Steganographic Stream Cipher Encryption Using True Random Number Generator

Jan Jakub Tatarkiewicz and Wieslaw Kuzmicz
RANDAEMON sp. z o.o.
Warsaw, Poland
(kuba.tatarkiewicz, wieslaw.kuzmicz)@randaemon.com

***Abstract*—A novel, computationally simple method of hiding any message in the stream of random bits by using a secret key is presented, and a hardware-based true random number generator that can be used to produce a stream of random bits is described. The source of entropy in this generator is the random emission of low-energy electrons from the nuclear decay of the ^{63}Ni isotope.**

***Keywords*—encryption; steganography; random number generator; nuclear decay**

I. Introduction

The method described in this paper can be labeled ***B**ury **A**mong **R**andom **N**umbers* (BARN). It can be also called *a needle in the haystack* search problem for decryption. The idea of this type of encryption is similar to spreading information between channels as originally patented by the actress Hedy Lamarr [1] and is nowadays used in WiFi and Bluetooth networks. Unlike present cipher standards e.g., AES [2], the method requires very low computational resources and perfectly fits applications in Internet-of-Things devices. In principle, any source of a stream of random bits can be used in BARN. In this paper, we utilized a true random number generator (TRNG) using nuclear decay as the entropy source that has been developed in-house, and its prototypes tested extensively. In Section II the encryption method is presented. Section III describes our TRNG. Encryption and decryption examples are shown in Section IV. Section V summarizes the paper and outlines the future directions of the works.

II. Encryption Method

A. The idea of a Combination of Steganography and Encryption with a Secret Key

The BARN algorithm combines the steganographic (*stegano {Greek} concealed, covered*) method [3] of concealing data inside a truly random string of bits and a cryptographic (*crypto {Greek} hidden, secret*) key [4], [5] that allows random distribution of this data, essentially creating a symmetrical stream cipher.

B. BARN Algorithms

Definitions:

message: a stream of bits to be encoded, e.g., a text file *message.txt* with ASCII-encoded text.

carrier: a stream of random bits where some bits will be replaced by message bits.

key table: an array of integers indicating which bits of the carrier will be replaced by message bits.

key: a stream of bits used to generate the entries in the key table

encoded message: a stream of bits in which some bits of the carrier were replaced by message bits.

decrypted message: decrypted stream of bits.

BARN encoder – step 1: generation of the *key table*:

1. Get secret key k from a file *key*, externally via a network, or from a keyboard entry

   For example, Diffie-Hellman [4], [5] key k negotiated with the other party can be used – for the public-domain Curve25519 algorithm [6] one gets a 256-bit key. A stream of random bits generated by a TRNG negotiated with the other party can also be used.

2. Convert the key k into a set of m non-zero integers from the range {1, …, 7}:

   On average there will be $m=INT(256/3*(1-1/8))=74$ non-zero integers.

   a. discard the least significant bit of key k,

   b. take consecutive 3 bits from the key k,

   c. if all 3 bits are equal to zero, then discard this group, otherwise convert them into k_j entry in the *key table*,

   d. repeat *b.* and *c.* until all bits of key k are used.

3. Multiply each of the integers k_j by 8 or another integer larger than 1.

   This step enlarges steps between *carrier* bits to be replaced by *message* bits. It is not necessary if the length of the *encoded message* matters – enlarging steps several times makes the *encoded message* longer but "more random" i.e., the randomness tests are easier to pass. However, the randomness of the *encoded message* is not critical.

Note that the method of generation of the *key table* described above is not the only possible, other algorithms can be used as well.

BARN encoder – step 2: encoding

4. Read the *carrier*: a stream of random bits from a file or RNG directly:

a. insert the first bit of the *message* instead of the bit k_1 of the *carrier*.

b. insert the second bit of the *message* instead of the bit k_1+k_2 of the carrier.

c. repeat inserting bits of the *message* into the *carrier* as a bit number $k_1+k_2+...+k_j$ where $j \in \{1, m\}$ until all bits of the *message* are inserted.

d. if the number of bits in the *message* is larger than *m*, then reuse the k_j set of numbers repeatedly starting again with k_1, etc. until the whole *message* is inserted into a random stream of bits.

e. if k_i was the last key table entry used to encrypt the *message*, then add k_{i+1} *- 1* random bits at the end of the *encrypted message.*

5. Save the *encrypted message* into a binary file *encrypted.bin*.

BARN decoder – step 1: retrieval of the key table

1. Get key *k* from a file *key*, externally via a network, or from a keyboard entry
2. To retrieve the *key table*, repeat steps 3 and 4 of the BARN encoder.

