Using Efficient TRNGs for PSEUDO Profile in National eID Card

https://doi.org/10.3991/ijes.v6i1.8357

Blerim Rexha, Dren Imeraj, Isak Shabani(✉)
University of Prishtina, Prishtina, Kosovo
isak.shabani@uni-pr.edu

Abstract—Applications that requires true random number generator (TRNG), which uses raw analog data generated from any noise source in nature, must convert the source normal distribution to uniform distribution. Many up to date implementations convert the raw analog data into digital data by employing a comparator or a Schmitt trigger. This method wastes a large amount of random input data, lowering the throughput of the TRNG. In new national electronic identity card (eID) beyond the true identity of his bearer and to address the increasing concern of user privacy while doing business in Internet an additional pseudo profile is set. This pseudo profile uses 20-byte random value generated by database server, using a script during personalization process. In this paper, we present a novel algorithm that enables efficient distribution conversion in low power devices. The low memory requirements and efficient processing make it suitable for implementation low power cryptographic devices but also in complex personalization systems. Furthermore, we compare the random data generated by our efficient TRNG vs. those generated by database server.

Keywords—eID, privacy, security, random, TRNG

1 Introduction

Random number generation for cryptography applications is usually a critical operation. The random data required is often used to form encryption keys, in this case, the random number generator must be practically unpredictable as described in [1]. Random number generator is used also for setting the [PSEUDO] profile in new biometric national identity cards. In order to protect the user privacy in Internet, beyond the real profile [IDENT], the national ID card has optionally the second profile, so called the [PSEUDO] profile. The [PSEUDO] profile consists of 20 bytes, expressed in hex format as 40 characters, following specific format [2]. This pseudonym identifier must be unique and it provides no traceability of the card holder's real identity to the web service provider. Internet according to Internet World Stats Internet usage growths for period 2000-2015 was 753% and Internet penetration in World level is about 42% [3]. With this high usage of Internet, as main communication platform of a

modern user, his security and privacy is becoming very important feature and random number generation is a key to its wide acceptance.

Predicting the outcome of the random number generator (RNG) can lead to access of encrypted data by an unauthorized third party or in case of national biometric card to identity theft. An example of this is the exploitation of Dual Elliptic Curve Deterministic Random Bit Generator's flaw is presented in [4] by the NSA. To minimize the possibility of outcome prediction, critical cryptography applications employ true random number generators (TRNGs) when feasible. These RNGs use a physical noise signal as the source of the random data. Since the signal output from some noise sources is practically impossible to predict, these signals are ideal for use as RNGs. Unfortunately, the distribution of the values in a noise source is usually not suitable for use in cryptography.

Cryptography applications usually require the data distribution to be uniform [5], meaning the probability that the RNG will generate any value in the value domain is always the same. On the other hand, many noise sources that are suitable for use in RNGs usually have a normal distribution according to [4]. The signal coming out of these sources oscillates around a specific point, i.e. 0V or some DC offset, and the possibility that some value X will be generated drops rapidly with the increase in magnitude of X. In order to use these data in cryptography applications, some means of converting the distribution from normal to uniform must be applied first.

2 National biometric eID card

A biometric identity card (ID), the credit card size, contains biometric information about its holder in printed form as well as in electronic format stored in microprocessor and is used to authenticate its bearer in real as well in Internet world. Such electronic ID card uses proven smart card technology to communicate to outside world, based on guidelines issued by the International Civil Aviation Organization (ICAO), a body run by the United Nations with a mandate for setting international travel document standards, Doc 9303 [6]. In order to use this eID card in the Internet for authentication the real identity [IDENT] profile is set in every ID card. This profile contains a digital X.509 certificate, in compliance with ICAO Public Key Infrastructure (PKI), signed by country issuing certification authority.

2.1 Authentication profiles

Government of Kosovo has issued first biometric national ID cards in December 2013, thus becoming first country supporting the new Supplemental Access Control (SAC) protocol for mutual authentication [7]. The Kosovo national ID card host three applications, as presented in Figure 1 and it uses SLE 78CLX1280P 16 bit crypto processor from Infineon. It has 128 kByte EEPROM and supports RSA 4096 key bit length, ECC up to 521 bit and 3DES and AES up to 256 bit length and the communication with outside world is done using the Near Field Communication (NFC) protocol [8].

