-dollar product ecosystem”, https://www.edn.com/ Home/PrintView?contentItemId=4368066 (2011)

[3]. Buchanan W.J., Woodward A & Helme S., “Cryptography across Industry sectors”, pg 145-162, https://doi.org/10.1080/23742917.2017.1327221 (2017)

[4]. Chirgwin R, “Android bug batters Bitcoin wallets”, https://www.theregister.co.uk/2013/08/12/ android_bug_batters_bitcoin_wallets/ (2013)

[5]. Dorrendorf L, Gutterman Z, Pinkas D., “Cryptanalysis of the Random Number Generator of the Windows Operating System”, https://eprint.iacr.org/2007/419.pdf (2007)

[6]. Goldberg I and Wagner D, “Randomness and the Netscape Browser”, Dr. Dobb's Journal, http://www.ddj.com/windows/184409807 (1996)

Cite this article as : Shreenabh Agrawal, "Software Psuedo Random Number Generators to Hardware True Random Number Generators : a transition in Data-Security ", International Journal of Scientific Research in Computer Science, Engineering and Information Technology (IJSRCSEIT), ISSN : 2456-3307, Volume 5 Issue 2, pp. 558-564, March-April 2019.
Journal URL : http://ijsrcseit.com/CSEIT1952132