Nonlinear Dyn (2015) 82:239–248
DOI 10.1007/s11071-015-2152-8

ORIGINAL PAPER

Hybrid pseudo-random number generator for cryptographic systems

Erdinç Avaroğlu · İsmail Koyuncu · A. Bedri Özer · Mustafa Türk

Received: 12 January 2015 / Accepted: 8 May 2015 / Published online: 22 May 2015

Abstract For a powerful cryptographic system, high-quality random number streams are essential. Those raw pseudo-random number generators (PRNG) that are used to generate high-quality random numbers have some disadvantages, such as failure to meet the R4 security requirement. Therefore, use of random number sequences generated by these generators in a cryptographic system puts the entire system at risk. This study proposes a new hybrid PRNG by means of an additional input introduced to transition and output functions used in a raw PRNG system in order to eliminate this risk. The additional inputs to the designed system have been implemented via the true random number generator developed by using the Sprott 94 G chaotic system on FPGA. The random number streams obtained from the recommended hybrid structure have been subjected to the NIST 800.22 and FIPS statistical test, which have given good results. According to these results, it has been proved that the recommended hybrid PRNG system meets the R4 security requirement and can be used in cryptographic applications.

Keywords Random number generator · Additional input · Sprott 94 G chaotic attractor

1 Introduction

Random number sequences have an important impact on robustness of cryptographic primitives that ensure that no attacker can predict or regenerate confidential data. Thus, random number sequences are crucial for powerful cryptographic applications [1].

Random number sequences generated must have good statistical characteristics in addition to these security requirements such as unpredictability and non-regenerability. As a result, powerful cryptographic systems require good random number sequences [2].

In the literature, various random number generators have been developed in order to obtain random number sequences required for powerful cryptographic systems. In general, these random number generators are divided into groups, namely true random number generators (TRNGs) and pseudo-random number generators (PRNGs) [3].

TRNGs use non-deterministic entropy sources to generate random numbers. Signals from an entropy

E. Avaroğlu (✉)
Department of Information Technology, İnönü University, Malatya, Turkey
e-mail: erdinc.avaroglu@inonu.edu.tr

İ. Koyuncu
Control and Automation Technology, Düzce University, Uzunmustafa, 81010 Düzce, Turkey

A. B. Özer
Computer Engineering Department, Fırat University, Elazig, Turkey

M. Türk
Electrical and Electronics Engineering Department, Fırat University, Elazig, Turkey

source are digitalized through sampling. Number sequences generated after sampling are subjected to post processing in order to increase randomness [3]. TRNGs have some drawbacks such as slowness, high costs and hardware dependency. However, TRNGs are used in cryptology as they are unpredictable and non-regenerable and have good statistical characteristics, as required [4–6]. Due to these properties, TRNG is introduced as an additional input to hybrid PRNGs [3].

In the literature, jitter [7], metastable [8] and chaotic systems [9–11] are used as entropy sources in a TRNG. Chaotic systems are specifically preferred in random number generators as they are sensitively dependent on initial conditions, exhibit unpredictable behaviors in the long term, and have aperiodic characteristics [12]. The literature includes various TRNG studies conducted using chaotic signs. These studies include TRNG based on chaotic cryptography and mouse movements [5], Piece-wise Affine Markov (PWAM) chaotic maps [6], oscillatory sampling method [9], a performance metric for discrete-time chaos-based truly random number generator [10], and chaotic structure with double spirals [11].

A raw PRNG generates long random number streams once an initial value called *seed* is provided as input to a deterministic algorithm. The fact that raw PRNGs are cheap, easily implemented, and fast and that they do not need any hardware can be considered as the advantages of raw PRNGs. However, number sequences generated by raw PRNGs can be predicted if the seed value is determined or the functions used in the system are not complex enough [3,13]. Therefore, any random number generator used to generate random numbers must meet four security requirements [3].

- R1: random numbers must be free from any statistical weakness.
- R2: knowing some sub-sequences of random numbers must not enable (an attacker) to estimate or predict predecessors or successors.
- R3: successor random numbers cannot be calculated if the internal state value is known or if it is possible to predict the internal state value even when it is not known.
- R4: following random numbers cannot be calculated if the internal state value is known or if it is possible to predict the internal state value even when it is not known.

