http://dx.doi.org/10.5755/j01.eie.25.4.23969 ELEKTRONIKA IR ELEKTROTECHNIKA, ISSN 1392-1215, VOL. 25, NO. 4, 2019

Secure and Efficient Hybrid Random Number Generator Based on Sponge Constructions for Cryptographic Applications

Selman Yakut, Taner Tuncer, Ahmet Bedri Ozer
Department of Computer Engineering, Engineering Faculty, Firat University, Elazig, Turkey
syakut@firat.edu.tr

***Abstract*—Random numbers constitute the most important part of many applications and have a vital importance in the security of these applications, especially in cryptography. Therefore, there is a need for secure random numbers to provide their security. This study is concerned with the development of a secure and efficient random number generator that is primarily intended for cryptographic applications. The generator consists of two subsystems. The first is algorithmic structure, Keccak, which is the latest standard for hash functions. The structure provides to generate secure random numbers. The second is additional input that generates with ring oscillators that are implemented on the FPGA. The additional inputs prevent reproduction and prediction of the subsequent random numbers. It is shown that the proposed generator is satisfies security requirements for cryptographic applications. In addition, NIST 800-22 test suite and autocorrelation test are used to demonstrate that generated random numbers have no statistical weaknesses and relationship among itself, respectively. Successful results from these tests show that generated numbers have no statistical weaknesses. Moreover, important advantage of the proposed generator is that it is more efficient than existing RNGs in the literature.**

***Index Terms*—Information security; Keccak algorithm; Random number generation; Ring oscillator.**

I. INTRODUCTION

Random numbers are used in many different areas, such as quantum mechanics, gambling games, statistics, and cryptography. These numbers are the most important parameters that affects the security of the whole applications, especially in cryptography [1], [2]. For example, any weakness in key values, initialization vectors or seed values threat application's security exactly.

A lot of random number generators are proposed in literature [1]. However, these generators are generally divided into three groups, such as True Random Number Generators (TRNGs), Pseudo Random Number Generators (PRNGs), and Hybrid Random Number Generators (HRNGs) [1].

The natural disorder and unpredictable physical systems can be utilized to generate true random numbers [3]. This disorder is called entropy, typical sources of which include temperature, jitter, mechanical systems, and a computer user's individual way of moving a mouse [4]–[7]. The general structure of these generators is shown in Fig. 1. The analogue signals from entropy sources are converted to digital data and sampled. In order to overcome statistical weaknesses or environmental problems, approaches, such as post-processing algorithms, are widely used [3].

Manuscript received 28 September, 2018; accepted 3 April, 2019.

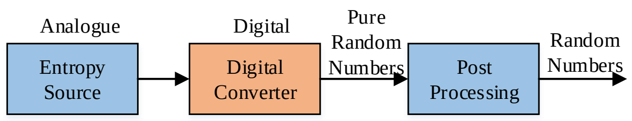


Fig. 1. TRNG general structure.

Pseudo random numbers are created using algorithmic structures and seed values. In PRNGs, algorithmic structure is known exactly and random numbers are generated according to the selected seed value. Figure 2 shows the general structure of a these generators. The output function is used to produce a random number and the state function is used to update the internal state value. If the same seed value is used in these generators, the same random number sequences are produced [4]. If seed values or intermediate values are known, it will be practically feasible to compute preceding random numbers.

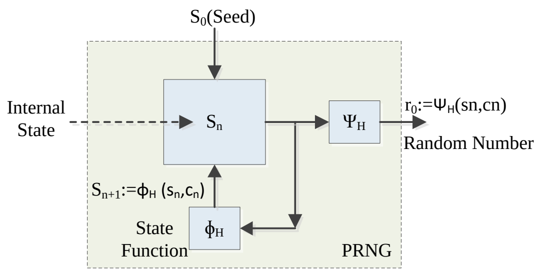


Fig. 2. PRNG general structure.

HRNGs consist of algorithmic structure and additional inputs. Encryption algorithms, hash functions, and equations can be used as the algorithmic structure. Additional input is composed of true random numbers, which are generated from entropy sources, such as chaotic systems or ring oscillators (ROs). Additional inputs to these generators prevent the prediction and regeneration of random number sequences and can be applied to one or both of the output and state functions shown in Fig. 2.

The security requirements given in Table I must be met in order to guarantee that the random number sequences provide appropriate levels of protection for cryptographic applications [1]. This ensures that these numbers have no statistical weaknesses. That is to say, it should be exclude replay attacks or correlation based attacks and passed from statistical tests. It also cannot be reproduced or predicted.

TABLE I. SECURITY REQUIREMENTS FOR RNGS IN CRYPTOGRAPHIC APPLICATIONS.