BARN decoder – step 2: decoding

3. Read contents of binary file *encrypted.bin* into memory and proceed as follows:

a. use bit k_1 of this file as the least significant bit of decrypted text, storing it in the memory as a binary string *decrypted message*

b. use bit k_1+k_2 of the file *encrypted.bin* as the second least significant bit of decrypted text, and add it to previously created binary string d*ecrypted message*

c. repeat reading bit number $k_1+k_2+...+k_j$ where $j \in \{1, m\}$ until all bits of the message are decrypted and stored into binary string *decrypted message*

d. if the number of bits in the encrypted message is larger than *m* i.e., there are more bits left in the file *encrypted.bin* after the whole set of k_j numbers were exhausted, then reuse the k_j set of numbers repeatedly starting again with k_1, until all bits of the message are decrypted

4. Store the decrypted message in an ASCII text file *decrypted.txt*.

III. True Random Number Generator

A. Method of Generation of Random Numbers

One could use pseudo-RNG (algorithmic) in the BARN method but such use would be prone to easier breaking by the analysis of a pattern of bits. Several true random number generators based on various physical effects were proposed [7 - 13]. We build a prototype RNG device [14] based on a purely quantum effect: random emission of low energy electrons during nuclear beta decay. Entropy is extracted [15] via comparisons of time differences between two pairs of beta decays recorded by the device [16].

The source of randomness in our TRNG is a tiny piece of a ^{63}Ni foil coupled with a detector of low-energy electrons. Electrons are emitted randomly in space and time. Each electron that hits the detector produces a current pulse, which is amplified and shaped to facilitate measurement of its time of arrival at the detector. A time-to-digital converter registers the time of an electron hit and stores this time in local memory. A logic circuit compares time differences between hits of two consecutive pairs of electrons. "1" bit is generated if the first time is longer than the second one, and "0" otherwise (Fig. 1). This convention can of course be reversed as well. Thus, for each random bit, four pulses are used. The random bits generated are stored in a local register and become available to a computer via a USB interface.

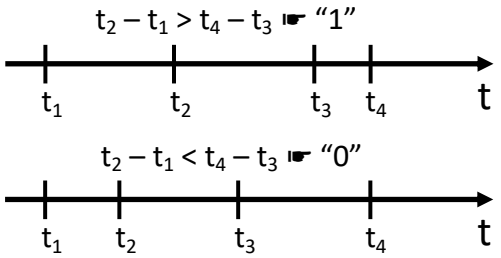

Fig. 1. *Method of generation of random bits by comparison of two-time intervals: t_1, t_2, t_3, t_4 – times of electron hits.*

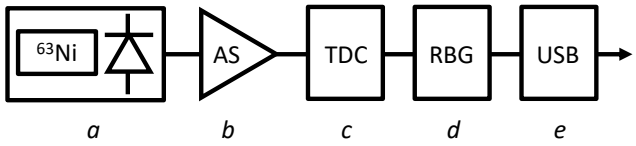

Fig. 2. *Block diagram of the TRNG: a – metal case with ^{63}Ni foil and electron detector, b – amplifier and shaper, c – time-to-digital converter, d – random bits' generator, e – USB interface*

B. ^{63}Ni as a source of randomness

The ^{63}Ni radioactive isotope is a perfect source of entropy. Its nuclear decay is the pure beta: $^{63}Ni_{28} \rightarrow {}^{63}Cu_{29} + e^- + \overline{\nu_e}$ - emitted are low-energy electrons and anti-neutrinos, neutrinos practically do not interact with anything. The maximum energy of electrons is 67 keV, and the mean energy is 17 keV. The half-life of ^{63}Ni is close to 100 years. Electron emission is random in time and space and is not affected by temperature, pressure, electric and magnetic fields, etc. The ^{63}Ni isotope is easily available in the form of thin foil.

C. Radiation safety

The maximum range of 70 keV electrons in copper is about 13 μm, hence no radiation can be detected outside the metal case with an electron detector and nickel. Moreover, 70 keV electrons do not penetrate the epidermis – the outer dead layer of human skin. Even if the tiny piece of nickel foil were swallowed – a highly unlike event – it would not make any harm. The

radiation dose absorbed would be about 0.75 mSv/year while the annual safe limit of intake is 0.5 Sv. The generator with a ^{63}Ni radioactive isotope is safe in manufacturing, use, and recycling.

D. Detectors of low-energy electrons

Two kinds of semiconductor diodes can be used for the detection of low-energy electrons: PIN diodes and SPAD diodes. Both types of diodes are well known as photon detectors but they can detect electrons as well. Both types of diodes have their advantages and drawbacks. PIN diodes as electron detectors are reverse-biased and produce current pulses when hit by electrons. SPAD diodes work in the so-called "Geiger regime": they are reverse biased by a voltage slightly higher than their avalanche breakdown voltage[17]. Electron hit initiates avalanche breakdown current. This current is much stronger that the current pulse in a PIN diode. As a result, SPAD diodes have much higher sensitivity [18]. Their drawback is that after the electron hit the avalanche current must be quenched by a special circuit. Moreover, these diodes tend to produce spontaneous breakdown events that must be distinguished from those produced by electron hits. The bias voltage value (typically about 20 V) is critical for reducing the spontaneous breakdowns to a minimum and at the same time maintaining high sensitivity to electrons.