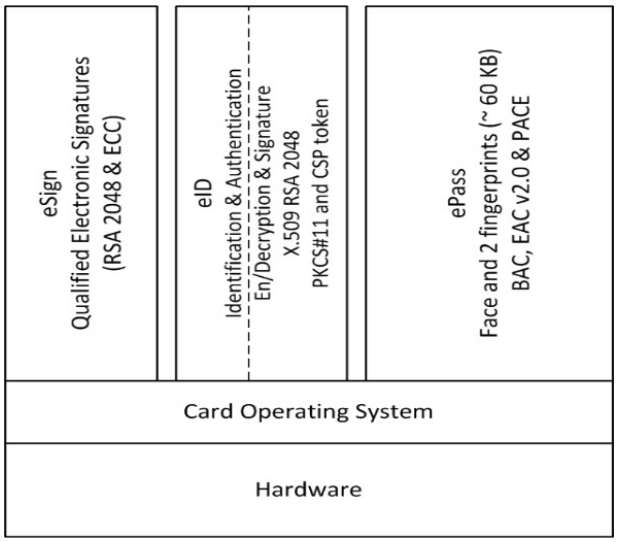

Fig. 1. National biometric ID card hosted apps

There has been critics expressed by [9] regarding the usage of X.509 certificates and PKI for user authentication in Internet. There are cases where user should not reveal his real identity to web service provider, such shopping in the Internet, voting or online petitions. Therefore, IACO has made optionally, the second profile in eID card, i.e. the [PSEUDO] profile. In general, the national ID card middleware communicates using the Public Key Cryptographic Standard (PKCS) #11 and Crypto Service Provider (CSP) library with cryptographic interested apps

The web authentication with biometric ID card is done using X.509 certificate using:

- [IDENT] i.e. the real profile or
- [PSEUDO] i.e. the anonym profile.

Whereby the corresponding private key, for both profiles, never leaves the card [2]. The user certificates are propagated into user personal store during the insertion card event into smart card reader, via middleware software. The middleware propagates also the root and intermediate certificates in proper system operating store, and correct installed [PSEUDO] certificate is presented in Figure 2.

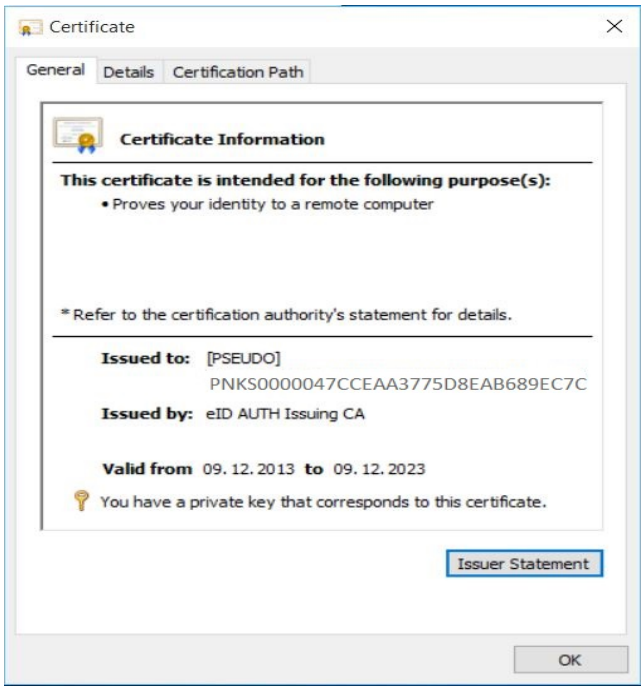

Fig. 2. X.509 [PSEUDO] profile

2.2 Randomness in [PSEUDO] profile

The [IDENT] certificates holds the real attributes of citizen as well as the 2048 bit public key. The [PSEUDO] certificate as any X.509 certificate and holder's subject is a pseudonym generated by following sequence [2]:

<pseudonym> ::= <identifier>
<country code>
<region code>
<generation counter>
<sequence of numbers>
[<check sum>] (1)

A sample pseudonym identifier would look like this:
PNKS0000047CCEAA3775D8EAB689EC7CFF28D0F243B7DDE700 (2)

Whereby the random value, in hex format, from (2) is:
RND=0x47CCEAA3775D8EAB689EC7CFF28D0F243B7DDE70 (3)

This random value, as presented in equation (3), is generated by database server during the personalization process using built in SQL function in data base server at

hardware facilities at Ministry of Internal Affairs, responsible for issuing national ID card. The SQL pseudo code, as currently used in data base server, is listed in Figure 3.

```
GenerateSQLRandom (SQLRND):
  Seed = GetDate()
  Lower1 = 1
  Upper1 = 499999
  Lower2 = 500000
  Upper2 = 999999

  RND1 = ROUND(((Upper1 - Lower1 -1) * RAND() + Low-
er1), 0)
  RND2 = ROUND(((Upper2 – Lower2 -1) * RAND() +  Low-
er2), 0)

  HexRND = HashBytes('SHA1', RND1+Seed+RND2)

  SQLRND = “PNKS” + “00000” + Upper(HexRND) + “0”
RETURN
```

Fig. 3. SQL pseudo code random number generator

The final aim of this random identifier is to serve as pseudonym, for real eID card holder, and it provides no traceability of the card holder's real identity to the web service provider.