R1:	The random numbers should not show any statistical weaknesses
R2:	The knowledge of subsequences of random numbers shall not allow to compute predecessors or successors practically or to guess them with no negligibly larger probability than without knowledge of these subsequences
R3:	It shall not be practically feasible to compute preceding random numbers from the internal state or to guess them with no negligibly larger probability than without knowledge of the internal state
R4:	It shall not be practically feasible to compute future random numbers from the internal state or to guess them with no negligibly larger probability than without knowledge of the internal state

The generation of secure and efficient random numbers is an important consideration in many areas. This is particularly true in the field of cryptographic applications [2]. Random numbers used in these applications should not contain reproducible, unpredictable, and statistical weaknesses [1]. However, the use of a TRNG for this purpose introduces a number of problems, which include low speed, high cost, hardware dependency, and statistical weaknesses. While various approaches, such as post processing algorithms, are proposed to overcome these problems, these methods can lead to large data loss because they ignore some data, which includes statistical weaknesses. Furthermore, the PRNG contains periodic repetition and the output space cannot exceed the seed space. Generators that do not have an entropy source are not appropriate for cryptographic applications, so that they cannot meet the R4 security requirement [4]. A HRNG can be used to overcome these difficulties and to take the advantage of these generators [3]. These generators consist of algorithmic method and additional input. Additional inputs are also taken from TRNGs, which prevent random numbers from being reproduced and guessed. The use of strong cryptographic constructs as an algorithmic method, therefore, guarantees the security of the generator. However, many of the proposed HRNGs cause data loss and work with low efficiency.

Numerous RNGs intended for cryptographic applications have been documented [1]. These methods use physical phenomena, including electrical noise, and chaotic systems as the entropy source [1]–[4]. Random numbers from these sources are then used in conjunction with strong algorithms to provide the security [1]. These generators use different algorithms, especially cryptographic constructs, such as encryption or hashing algorithms, to guarantee security [8], [9]. Avaroğlu, *et al*. proposes a hybrid structure using the Advanced Encryption Standard (AES) algorithm operating in conjunction with a chaotic additional input, and this work proves to be satisfactory for cryptographic applications [9]. Thamrin, *et al*. uses an optical mechanism and linear feedback recording to generate hybrid random numbers, which satisfy the requirements of the appropriate statistical tests [10]. Grøstl generate secure random numbers using the hash function and additional chaotic inputs. Numbers produced with this generator are found to meet the statistical tests and security requirements successfully [11]. Avaroğlu, *et al*. suggests the use of a RO and chaotic functions to generate random numbers using a hybrid RNG, and the resulting output from this generator is again successfully verified [12]. Yuan, *et al*. proposes a structure consisting of digital-analogue constructs, using electronic elements with chaotic structures, and the data produced by this work are once more validated [13]. Avaroğlu proposes a generator based on chaotic structures and shows that the statistical requirements are satisfied [14]. Łoza, *et al*. generate random numbers using the RO and Sha-256 algorithms and proves that they are robust [15]. Magfirawaty and his colleagues generates random numbers using chaotic structures and the hash function, and, once more, these are statistically verified [16]. Wang and Li generate random numbers using an approach based on RO and XOR operations, and demonstrates successful results from statistical tests [17]. Liu, *et al*. propose RO-based approach to generate true random numbers and show analysis of these numbers. Buchovecká, *et al*. have produce and analyse real random numbers with a RO-based approach [18], [19]. Chen, *et. al*. produce and analyse these numbers based on hash functions [20]. Kote, *et al*. propose TRNG that is based on the electronic circuit. In the study, the proposed generator is successful according to the analyses results of produced number [21]. Wieczorek and Gołofit propose TRNG based on two stages of randomness. The first stage is based on a chaotic circuit and the second stage is based on metastability of a flip-flop that is stimulated by chaotic initial conditions. The proposed generator shows successful results in the randomness tests [22]. Wieczorek presents a generator that utilizes both the ring oscillations and metastability phenomena. The proposed generator is successful in the randomness tests [23]. Wieczorek proposes and analyses TRNG based on nearly-metastable operation of groups of FPGA flip-flops and an adaptive feedback loop. Numbers that produce from the generator are successful in the randomness tests [24]. Wieczorek and Gołofit present a dual-metastability time-competitive generator. The generator is analysed both numerically and theoretically and the statistical test results of the presented generator are successful [25]. Wieczorek presented a dual-metastability time-competitive generator. Empirical and statistical test results of random number that take from the generator are successful [26].

During this study, a secure and efficient HRNG is developed for cryptographic applications. In this generator, Keccak hash algorithm is used as an algorithmic method and RO is used in generating additional input. In order to make the developed generator more efficient, Keccak algorithm is rearranged and used. The structure of Keccak algorithm is reorganized, so that for each operation of the algorithm, a pure true number of 512 bits is taken and 1600-bit output data is generated after 24 rounds of processing by the kernel function. From this data, 1024-bit true random numbers and 1088-bit output data for the next iteration are produced. Five ROs, each with 3 inverter gates, are used to generate additional inputs, and these are designed in an FPGA environment. Thus, problems arising from the fact that these numbers are produced outside are eliminated as well as real random numbers that are secure for additional input. Random numbers generated using this method satisfy the security requirements for cryptographic applications. The NIST 800-22 test suite and the autocorrelation test show that this procedure does not introduce any statistical weaknesses. The most important advantage of the proposed generator is that the pure real random number is not lost and the generator is 2 times more efficient because of difficulties and high cost to generate true random numbers to make data loss more important.