E. Experimental generators

We tested both detectors. Two "proof of concept" generators have been developed. In the Institute of Microelectronics and Photonics (IMiF) in Warsaw, a PIN diode technology for our application has been developed. A batch of these diodes has been made and encapsulated in metal cases together with a tiny piece of ^{63}Ni foil, and complete generators (Fig. 3) have been built in IMiF. SPAD diodes with quenching circuits (delivered as complete modules by Xfab silicon foundry) were used by ChipCraft company. These were also developed into complete generators (Fig. 4).

Fig. 3. *TRNG with PIN diode (in red circle)*

Fig. 4. *TRNG with SPAD diode and quenching circuit module (in red oval)*

F. Testing and Practical Application

Streams of random bits uploaded to computers, up to 5,000,000 bits long, were heavily tested with NIST [19] and ENT [20] batteries of tests and passed them all positively, i.e., generated sequences of random bits were truly random. These streams were used in testing the encryption and decryption of text files of various lengths. To this end, two applications, *BARN Writer* and *BARN Reader*, have been developed to test the BARN process on real data.

IV. Examples and Discussion

An example of a short 10-bit message (number *711* in binary; not converted from ASCII as this would require 24 bits and in the following examples it would be very hard to guess some decoding keys, see discussion below), simple, quaternary *key table* $k = \{1, 3, 2, 1\}$, and 21 bits of random number stream (of which 19 are used only) is presented in Figures 5, 6, and 7 to numerically illustrate the BARN algorithm.

bit#	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	...
bits	0	0	0	1	1	0	1	0	0	0	0	0	1	0	1	1	1	0	1	0	0	...

Fig. 5. *Stream of random bits*

bit#	1	2	3	4	5	6	7	8	9	10
bits	1	1	1	0	0	0	1	1	0	1

Fig. 6. *10-bit message*

bit#	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
bits	**1**	0	0	**1**	1	**1**	**0**	**0**	0	0	**0**	0	**1**	**1**	**0**	1	1	**1**	1
shift	*1*			*3*		*2*	*1*	*1*			*3*		*2*	*1*	*1*			*3*	

Fig. 7. *Random stream of bits with an embedded message*

The example above shows that for keys limited to integers 1, 2, and 3 (mean 2), the *encrypted message* is about 2x longer than the *message*. This can be a useful encryption method for long messages e.g., for encryption of speech or video. From a secret key of 256 bits, one can obtain 128 numbers in the range between 1 and 3 with an average of 32 zeroes thus getting on average 96 entries for the *key table*. When all 7 integers in the *key table* are used (mean 4), on average the *encrypted message* file will be 4x longer than the *message* but it will be much harder to decrypt (many more combinations possible). For example, when 7 objects are combined in a sample of size 74 then the number of permutations with repetitions is $7^{74} > 2^{206}$. For the simplified model of the limited number of integers used in the example above, there are 3^{96} permutations with repetitions of 3 integers on 96 positions, which is a number larger than 2^{152}. The above considerations lead us indirectly to the so-called *knapsack problem*, which is known [4], [21], [22] to be NP-complete [23]. Big numbers of possible permutations suggest that the BARN method can be also NP-complete but we are not able to make rigorous proof yet. We did run the very same randomness tests

[19] and [20] on encrypted files that were also run on our RNG. They mostly show not-so-good randomness of the *encoded messages*. Some tests cannot be effectively run on short-bit streams (shorter than at least 1M bits). However, these results do not mean that statistical tests can help to decode the cipher. On the contrary, these tests only point out that the whole string of bits is statistically less random after encryption but they cannot pinpoint which particular bits contribute to this non-randomness.

The very first, brute force guess of a *key table k* that satisfies the BARN algorithm in the above-simplified example is {1, 1, 1, 1} – it converts all bits of the stream with an encrypted message into the number *503865*. Another example of a decryption with a guessed *key table k* of {2, 1, 2, 1} produces a result of *3340*, which is also plausible but not related to the original *message* at all.

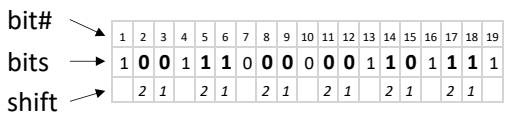

Fig. 8. *Decryption with a guessed key {2, 1, 2, 1}*

While decrypting longer streams of bits, the number of plausible solutions becomes enormous and thus very effectively helps to hide the true message that was encrypted.