We have analysed the code, as presented in Fig. 3, and we have run this function with overloaded request. For this purpose, we have written a small application, in C# programming language, that sends many requests per second to database server to generate the [PSEUDO] values. The database server saved these values in a binary file. In real life scenario, this would as the case where in a large populated country many citizens are applying simultaneously, either through Internet or municipality offices, for the new eID card. The file, with generated random date, is viewed and analyzed by [10], whereby is easy concluded that the generated data are not truly random, as presented in Figure 4. In file are many 2-byte repeated blocks, as presented with dash red lines in Figure 4.

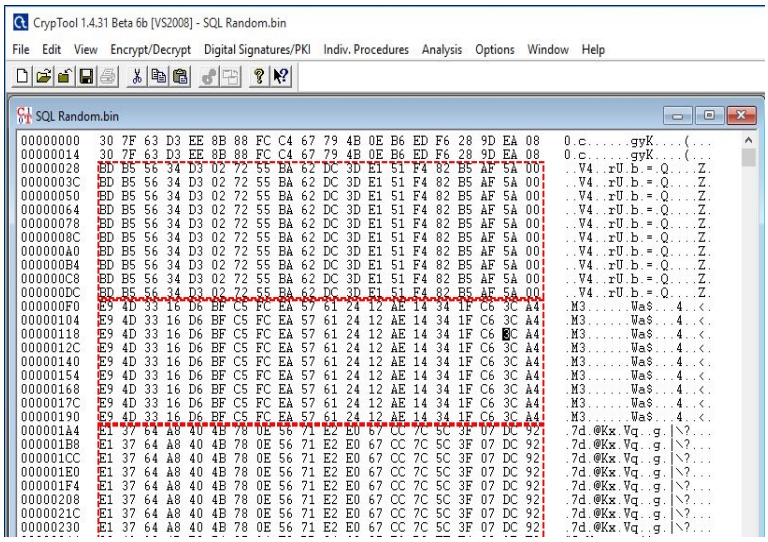

Fig. 4. SQL random data

Therefore, in this paper we propose a novel solution, using the new efficient TRNG for generating random values for [PSEUDO] data profiles to be used in new national eID cards.

3 Random data generators

Converting data with a normal distribution to data with a uniform distribution can be easily achieved if data losses are tolerated. A simple way to achieve this conversion is by using a comparator with a threshold positioned at the DC offset of the noise signal as shown in [11]. An approach of using environmental noise measurements for generation of truly random number sequences is presented in [12]. This TRNG, as described in [12], uses signals from the built in microphone that is ubiquitous most current personal computers and other personal information processing system data.

Since the noise has a normal distribution, represented by a Gaussian curve as presented in Figure 5, with the peak of the curve located at the DC offset, the probability that a certain value is located below the DC offset is the same as the probability that the value is located above the DC offset. If the output of the comparator is represented as a binary value, i.e. binary 0 if the noise signal is below the DC offset and binary 1 if the noise signal is above the DC offset, these data can be packed into bytes. Since each bit in any generated byte has the same probability of being either binary 1 or binary 0 as any other bit, the distribution of the generated bytes will always be uniform.

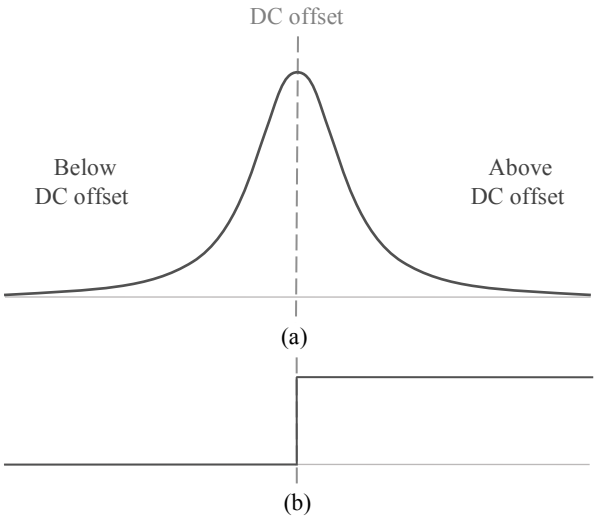

Fig. 5. Comparator output (b) depending on the noise signal input (a)

The main problem with the solution above is the data losses during signal quantization as described in [13, 14]. Since the noise signal is analog, each sample taken from the signal using the solution presented above contains more information than whether the signal is above or below the DC offset. This implies that the solution presented above is not efficient in terms of data throughput.