The work undertaken by this study can be summarized as follows:

1. A model based on Keccak, the latest standard of abstract algorithms, is used;
2. Weaknesses arising from the generator or environmental factors are eliminated without data loss and without the use of any post-processing algorithm;
3. In order to make these numbers more secure, additional inputs are generated on a circuit that were designed on the FPGA;
4. The resulting RNG is found to meet the security requirements for cryptographic use;
5. Successful results are obtained from statistical tests on the output from the generator.
6. The proposed generator takes 512 bits of raw truerandom numbers as an additional input and generates a true random number of 1024 bits.

This paper is organized as follows. In Section II, the structure of the proposed generator is considered, both the structure of the hash algorithm, which is an algorithmic part of the generator, and a true random number generator that forms additional inputs are examined. In Section III, security analysis and statistical analyses of the proposed generator are carried out. In Section IV, the results of the developed generator are discussed.

II. The Proposed Hybrid Generator

During this study, a secure and efficient HRNG is designed for cryptographic applications. This generator consists of two basic subsystems. The first of these produced are pure true random numbers using a RO. The second subsystem uses an algorithmic function that allowes secure random numbers to be generated from the pure true random numbers. Keccak hash algorithm is used for this purpose.

The general structure of the proposed RNG is given in Fig. 3. In the generator 512 bits of pure true random number data is used as an additional input. In the first operation of this generator, the additional input (S0) is given to Keccak function, while the latter is given to the additional input state function. In the first operation of the generator, 512 bits of pure TRNG data is taken as an input by the algorithmic section and subjected to algorithmic operations, and a 1600-bit hash value is generated. In these algorithmic operations, a 1600-bit input vector is taken and a 1600-bit output vector is generated after executing 24 rounds of its operations. The generated 1600-bit hash data forms the input data of the output function. Using the output function, 1024-bit real random number and 1088-bit intermediate input data are generated from this hash data. This new input vector of 1600-bit is then generated for the algorithmic partition using 1088-bit interleaved input data received by the transition function and the 512-bit pure TRNG data forms the additional input. In subsequent runs of the generator, this 1600-bit input vector is processed by the algorithm and then output to the function. Thus, in each run of the developed generator, 1024-bit safe real random numbers are generated.

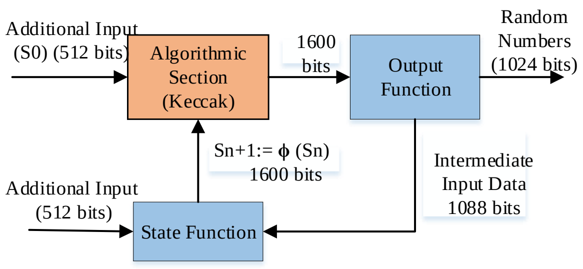


Fig. 3. General structure of the generator.

The output function in the proposed generator is arranged as shown in Fig. 4. In the figure, relation of the processed and generated data, which is operated by the generator, is shown. The 1600-bit output vector received by this function is divided into three parts: Rn1, Rn2, and Cn. Rn1is a data block taken from TRNG, Rn2 is a data block taken from previous round, and Cn is a security parameter for Keccak. n-bit TRNG data is received and 2n-bit TRNG is produced thanks to this structure. In each run of the output function, random numbers are generated using pieces of 512-bit Rn1 and Rn2, each. Using 512 bits of Rn2 and 576 bits of Cn, 1088 bits of intermediate input data are generated.

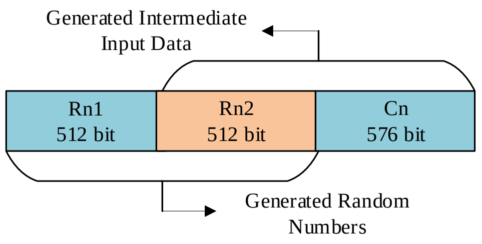


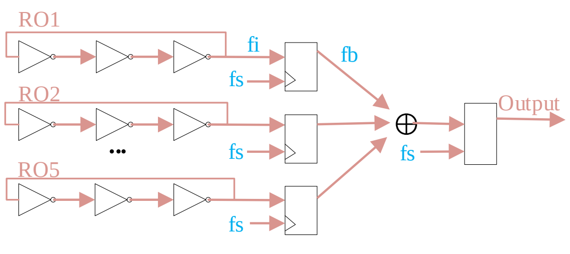
Fig. 4. Data processed by the generator at each step.