In the literature, there are raw PRNGs designed by using encryption methods such as triple data encryption standard (3-DES) [14] and advanced encryption standard (AES) [14,15]. However, these designs cannot meet the R4 security requirement among those explained above, which causes a security gap in cryptographic systems [3]. Due to all these reasons, raw PRNGs are not appropriate for cryptographic applications [16,17]. Yet, this security gap can be eliminated by receiving regular additional inputs from a powerful TRNG [3].

In the study that is presented to ensure that a raw PRNG designed by using encryption methods meets the R4 requirement so that it can be used in cryptographic systems, an additional input has been added to transition and output functions of the raw PRNG [3,18,19]. An additional input improves the security as it makes the system unpredictable and random. For this purpose, the study uses AES, an encryption system used in cryptography, as the raw PRNG. In order to make sure that AES meets the R4 requirement, a new hybrid PRNG system has been developed by introducing, as additional inputs, the random bits generated by the TRNG implemented through use of a Sprott 94 G chaotic attractor. In addition to the R4 requirement, with the aim to avoid any distinguishing attacks [20], the value of key k_0 has been also generated by the TRNG unit. The randomness of the random bit sequences generated by the proposed hybrid PRNG system has been subjected to the NIST and FIPS statistical tests, and the results are presented.

This article is organized as follows: Sect. 2 describes the PRNG and additional inputs. Section 3 describes the structure of the AES, which is used as the raw PRNG, and Sect. 4 the TRNG unit based on the Sprott 94 G chaotic attractor implemented on the FPGA for the additional input. Section 5 presents the hybrid PRNG system developed. The final section discusses the results.

2 Pseudo-random number generator

A PRNG uses the seed value received from any entropy source as a random input to generate random number streams. As these generators are deterministic, the output value cannot exceed the seed value introduced. Moreover, their sequences repeat themselves after a while. In other words, the system has a "period" [3,16,17].

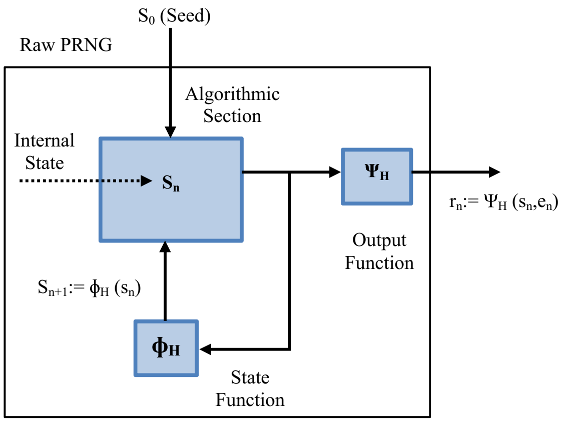


Fig. 1 Raw PRNG general block diagram

Figure 1 shows the general block diagram of a raw PRNG. A raw PRNG is defined with the set of variables (S, R, Ψ, Φ, p_s).

- s_n = internal state value, $s_n \in S$
- S : state space, R: output space
- p_s : probability distribution of random seed
- Ψ : output function, Φ : transition function
- $\Psi : S \rightarrow R, r_n := \Psi_H(s_n, e_n)$ the output function calculates r_n, the random number following the current internal state value s_n
- $s_1 = \Phi(s_0)$ function calculates the next internal state value

During generation, the important state, all the state values $s_1, s_2, \ldots, s_n$ and generations of all the generated random numbers $r_1, r_2, \ldots, r_n$ depend completely on the seed value s_0. This dependency leads to the risk of regeneration of the entire system once the s_0 value is known. Therefore, in order to ensure unpredictability, the seed value s_0 must be chosen randomly. The seed is generally generated by a TRNG in order to ensure unpredictability [3].

A drawback of PRNGs (compared to TRNGs) is that the output is determined completely by the seed and the next random number depends only on the current internal state value. Hence, the internal state value must be maintained even when the system is not active. To ensure unpredictability of a raw PRNG, the seed entropy must be high and transition and output functions must be adequately complex. In order to correct such deficiencies and ensure complexity of the functions, the system is provided with an additional input from the finite set E, as seen in Fig. 2, which is different from a raw PRNG. The additional input improves the security as it makes the system unpredictable and random.