In Figure 6 is presented the original distribution of over 130 million samples taken directly from the noise source. Clearly, the distribution in Figure 5 is far from being a normal distribution.

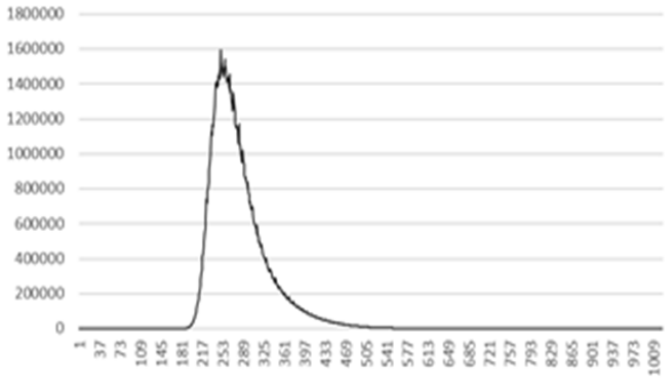

Fig. 6. The original distribution of the noise signal

Figure 7 shows the distribution of the values obtained by calculating the difference of two consecutive samples, Xi and Xi+1, from more than 130 million samples used previously. Figure 7 shows a clear normal distribution. If a signal with the distribution, as presented in Figure 6 is fed into a comparator, no matter where the threshold is specified, the probability of getting a binary 1 and the probability of getting a binary 0 out of the comparator will not be the same since the distribution is not symmetrical around the DC offset.

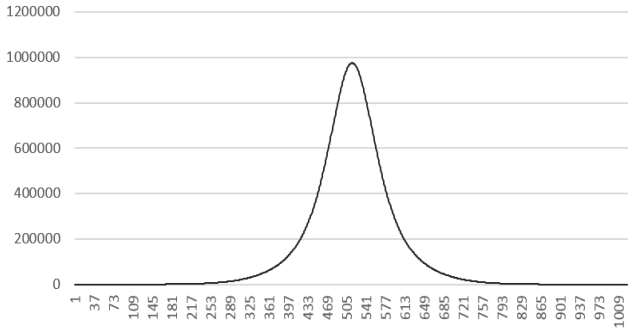

Fig. 7. The distribution of the values obtained by calculating the difference between two consecutive samples

Another problem with the solution above is the difficulty modifying the noise signal before sampling the bits. Some noise sources do not originally have a normal distribution of values, therefore they require some operations to be applied in the noise signal in order to make them suitable for conversion. An example of this approach is the output of a reverse PN junction noise source.

To overcome the problems described above, in this paper is presented an algorithm that uses samples of the noise signal quantized using an analog to digital converter. This way, the device running the algorithm can use more information from each sample as well as be able to pre-process the samples in order to alter the distribution of the input signal before feeding the samples to the distribution conversion algorithm.

3.1 True random raw data

In order to test the algorithm presented in this paper, special hardware has been designed and implemented. The noise source used in this implementation is a reverse biased PN junction as described in [15]. This noise source is truly random since its signal is a result of two non-deterministic physical phenomena, quantum tunneling and avalanche multiplication [16, 17]. This noise source has been chosen for its good noise quality, high frequency and high stability.

The noise generation circuit consists of a bipolar junction transistor (BJT) with its base-emitter terminals reverse biased. Under high enough reverse voltage on those terminals, the current flowing through this transistor will become noisy due to the effects mentioned above. This current is amplified by another BJT transistor which is

later AC coupled. The AC coupling will remove the DC offset generated by the amplifying BJT transistor since the magnitude of that DC offset is too high compared to the magnitude of the noise signal. To make the noise signal suitable for the analog to digital converter, a new DC offset is added on the noise signal. Since the impedance of the DC offset generation resistors and the AC coupling capacitor is high, the analog to digital converter may alter the noise signal while sampling it as shown in [18]. To overcome this issue, a buffer is added between the analog to digital converter and the DC offset generator resistors. This buffer consists of an operational amplifier configured a unity gain buffer. The reason for using a unity gain configuration on the operational amplifier is because of the gain-bandwidth product associated with the operational amplifier. As described in [19], increasing the gain of the operational amplifier reduces its bandwidth. The problem introduced by the lack of amplification is that the noise signal level is low. To overcome this problem, a suitable reference voltage for the analog to digital converter is used.