The additional inputs, which constitute an important part of the proposed generator, are produced using a RO-based approach, in which 5 ROs, each containing 3 inverter gates, are used. This technique is implemented in FPGA to produce pure true random numbers, which are used to create additional inputs. Details of this pure data generator are given in Section IIA. In Section IIB, Keccak algorithm, which constitutes the algorithmic part of the proposed generator, is given.

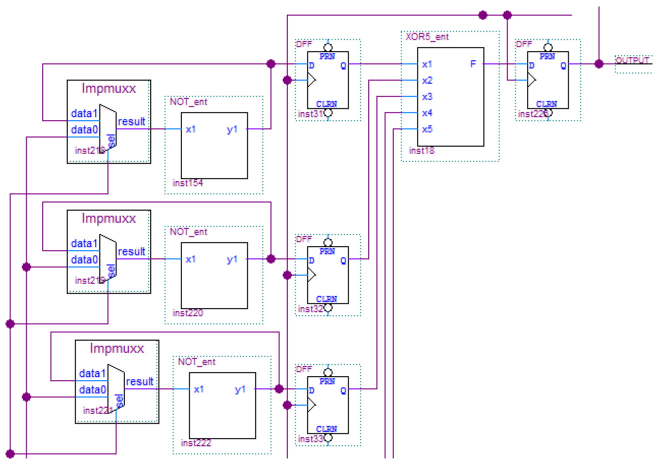
A. The Method of Obtaining the Additional Input

The irregularities and unpredictability of entropy sources are used to produce true random numbers. Although there are a lot of entropy sources, electrical noise is commonly used because of providing more reliable methods [4]. Random numbers generated using ROs have a widespread use and this technique can be achieved using odd number of NOT gates. Each of these gates has a different delay, which is expressed as jitter, and these delays are used as a source of entropy. This approach utilizes multiple ring oscillators in combination with the XOR gate followed by sampling with a flip-flop. It is desirable that the numbers of ring oscillators and inverter gates are minimized in order to consume much less power. Numerous methods based on this approach are proposed, which can be used in conjunction with strong cryptographic constructs to design safe RNGs [1], [12]. The proposed generator needs true random numbers in order to provide additional inputs. For this purpose, a true RNG with a RO base is implemented. To achieve low power consumption, the number of ROs and the number of inverters per RO are 5 and 3, respectively. Figure 5 provides details of the RO design used.

Each RO generated logic values are of 0 at the beginning of the process, because the data0 input is logic 1. When an external signal (logic 1) is applied from the physical medium to the selection input of the multiplexer (mux), the input value of data1 is observed at the output of the mux. Thus, non-periodic and random 1 and 0 values are continuously observed at the output of the odd-numbered inverter. Each RO output is sampled with D-type flip-flops and subjected to XOR processing. Random numbers are generated by resampling the XOR output as shown in Fig. 5(b).



(a)

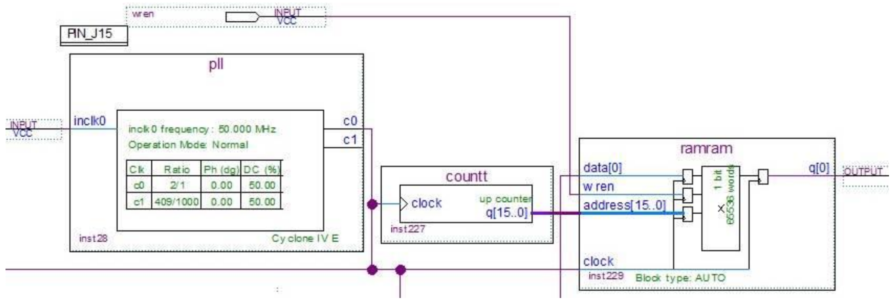


(b)

Fig. 5. Number generator using 3 inverters and 5 ROs (a); RO Design realized in altera's quartus software (b).

A backup unit is designed to use the numbers generated by this structure as additional inputs. Figure 6 shows how these numbers are recorded. The phase-locked loop equipment is used for synchronous operation of the RO and memory units and has a frequency of c0 = 50 MHz. A counter is used to write each generated number to memory addresses and specify the address of the memory handler. The Wren entry is set so that the generated number could be written to memory. Quartus software allows the numbers stored in memory elements to be written to a file in text format, so that they could be easily accessible.

Figure 7 contains a sample of the numbers obtained from the system in real time, which are recorded in the memory unit.



inclk0 frequency: 50.000 MHz
Operation Mode: Normal

Clk	Ratio	Ph (dg)	DC (%)
c0	2/1	0.00	50.00
c1	409/1000	0.00	50.00

Fig. 5. Memory design for recording numbers.

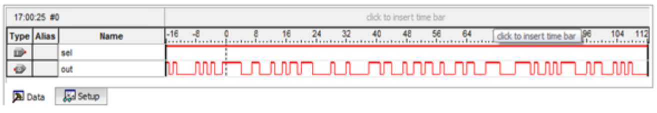


Fig. 6. Numbers generated in real time and recorded in the memory unit.