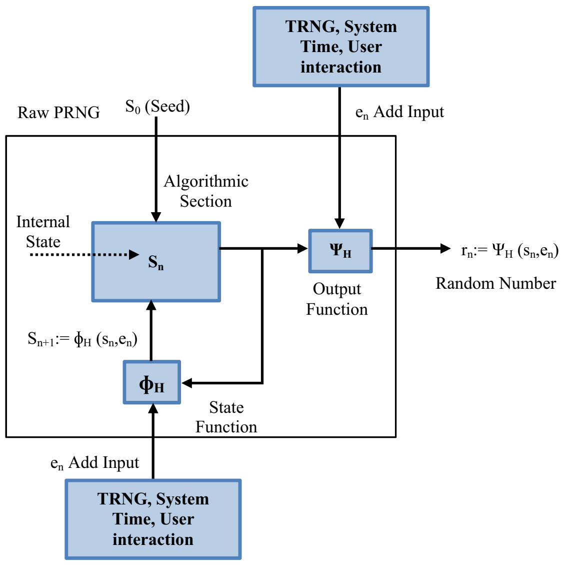


Fig. 2 Hybrid PRNG general block diagram

3 Advanced encryption standard (AES) structure

The AES block encryption algorithm is used for the algorithmic part of the developed system. The algorithmic part of the AES meets the R1–R3 requirements as necessary for random number generators. AES is the applicable block encryption standard developed by J. Daemen and V. Rijmen in 1997 and adopted as a standard in 2000. It is more secure and efficient compared to encryption algorithms such as DES and 3DES. AES is an iterative block cipher based on a design principle known as a substitution-permutation network (SPN). AES operates on a 4 × 4 column-major order matrix of bytes, called the state. Matrix calculations are done in a special finite field. AES supports 128-, 192-, 256-bit keys. The number of cycles of repetition for 128-bit, 192-bit, and 256-bit keys is 10, 12, and 14, respectively. As shown in Fig. 3, AES consists of several stages, each processed through 128-bit data input. These stages include key addition, byte substitution, ShiftRow, and MixColumn. The final cycle does not include MixColumn [21,22].
AES groups the message, expressed in bits, into bytes.

- $x_{00}\, x_{10}\, x_{20}\, x_{01}\, x_{11}\, x_{21}\, x_{31}\, x_{02}\, x_{22}\, x_{32}\, x_{03}\, x_{13}\, x_{23}\, x_{33} \ldots\ldots\ldots$

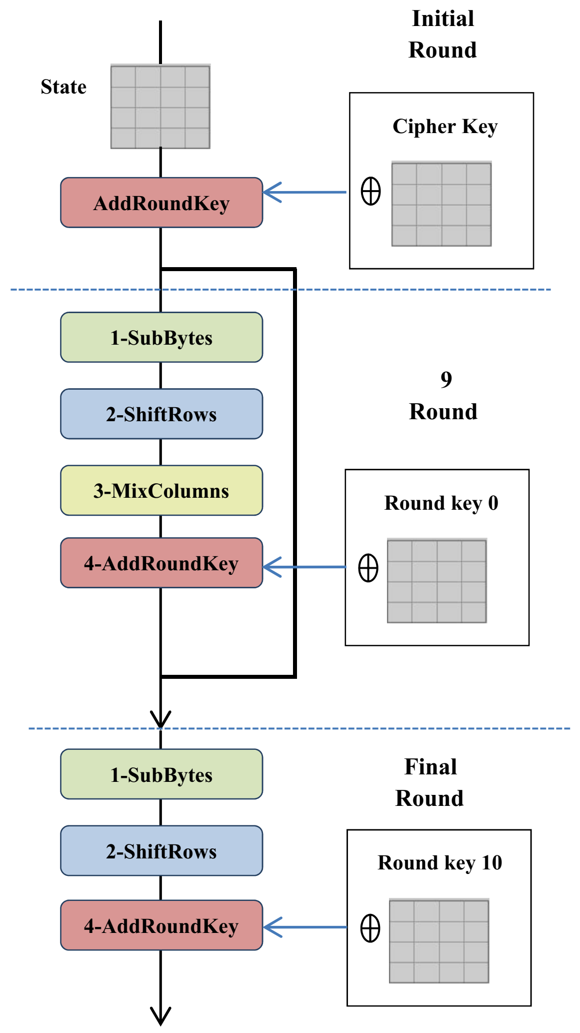


Fig. 3 AES structure [23]

The received text is expressed in 4×4 matrices for a 128-bit key, 4×6 matrices for a 192-bit key, and 4×8 matrices for a 256-bit key. For a 128-bit text received, the generated matrix is shown in Eq. (1).