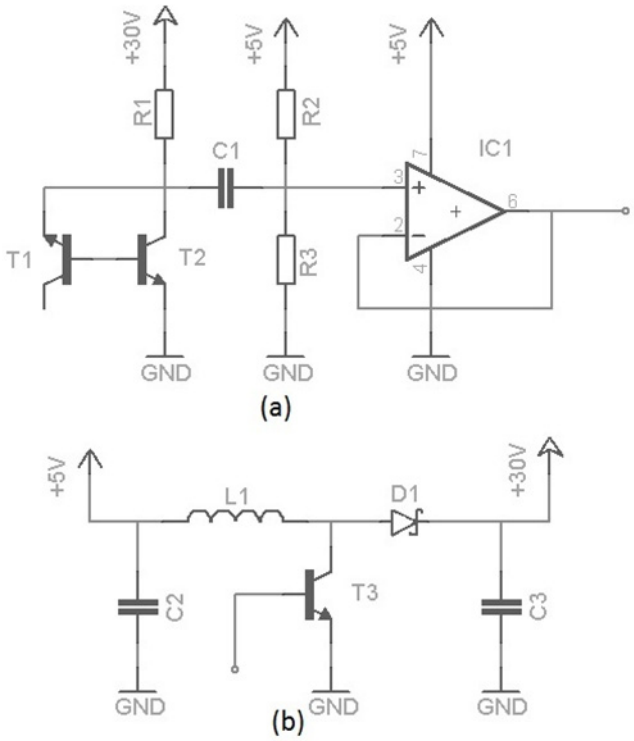

Fig. 8. Noise generator circuit (a) and boost converter (b)

Figure 8 presents a schematic diagram of the implemented noise generation circuit, as true raw random data generator. The reverse bias voltage is generated by a boost converter that converts 5V to 30V. Feeding 30V to the noise generation and amplification stage will generate about 400mV peak AC coupled. The DC offset generation

stage will generate an offset of 550mV, which will make the signal suitable for the analog to digital converter that uses a 1.1V reference voltage. An important thing to consider is the values of the AC coupling capacitor and the DC offset generation resistors. Since this stage acts as a high pass filter, it is important to choose the values properly so that important low frequency components of the noise signal is not attenuated.

3.2 Shifting to uniform distribution

Suppose a set consisting of a finite number of 10 bit randomly generated values with a normal distribution. Also suppose that this set is represented as a binary matrix, with its rows containing the sample's bits and the columns showing the samples themselves, as presented in Table 1.

Table 1. An example of a binary matrix of samples

MSB (9)	0	1	0	0	1	0	1	1	0	0
8	1	1	0	1	0	1	1	1	0	1
7	1	0	1	1	1	1	0	1	0	1
6	1	1	1	1	0	1	1	1	1	0
5	1	1	0	1	1	0	0	0	0	1
4	1	0	1	0	1	1	0	0	1	1
3	1	0	1	1	0	1	1	0	0	0
2	0	1	0	1	0	0	1	1	0	0
1	1	0	1	0	0	0	1	0	1	1
LSB (0)	1	1	1	0	1	0	1	0	1	0

If one calculates the average time it takes a bit position (on a specific row) to change its state, it can be concluded that the time interval increases exponentially going from the least significant bit (LSB) to the most significant bit (MSB). In order for this data to be used as random data, the average time it takes the LSB to change its state must not be higher than 1 sample, otherwise each two following samples would have the same value.

Since the LSB takes 1 sample to change its state, the time it takes other bit positions to change their state can also be represented in samples. To shift to uniform distribution, each bit position must have the same average time it takes to change its state, 1 sample.

The goal is to reconstruct a new set of N-bit values with a uniform distribution. Since it is known that the LSB takes an average of 1 sample to change its state, each LSB can be used to construct the new set of N-bit values. But this is not true for the other bit positions. Since other bit positions take an average of more than 1 sample to change their state, these bit positions must wait before they are used. The waiting time depends on their average time it takes them to change the state. If a bit position takes an average of 2 samples to change its state, only one in two samples from the original

set can pass this bit to the new set. In the meantime, the bit positions that are waiting can be XORed.

By using the bit positions not more than their average time needed to change their state in the original set, it is ensured that each bit in the new set has an average time of state change of 1 sample.

3.3 Algorithm implementation

In Figure 9 is presented the C pseudo code, in accordance to the algorithm definition in this section. The array holding the Thresholds can be experimentally determined using the algorithm defined in Figure 10.

```
ConvertToUniform (Data, DataLength):
  n = 0
  XORSample = 0
  XOR_Counts = {0, 0, 0, 0, 0, 0, 0, 0, 0, 0}
  Thresholds = {1, 2, 4, 8, 16, 32, 64, 128, 256, 512}

  WHILE n < DataLength
    XORSample = XORSample XOR Data[n]
    n = n + 1
    FOR i = 0 TO 9
      XOR_Counts[i] = XOR_Counts[i] + 1

      IF XOR_Counts[i] IS EQUAL TO Thresholds[i]
        IF BIT i OF XORSample IS 0
          ADD BIT 0 TO OUTPUT
        ELSE
          ADD BIT 1 TO OUTPUT
        ENDIF
        XOR_Counts[i] = 0

        IF i IS EQUAL TO 9
          XORSample = 0
        ENDIF
      ENDIF
    ENDFOR
  ENDWHILE
RETURN
```

Fig. 9. C pseudo code for conversion algorithm

An additional function inserts the bit given as a parameter to the output data. At this point, the data constructed by this function has a uniform distribution since the bits given as a parameter the funcion have passed through the distribution conversion function.