An oscilloscope image for a fraction of the actual random numbers actually obtained from the designed system is given in Fig. 8.

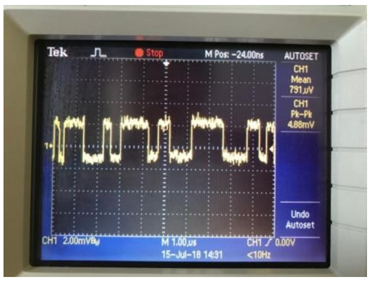


Fig. 7. Oscilloscope pattern of true random numbers actually generated.

B. Algorithmic Part of the Proposed Generator: Keccak

The hash algorithms receive an input message of any length and generate an output value, or hash of a certain length that is specific to that message. These algorithms form the most important part of many cryptographic applications, such as digital signatures, key generation, and pseudorandom number generation.

Security properties of hash algorithms are critical for the security of the application for which they are used. The most important security consideration for these algorithms is to ensure that they are preimage resistant. In other words, it should not be possible to construct the original message from the hash value. Another important factor is that the same output value should not be generated for two different messages. This is expressed as collision resistance.

Keccak that is the latest standard for hash algorithms can take an input sequence of any length and generate hash that is various lengths since it is based on sponge constructions [27], [28]. The general structure of Keccak is given in Fig. 9. The input and intermediate vectors of Keccak consist of two parameters *r* and *c*, where, *r* indicates the processed message size per round, *c* is a parameter used for security, *f* is the kernel function, *n* is the number of blocks, *bi* is the processed block size, and *hi* is a produced hash value per round.

The algorithm is divided into two parts during this process. The first of these is dedicated to receiving the input message blocks, while the second is responsible for generating the hash values.

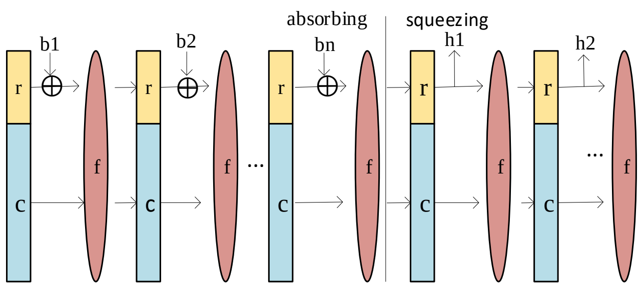


Fig. 9. General structure of Keccak algorithm.

The first part of the algorithm consists of 3 steps:

1. The initial vector is firstly set to zero. The input message is then divided into blocks of equal length and can be added to the last block if necessary. 10 ... 1 bit arrays are inserted instead of the missing bits;
2. Starting with the first block, all blocks are then XOR’ed with the first *r* bits of the input vector. A new *r*-bit fragment of the resulting input vector is generated from this process;
3. The input vector is modified by the function *f*. After all input blocks are processed, the first part of the function is completed.

The second part of the function is hash producing. Upon completion of the first part of Keccak algorithm, the first *r* bit of the output vector is taken and, if necessary, the output vector is again processed using the *f* function. This process continues until the hash value is generated at the desired length.

One of the most significant features of these structures is that they are flexible, allowing the size of the message block, the number of operations, and the size of the input and intermediate vectors to change. The vector expresses the limited range defined by the parameter $v = 5 \times 5 \times w$ and $w = 2l$, $l = 1, 2, \dots 6$. Therefore, the size of the input and intermediate vectors can take on 25, 50, ...,1600 bits. These vectors are converted to a three-dimensional array, *A*, before being processed by the kernel function. The size of *A* is taken as $A = 5 \times 5 \times w$. The algorithm performs subsequent operations on this array.

The number of cycles of the kernel function changes according to the vector size in Keccak hash algorithm. This change is expressed as $12 + 2 \times l$. For example, when the 1600-bit input vector is considered, the value of l becomes 6 and the expression of $12 + 2 \times l$ provides 24 rounds. For security reasons, it is desirable that the vector size should be large [23].

The most basic part of Keccak is the *f* function, which forms the crucial part of the sponge construction. The function comprises 5 basic steps, which are being composed of operations, such as AND, OR, and XOR. The steps are represented by the symbols θ, ρ, π, χ, and ι. They are performed in this order during each round [28]. At each operation step, *A* is taken as an input and *A'* is generated after the operations are completed. In the next round, these operations are repeated [28].

III. ANALYSIS OF THE PROPOSED GENERATOR

The random numbers used in cryptographic applications have a vital importance on the security of the whole application. Therefore, the security of random numbers used in these applications should be assured. For the security of random number generators, they must not contain statistical weaknesses and the security requirements given for these generators must be met. Furthermore, the security of the algorithmic method and the additional inputs used in the generator must be assured.