V. Conclusions

We have shown that a very simple combination of steganography and cryptography with the use of a stream of true random numbers is producing a robust stream cipher method. We are planning to develop an encryption engine: an integrated circuit, which will combine both our TRNG and our BARN encryption method.

Acknowledgment

We are very grateful to Prof. R. Szczygiel for his insightful report on the sensitivity of detector diodes, to dr G. Janczyk, dr D. Szmigiel and their teams from IMiF, and dr K. Siwiec and dr J. Jasinski from ChipCraft for their expert work on our "proof of concept" true random number generators.

References

[1] Markey H.K. et al. US patent 2,292,387 *Secret communication system*, 1941

[2] Dworkin M., Barker E., Nechvatal J., Foti J., Bassham L., Roback E., and Dray J. *Advanced Encryption Standard (AES)*, Federal Inf. Process. Stds. (NIST FIPS), National Institute of Standards and Technology, Gaithersburg, MD, 2001 [online] https://doi.org/10.6028/NIST.FIPS.197 (accessed January 20, 2023)

[3] Mazurczyk W., Szczypiorski K. and Lubacz J. *Four ways to smuggle messages through internet services*, IEEE Spectrum, vol. 50, no. 11, pp. 42-45, 2013

[4] Diffie W. *The first ten years of Public-Key Cryptography*, Proc. IEEE, vol. 76, no. 5, pp. 560-577, 1988

[5] Hellman M.E. et al. US patent 4,200,770 *Cryptographic apparatus and* method, 1980

[6] Bernstein D.J. *Curve25519: New Diffie-Hellman Speed Records*, in Yung M., Dodis Y., Kiayias A., Malkin T. (eds) *Public Key Cryptography* - PKC 2006, Lecture Notes in Computer Science, vol. 3958, 2006 Springer, Berlin, Heidelberg

[7] [online] https://www.random.org/ (accessed February 22, 2023)

[8] [online] https://www.fourmilab.ch/hotbits/ (accessed February 22, 2023)

[9] [online] https://www.protegost.com/ (accessed February 22, 2023)

[10] [online] *https://comscire.com/* (accessed February 22, 2023)

[11] [online] https://www.quintessencelabs.com/ (accessed February 22, 2023)

[12] [online] *https://www.idquantique.com/random-number-generation/products/quantis-qrng-chip/* (accessed February 22, 2023)

[13] [online] *https://quside.com/q100-chipset* (accessed February 22, 2023)

[14] Tatarkiewicz J.J. et al. US patent 10,901,695 *Apparatus, systems, and methods for beta decay based true random number generator*, 2021

[15] Peter M. and Schindler W. *A Proposal for Functionality Classes for Random Number Generator*, BSI publication AIS 31 version 2.35, (2022) [online] https://www.bsi.bund.de/SharedDocs/Downloads/EN/BSI/Certification/Interpretations/AIS_31_Functionality_classes_for_random_number_generators_e.pdf (accessed January 20, 2023)

[16] Walker J. *How HotBits Works*, 1996[online] https://www.fourmilab.ch/hotbits/how3.html (accessed January 20, 2023)

[17] Dalla Betta G-F., Pancheri L., Henderson R. and Richardson J. (2011)Avalanche Photodiodes in Submicron CMOS Technologies for High-Sensitivity Imaging, in Advances in Photodiodes, ed. G-F. Dalla Betta, London: InTechOpen, pp. 225-48, ISBN 978-953-307-163-3, [online] https://cdn.intechopen.com/pdfs/14343/InTech-Avalanche photodiodes in submicron cmos technologies for high sensitivity imaging.pdf (accessed January 20, 2023)

[18] Szczygiel R. unpublished report, 2021

[19] Bassham L., Rukhin A., Soto J., Nechvatal J., Smid M., Leigh S., Levenson M., Vangel M., Heckert N. and Banks D. *A Statistical Test Suite for Random and Pseudorandom Number Generators for Cryptographic Applications*, Special Publication (NIST SP), National Institute of Standards and Technology, Gaithersburg, MD, 2010 [online] https://tsapps.nist.gov/publication/get_pdf.cfm?pub_id=906762 (accessed January 20, 2023)

[20] Walker J. *ENT - A Pseudorandom Number Sequence Test Program*, 2008 [online] https://www.fourmilab.ch/random/ (accessed January 20, 2023)

[21] Kellerer H., Pferschy U. and Pisinger D. *Knapsack problems*, Berlin, 2004, Springer ISBN 978-3-540-40286-2

[22] Martello S. and Toth P. *Knapsack problems: Algorithms and computer implementations*, 1990, Wiley-Interscience, ISBN 978-0-471-92420-3

[23] Aaronson S. *Quantum Computing Since Democritus*, 2013, Cambridge University Press, ISBN 978-0-521-19956-8