In order to make use of most of the data generated by the noise source, the Thresholds array of the algorithm must be determined based on the characteristics of the noise source. Index N of the Thresholds array holds the average number of samples that need to be read before the state of bit position N changes. Since the characteristics of each noise source differ, in order to achieve the best results, the Thresholds array must be determined experimentally. This can be done once, if the characteristics of the noise source do not drift, or the values of the Thresholds array can be updated each time a new sample is obtained from the noise source. The first method requires less computational resources but can lead to errors in the generated random data, while the second method is more resource intensive but offers better reliability.

Since the Thresholds array holds the number of samples that need to be read before a specific bit position can be used, the array can be constructed by calculating the number of bit flips on a large set of samples and dividing the number of samples in the set by the number of bit flips.

The C pseudo code, as presented in Figure 10, can be used to determine the Thresholds array for a given sample set.

```
CalculateThresholdsArray (SampleSet, SampleCount,
                    Thresholds, SampleSize):
  FOR n = 0 TO SampleSize – 1
    IF BIT n OF SampleSet[0] IS 0
      LastBit = 0
    ELSE
      LastBit = 1
    ENDIF
    FlipCount = 0

    FOR i = 1 TO SampleCount – 1
      IF BIT n OF SampleSet[i] IS 0
        CurrentBit = 0
      ELSE
        CurrentBit = 1
      ENDIF
      IF (LastBit IS 1 AND CurrentBit IS 0) OR
      (LastBit IS 0 AND CurrentBit IS 1)
        FlipCount = FlipCount + 1
        LastBit = CurrentBit
      ENDIF
    ENDFOR
    Thresholds[n] = SampleCount / FlipCount
  ENDFOR
RETURN
```

Fig. 10. C pseudo code for calculating the *Threshold* array

The sample set fed to the function as defined in Figure 10 must be as large as possible in order to get good results.

4 SQL RND vs. true RNG

The quality of the algorithm is measured based on the performance and the algorithm in terms of data throughput and based on the quality of the generated random data.

The throughput of the algorithm is based on the Thresholds array. The number of uniformly distributed bits that can be obtained from an N bit normally distributed input value, as presented in equation (4).

$$M = \sum_{i=0}^{N} \frac{1}{Thresholds[i]} \tag{4}$$

Using Equation (4), it can be concluded that the *Thresholds* array in Section III has an output of almost 2 uniformly distributed bits for each normally distributed input sample. Since the threshold for the LSB will always be 1 (otherwise the data cannot be used as random data, as described in in this section), and the thresholds for the other bits are always finite, the effective output of the algorithm is always more than 1 bits for each sample. Compared to the traditional method of extracting uniformly distributed data using a comparator or a Schmitt trigger, the algorithm offers higher throughput since the traditional methods offer a throughput of 1 bit for each sample.

The quality of the generated data is measured using a set of randomness testing tools offered by the CrypTool software [10], most of which are proposed by NIST and Intel [20, 21].

While measuring the quality of the pseudo-random data generated by the SQL pseudo random number generator, it has been noticed that on high pseudo-random number generation request rates the SQL pseudo-random number generator fails to generate new data. The Figure 8 shows the visualization of the random data using Vatanyi algorithm, using [10].

The Figure 10 (a) shows the curve generated with SQL random data, whereas the (b) part shows the Vitanyi curve generated using the TRNG data. The diagram shows the proportionality constant as a function of the number of points. It is standardized to the mean (mean = 1.0).

The mean and the standard variation are also output. It is to be expected:

- that with a sequence that is not uniformly distributed the curve will decline, and
- that in a sequence that is not independent the output graph will rise.

The SQL data base server fails its randomness only if it is overloaded with many request per seconds due to seed value *GetDate(),*which precision is second based and the narrow band set for possible random values, as presented in Code 1.

Alongside the visual tests, the data has been tested with randomness tests as well. The Table 2 shows the results for the SQL generated pseudo-random data compared with the TRNG generated 128 kB random data and theirs maximal allowed values.