In the proposed generator, the algorithmic part is created by Keccak, which is the last standard for cryptographic hash algorithms, because it is faster and more efficient than other candidate algorithms [29]. The security of this algorithm guarantees the security of the generator since it constitutes the main structure of the generator. The use of 24 bit and 1600-bit vectors prevents the possibility of attacks, which aim to decrease the number of rounds or input vectors. Possible weaknesses that may exist in this algorithm have already been determined during the security verification of the generator and no further analysis is considered necessary in this respect.

RNGs used in cryptographic applications must meet specific security requirements in order to be considered secure for use. These requirements, R1 to R4, were mentioned earlier in this paper. Strong cryptographic constructs, such as Keccak, satisfy the security requirements of R1 to R3. Moreover, when the security parameters of hash algorithms are taken into consideration, they are ideal structures for use as algorithmic methods. Using these constructions, secure and efficient random number generators can be designed. The use of this algorithm ensures it is not possible to calculate the preceding and subsequent numerical values, even if intermediate values are obtained. The use of pure true random numbers as an additional input also enables the security requirements of R4 to be satisfied, and it is therefore considered impossible to determine the random numbers, which are generated.

Potential weaknesses that might result from environmental factors are prevented since additional inputs are generated by RO that implement on FPGA. Moreover, the power consumption is reduced by minimizing the number of RO and inverter gates used. Successful results are obtained despite the fact that the additional inputs are used without any post-processing.

One of the most important security considerations for RNGs is that they do not contain any statistical weaknesses, which make them susceptible to various attacks. Therefore, it should be assured that the numbers used in cryptographic applications do not contain statistical weaknesses. A lot of tests are used to statistically analyse the output from RNGs and these contain similar procedures and features. The test suite recommended by NIST and autocorrelation method are most commonly used and accepted.

The NIST test suite comprises 14 different procedures and examines the generator and resulting random number sequence for statistical weaknesses [30]. Procedures used in this determination phase are described in the literature [30]. In Table II, the test results for the pure real random numbers and the numbers generated by the proposed method are given. Successful results are obtained when these tests have are conducted on the generator developed during this study (Table III).

TABLE II. STATISTICAL TEST RESULTS OF THE PROPOSED HRNG AND THE RESULTS OF WITHOUT THE GENERATOR.

	Raw True Random Numbers		**Numbers of Proposed Generator**	
Tests	**P-Value**	**Result**	**P-Value**	**Result**
Frequency Test	-	Unsuccess	0.4968	Success
Block-frequency	-	Unsuccess	0.5023	Success
Runs	-	Unsuccess	0.6234	Success
Test for the Longest Run of Ones in a Block	-	Unsuccess	0.2867	Success
Binary Matrix Rank	0.4247	Success	0.6403	Success
Discrete Fourier Transform	-	Unsuccess	0.0182	Success
Non-overlapping Template Matching	-	Unsuccess	0.7173	Success
Overlapping Template Matching	0.0215	Success	0.6038	Success
Maurer's Universal Statistical	-	Unsuccess	0.7332	Success
Linear Complexity	0.6155	Success	0.6051	Success
Serial Test1	-	Unsuccess	0.2557	Success
Serial Test2	0.8790	Success	0.4979	Success
Approximate Entropy	-	Unsuccess	0.1605	Success
Cumulative Sums	-	Unsuccess	0.5487	Success

These tests are used to show that the random number sequences generated by autocorrelation are self-contained. The corresponding mathematical description is given below

$$A(d) = \sum_{i=0}^{n-d-1} (b_n \oplus b_{(n+d)}), \tag{1}$$

where $\oplus$ denotes the XOR operation, n denotes size, and d denotes the integer value in the range $0 \leq d \leq n / 2$. The relationship between zero and one is given as follows

$$X_5 = \frac{2A(d) - (n-d)/2}{\sqrt{(n-d)}} \tag{2}$$

The fact that the test results from this study were in the range | X$_5$ | <1.6449 shows that the required criteria are satisfied. Successful results of the autocorrelation test for the proposed generator are given in Table III.

The generator designed during this study provides an important advantage in that it increases efficiency because it takes as input n bits data and produce 2 × *n*-bits true random numbers. Most existing methods incur data loss due to the fact that they take as input *n*-bits data and produce less than *n*-bits true random numbers, and this is an important consideration when taking into account the difficulty and cost of producing pure random numbers. The method developed during this study avoids this while ensuring that the efficiency is doubled in comparison to other techniques. The efficiency of this generator is seen to be the most important advantage when compared to other methods or the post processing. In Table IV, similar random number generators suggested in the literature, used algorithmic method and additional inputs, used statistical tests and performance are given. In the last column, the efficiency of these methods was examined. In the Table IV, it is seen that the proposed generator is much more efficient than the other generators.

TABLE III. AUTOCORRELATION TEST RESULTS.

D value	**X5 value**	**Result**
8	-0.3440	Success
10	-0.1250	Success
13	0.1172	Success
20	0.9761	Success
25	0.8591	Success
100	-0.3441	Success
500	-0.7227	Success
1000	0.5875	Success

TABLE IV. SUMMARY OF PROPOSED NUMBER GENERATORS IN THE LITERATURE.

Similar Studies	**Additional Input**	**Algorithmic Method**	**Statistical Tests**	**Produ-ctivity, %**
[9]	Chaotic Input	AES	NIST 800-22	100
[10]	Optical	LFSR	NIST 800-22	100
[11]	Chaotic Input	Grøstl	NIST 800-22	100
[12]	RO	Chaotic Functions	NIST 800-22	100
[13]	Electronic Components	Chaotic Functions	NIST 800-22	100
[14]	Chaotic Input	Chaotic Functions	Autocorrelation NIST 800-22	50
[15]	RO	Sha-256	NIST 800-22	50
[16]	Chaotic Input	Sha-256	Autocorrelation NIST 800-22	50
[17]	RO	XOR Process	NIST 800-22	50
[18]	RO	XOR Process	NIST 800-22	50
[19]	RO	-	NIST 800-22	25
[20]	RO	Sha-256	NIST 800-22	50
[21]	Electronic Components	-	NIST 800-22	100
[22]	Chaotic Input	Electronic Component	NIST 800-22 Diehard	100
[23]	RO Metastability	-	NIST 800-22 Diehard	100
[24]	Electronic Component	Adaptive Feedback Loop	NIST 800-22 Diehard	100
[25]	Electronic Component Metastability	-	NIST 800-22 Diehard	100
[26]	Electronic Component	-	NIST 800-22 Diehard	100
Proposed Method	RO	Keccak	Autocorrelation NIST 800-22	200

IV. CONCLUSIONS

Random numbers are used in many areas, such as game theory, statistics, and cryptography. They are vital in influencing the security of these applications. In cryptography, secure random numbers must be used in order to guarantee security. During this study, a HRNG is designed, which is secure and efficient and can be used in many areas, especially cryptography. The generator consists of two parts: the algorithmic method and the additional input. Keccak hash algorithm, which is the latest standard, is rearranged and additional inputs were obtained through a RO-based approach. Additional input prevents the reproduction and prediction of random numbers. Each run of the generator is given an additional 512-bit pure true random number input and returns a 1024-bit true random number. Proposed generator meets the security requirements for cryptographic applications. The successful results from the NIST and autocorrelation tests show that it does not contain any statistical weaknesses. It also provides significant advantages in that it avoids data loss, which is a major concern when considering the difficulties and cost of random number production and is twice as efficient as existing RNGs in the literature.

REFERENCES

[1] Ç. Koç, *Cryptographic Engineering*. Springer, New York, 2009. DOI: 10.1007/978-0-387-71817-0.

[2] K. Wold, “Security properties of a class of true random number generators in programmable logic”, PhD thesis (philosophy), Gjøvik University College in Information Security, 2011.

[3] A. J. Menezes, P.C. van Oorschot, S. A. Vanstone, *Handbook of Applied Cryptography*. 1st edn. CRC Press, Boca Raton 1996.

[4] E. Avaroğlu “Hardware Based Realization Of Random Number Generator”, Phd Thesis, Electrical and Electronics Engineering Fırat, University, 2014.

[5] B. Sunar, W. J. Martin, and D. R. Stinson, “A provably secure true random number generator with built-in tolerance to active attacks, *IEEE Transactions on Computers*, vol. 58, no. 1, pp. 109–119, 2007. DOI: 10.1109/TC.2007.250627.

[6] İ. Koyuncu, A. T. Ozcerit, and İ. Pehlivan, “Implementation of FPGA-based real time novel chaotic oscillator”, *Nonlinear Dynamics*, vol. 77, no. 1–2, pp. 49–59, 2014. DOI: 10.1007/s11071-014-1272-x.

[7] S. Robson, B. Leung, and G. Gong, “Truly random number generator based on a ring oscillator utilizing last passage time, *IEEE Transactıons On Cırcuıts And Systems*, vol. 61, no. 12, pp. 937–941, 2014. DOI: 10.1109/TCSII.2014.2362715.

[8] R.N Akram, “Pseudorandom number generation in smart cards an implementation performance and randomness analysis”, in *Proc. 5th International Conference on New Technologies, Mobility and Security (NTMS),* pp. 1–7, 2012. DOI: 10.1109/NTMS.2012.6208760.

[9] E. Avaroğlu, İ. Koyuncu, A. B. Özer, and M. Türk, “A Hybrid pseudo-random number generator for cryptographic systems”, *Nonlinear Dynamics*, vol. 82, pp. 239–248, 2015. DOI: 10.1007/s11071-015-2152-8.

[10] N. M. Thamrin, G. Witjaksono, A. Nuruddin, and M. S. Abdullah, “An Enhanced Hardware-based Hybrid Random Number Generator for Cryptosystem”, in *Proc. 2009 International Conference on Information Management and Engineering,* 2009. DOI: 10.1109/ICIME.2009.115.