Alongside the visual tests, the data has been tested with randomness tests as well. The Table 2 shows the results for the SQL generated pseudo-random data compared with the TRNG generated 128 kB random data and theirs maximal allowed values.

Another test, also proposed in [21], which indicates missing patterns in the generated data is the entropy. The entropy indicates the level of "chaos" in the data. If the generated data is organized in bytes (8 bit words), the resulting entropy of the 128 kB of data used for the tests, as presented in Table 2, using [10] the value of entropy comes out 7.99. With such a high entropy, compression algorithms would not be able to find any patterns in the data, making it impossible to compress the generated random data.

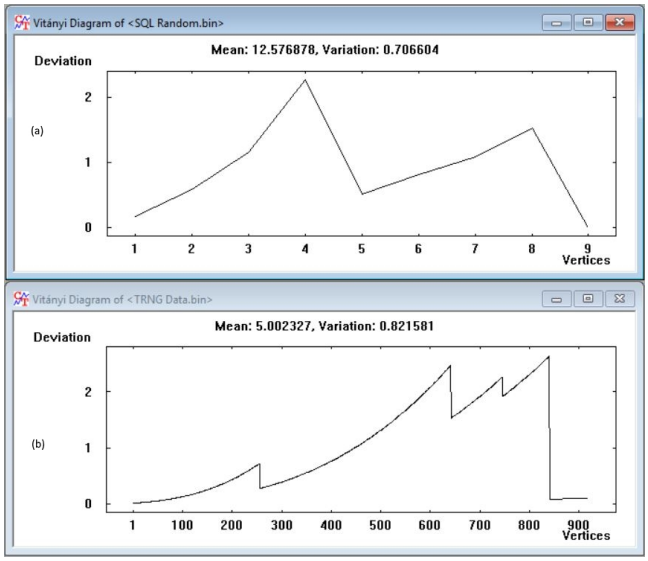

Fig. 11.C pseudo code for calculating the *Threshold* array

Table 2. Randomness test results

Test	SQL	TRNG	Max.
Frequency test	11.97	0.623	3.841
Poker test	23.04	5.423	14.07
Runs test	95.42	4.217	9.488
Serial test	17.20	2.167	5.991

5 Conclusions

The algorithm presented on this paper increases the distribution conversion performance compared to the traditional method of using a comparator or a Schmitt trigger for generating random data.

The [PSEUDO] X.509 profile can be used in many field of current e-Services, even there are critics expressed by [9] regarding the usage of X.509 certificates and Public Key Infrastructure. Nevertheless, the proposed solution using [PSEUDO] X.509 certificate strengthens the fundamental citizen right, i.e. the freedom of speech, as it does not reveal his real identity under the assumption that [PSEUDO] profile contains true random value, as one can generate by proposed solution.

The implementation proposed in this paper is lightweight and can easily be implemented in microcontrollers or other embedded applications. The same microcontroller can be also used to sample the analog noise signal which makes the TRNG cost effective and small in size.

The algorithm can be dynamically adopted to obtain as much data from the noise source as possible. This is done by updating the Thresholds array every time a new sample is obtained.

The implementation of the distribution conversion in software also offers the flexibility of specifying the output data format, as well as the distribution itself. The user can specify whether the distribution should be converted or not.

To test the algorithm, a special electrical circuit has been built. This circuit uses the reverse biased Emitter-Base junction of a BJT transistor to generate a noise signal. This signal is then fed to the analog to digital converter of a microcontroller, which samples the noise signal and feeds it to the algorithm. Prior to developing the algorithm, the circuit was used to create a cache of 130 million 10 bit samples of the noise. These data is used to test the algorithm during the development. The algorithm passes the frequency test, poker test, runs test and serial test. It is important to note that the data in generated by the algorithm has an entropy of 7.99 when packed in 8 bit words.

In conclusion, the algorithm presented in this paper is suitable for embedded security applications that require a TRNG with a relatively high data throughput. The noise signal can also be different from pure Gaussian and then pre-processed in the microcontroller before being fed to the algorithm.

The used SQL random number generator in data base server should not be used as it is, due to it’s not capability of generating true random numbers on overloaded working environment.

The future work remains to add this efficient TRNG as a dedicated hardware for other application, interested in reliable random data using any well-proven encryption protocols for securing data transfer.

6 References

[1] Collantes MH. and Escartin JCG, (2016), 'Quantum Random Number Generators', [Online]. Available: https://arxiv.org/pdf/1604.03304.pdf, (Accessed on January 17th, 2018).

[2] Giesecke and Devrient GmbH, (2015), 'Help files and technical notes for HIGHSEC eID App, Middleware', [Online]. Available: http://mpb.rks-gov.net/eID.html (Accessed on January 17th, 2018).