[11] F. Özkaynak, “Cryptographically secure random number generator with chaotic additional input”, *Nonlinear Dynamics,* vol. 78, pp. 2015–2020, 2014. DOI: 10.1007/s11071-014-1591-y.

[12] E. Avaroğlu,T. Tuncer, A. B. Özer, and M. Türk, “A new method for hybrid pseudo random number generator”, *J Micro-electron Electron Compon Mater*, vol. 44, pp. 303–311, 2014.

[13] Y. Zeshi, L. Hongtao, M. Yunchi, H. Wen, and Z. Xiaohua, “Digital-analog hybrid scheme and its application to chaotic random number generators”, *International Journal of Bifurcation and Chaos*, vol. 27, no. 14, 2017. DOI: 10.1142/S0218127417502108.

[14] E. Avaroğlu, “Pseudorandom number generator based on Arnold cat map and statistical analysis”, *Turkish Journal of Electrical Engineering & Computer Sciences*, vol. 25, no. 1, pp. 633–643, 2017. DOI: 10.3906/elk-1507-253.

[15] S. Łoza, L. Matuszewski, and M. Jessa, “A random number generator using ring oscillators and SHA-256 as post-processing”, *International Journal of Electronics and Telecommunications*, vol. 61, No. 2, pp. 199–204, 2015. DOI: 10.1109/ICSES.2014.6948739.

[16] M. Magfirawaty, M. T. Suryadi, and K. Ramli, “Performance analysis of zigzag map and hash function to generate random number”, in *Proc. 2017 International Conference on Electrical Engineering and Informatics (ICELTICs),* 2017. DOI: 10.1109/ICELTICS.2017.8253286.

[17] Y. Wang and S. Li, “A high-speed digital true random number generator based on cross ring oscillator”, *IEICE Trans. Fundam. Electron. Commun. Comput. Sci.*, vol. E99.A, no. 4, pp. 806–818, 2016. DOI: 10.1587/transfun.E99.A.806.

[18] Y. Liu, R. C. C. Cheung, and H. Wong, “A bias-bounded digital true random number generator architecture”, *IEEE Trans. Circuits Syst. I Reg.* Papers, vol. 64, no. 1, pp. 133–144, 2017. DOI: 10.1109/TCSI.2016.2606353.

[19] S. Buchovecká, R. Lórencz, F. Kodýtek, and J. Buček, “True random number generator based on ring oscillator PUF circuit”, *Microprocessors and Microsystems*, vol. 53, pp. 33–41, 2017. DOI: 10.1016/j.micpro.2017.06.021.

[20] S. Chen, B. Li, and C. Zhou, “FPGA implementation of SRAM PUFs based cryptographically secure pseudo-random number generator”, *Microprocessors and Microsystems*, vol. 59, pp. 57–68, 2018. DOI: 10.1016/j.micpro.2018.02.001.

[21] V. Kote, *et al*., “A True random number generator with time multiplexed sources of randomness”, *Radioengineering Journal,* vol. 27, no. 3, 2018, DOI: 10.13164/re.2018.0796.

[22] P. Z. Wieczorek and K. Gołofit, “True random number generator based on flip-flop resolve time instability boosted by random chaotic source”, *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 65, no. 7, 2018, pp. 1279–1292. DOI: 10.1109/TCSI.2017.2751144.

[23] P. Z. Wieczorek, “Lightweight TRNG based on multiphase timing of bistables, *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 63, no. 7, 2016, pp. 1043–1054. DOI: 10.1109/TCSI.2016.2555248.

[24] P. Z. Wieczorek, “An FPGA implementation of the resolve time-based true random number generator with quality control”, *IEEE Transactions on Circuits and Systems I: Regular Papers,* vol. 61, no. 12, pp. 3450–3459, 2014. DOI: 10.1109/TCSI.2014.2338615.

[25] P. Z. Wieczorek and K. Gołofit, “Dual-metastability time-competitive true random number generator, *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 61, no. 1, pp. 134–145. DOI: 10.1109/TCSI.2013.2265952.

[26] P. Z. Wieczorek, “Dual-metastability FPGA-based true random number generator”, *Electronics Letters*, vol. 49, no. 12, pp. 744–745, 2013. DOI: 10.1049/el.2012.4126.

[27] *Federal Information Processing Standards Publication 202 SHA-3 Standard: Permutation-Based Hash and Extendable-Output Functions*, 2015.

[28] G. Bertoni, J. Daemen, M. Peeters, and G. Van Assche, “Keccak sponge function family main document”, 2019 - http://keccak. noekeon.org/Keccak-main-2.1.pdf.

[29] X. Guo, S. Huang, L. Nazhandali, and P. Schaumont, “Fair and comprehensive performance evaluation of 14 second round SHA-3 ASIC implementations”, *NIST 2nd SHA-3 Candidate Conference,* 2011.

[30] A. Rukhin, *et al*., “A statistical test suite for random and pseudorandom number generators for cryptographic applications”, *NIST Special Publication 800–22rev1a*, Gaithersburg, MD, USA 2010.