[3] Internet World Stats, (2018, December 15), [Online]. Available: http://www.internetworldstats.com/stats.htm, (Accessed on December 15th, 2017).

[4] Perlroth Nicole et al (2013), "N.S.A. able to foil basic safeguards of privacy on web". International New York Times, [Online]. Available: http://www.nytimes.com/2013/09/06/us/nsa-foils-much-internet-encryption.html, (Accessed on January 17th, 2018).

[5] Marton Kinga et al, "Randomness in Digital Cryptography", Romanian Journal of Information Science and Technology, Volume 13, pp219–240, Number 3, 2010.

[6] ICAO Standard Document 9303, Machine Readable Travel Documents, Seventh Edition, 2018.

[7] Ministry of Internal Affairs, Kosovo, (2013), [Online]. Available at http://www.mpb-ks.org/repository/docs/MIA_Bulletin,_31.12.2013.pdf (Accessed on January 17th, 2018).

[8] Infineon, (2015) Technical details for SLE 78CLX1280P, [Online]. Available: http://www.infineon.com/ (Accessed on January 17th, 2018).

[9] Stefan A Brands "Rethinking Public Key Infrastructure and Digital Certificates", Building in Privacy (Ph.D. thesis updated as book). The MIT Press, ISBN = 0-262-02491-8, 2000.

[10] CrypTool, Open-source program for cryptography and cryptanalysis, [Online]. Available: www.cryptool.org, (Accessed on January 17th, 2018).

[11] NXP, Analog Comparator Tips and Tricks for the MC9S08QG MCU, [Online]. Document Number: AN3552, 2007 available: http://www.nxp.com/assets/documents/data/en/application-notes/AN3552.pdf, (Accessed on January 17th, 2018).

[12] N.G. Bardis et.al, 'True Random Number Generation Based on Environmental Noise Measurements for Military Applications', Proceedings International Conference on Signal Processing, Robotics and Automation, ISBN: 978-960-474-054-3, 2009.

[13] D. Marco, and D.L. Neuhoff, 'The Validity of the Additive Noise Model for Uniform Scalar Quantizers', IEEE Transactions on Information Theory (Volume: 51, Issue: 5), 2005.

[14] Jay Hodgson 'Understanding Records: A Field Guide To Recording Practice", Bloomsbury Academic, ISBN 978-1-4411-5607-5, 2010.

[15] Pralgauskaitė Sandra, Palenskis Vilius, and Matukas Jonas, (2013) 'White noise peculiarities in diode structures', Noise and Fluctuations (ICNF), 2013

[16] T. Umezawa, K. Akahane, A. Kanno, 'RF response of PIN photodiode with avalanche multiplication using quantum dots', Lasers and Electro-Optics Pacific Rim (CLEO-PR). 2013.

[17] Xiangyi Guo, A.L. Beck, Xiaowei Li, 'Study of reverse dark current in 4H-SiC avalanche photodiodes', IEEE Journal of Quantum Electronics (Volume: 41, Issue: 4), 2005

[18] G. Xing, S. H. Lewis, and T. R. Viswanathan, 'Self-biased unity-gain buffers with low gain error, 'IEEE Trans. Circuits Systems II, 2009.

[19] Zahra Dorostghol; Noushin Ghaderi; Majid Ebnali-Heidari, 'A novel high gain-bandwidth product and low power band-pass preamplifier using active inductor', Electrical and Electronics Engineering (ELECO), 2015.

[20] Andrew Rukhin et al., "A Statistical Test Suite for Random and Pseudorandom Number Generators for Cryptographic Applications" NIST, Washington, 2010.
[21] M. John, 'Intel® Digital Random Number Generator (DRNG) Software Implementation Guide, Intel. 2014, Available: https://software.intel.com/sites/default/files/managed/4d/91/DRNG_Software_Implementation_Guide_2.0.pdf, (Accessed on January 17th, 2018).

7 Authors

Blerim Rexha graduated with distinguished mark from University of Prishtina, Faculty of Electrical and Computer Engineering. He received the Ph.D. from Vienna University of Technology, Institute of Computer Technology in 2004 in field of data security, smart cards and online security. Currently he is head of computer engineering department at the Prishtina University, where he teaches the courses about data security, network security and software engineering.

Dren Imeraj received his master degree from University of Prishtina in 2016. His research interests include algorithms, security, privacy and electronic wearables and mobile apps.

Isak Shabani graduated from University of Prishtina, Faculty of Electrical and Computer Engineering. He received the PhD from University of Prishtina in 2012. His research interest include algorithms, distributed systems and human computer interaction

Article submitted 06 January 2018. Resubmitted 08 February 2018. Final acceptance 23 February 2018. Final version published as submitted by the authors.