- *Byte Sub* Byte substitution step consists of 16 S-boxes (substitution box). These S-boxes are identical and the only nonlinear elements of AES. In this step, each byte in the matrix is updated with an 8-bit S-box. For this purpose, the inverse of each byte in the matrix is determined modulo $m(a) = a^8 + a^4 + a^3 + a + 1$, as shown in Eq. (2). If $x = x^{-1} = b(b_7\, b_6\, b_5\, b_4\, b_3\, b_2\, b_1\, b_0)$ and S-box output is $c = (c_7\, c_6\, c_5\, c_4\, c_3\, c_2\, c_1\, c_0)$ the 8×8 matrix has an inverse modulo 2.
- *Shift Row* This operation, operating on the matrix rows, shifts the byte values in each row by a certain offset. For AES, the first row is left unchanged, while each byte of the second, third, and fourth rows are shifted to the left by offsets of one, two, and three, respectively. For a 128-bit text received, the shifting is shown in Eq. (3).
- *Mix Column* In this step, the 4 bytes of each column are combined using an invertible linear transformation, as shown in Eq. (4). Therefore, each column is multiplied modulo $a^4 + 1$ with the polynomial $z(a) = \text{‘}03'a^3 + \text{‘}01'a^2 + \text{‘}01'a + 02$. This multiplication is done in a finite field (Galois Field) GF(2^8).
- *Add Roundkey* In this step, the round key is combined with the state matrix. Once the round key is divided into bytes, it is combined with the corresponding matrix elements using bitwise XOR, as shown in Eq. (5).

$$\begin{bmatrix} x_{00} & x_{01} & x_{02} & x_{03} \\ x_{10} & x_{11} & x_{12} & x_{13} \\ x_{20} & x_{21} & x_{22} & x_{23} \\ x_{30} & x_{31} & x_{32} & x_{33} \end{bmatrix} \tag{1}$$

$$\begin{bmatrix} c_0 \\ c_1 \\ c_2 \\ c_3 \\ c_4 \\ c_5 \\ c_6 \\ c_7 \end{bmatrix} = \begin{bmatrix} 1&1&1&1&1&0&0&0 \\ 0&1&1&1&1&1&0&0 \\ 0&0&1&1&1&1&1&0 \\ 0&0&0&1&1&1&1&1 \\ 1&0&0&0&1&1&1&1 \\ 1&1&0&0&0&1&1&1 \\ 1&1&1&0&0&0&1&1 \\ 1&1&1&1&0&0&0&1 \end{bmatrix} \begin{bmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \\ a_4 \\ a_5 \\ a_6 \\ a_7 \end{bmatrix} \oplus \begin{bmatrix} 0 \\ 1 \\ 1 \\ 0 \\ 0 \\ 0 \\ 1 \\ 1 \end{bmatrix} \tag{2}$$

$$\begin{bmatrix} x_{00} & x_{01} & x_{02} & x_{03} \\ x_{10} & x_{11} & x_{12} & x_{13} \\ x_{20} & x_{21} & x_{22} & x_{23} \\ x_{30} & x_{31} & x_{32} & x_{33} \end{bmatrix} = \begin{bmatrix} x_{00} & x_{01} & x_{02} & x_{03} \\ x_{10} & x_{11} & x_{12} & x_{13} \\ x_{20} & x_{21} & x_{22} & x_{23} \\ x_{30} & x_{31} & x_{32} & x_{33} \end{bmatrix} \begin{array}{l} \text{unchanged.} \\ \text{shifted to the left by 1 bytes.} \\ \text{shifted to the left by 2 bytes.} \\ \text{shifted to the left by 3 bytes.} \end{array} \tag{3}$$

$$\begin{bmatrix} b_{00} & b_{01} & b_{02} & b_{03} \\ b_{10} & b_{11} & b_{12} & b_{13} \\ b_{20} & b_{21} & b_{22} & b_{23} \\ b_{30} & b_{31} & b_{32} & b_{33} \end{bmatrix} = \begin{bmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{bmatrix} \begin{bmatrix} x_{00} & x_{01} & x_{02} & x_{03} \\ x_{10} & x_{11} & x_{12} & x_{13} \\ x_{20} & x_{21} & x_{22} & x_{23} \\ x_{30} & x_{31} & x_{32} & x_{33} \end{bmatrix} \tag{4}$$

$$\begin{bmatrix} x_{00} & x_{01} & x_{02} & x_{03} \\ x_{10} & x_{11} & x_{12} & x_{13} \\ x_{20} & x_{21} & x_{22} & x_{23} \\ x_{30} & x_{31} & x_{32} & x_{33} \end{bmatrix} \oplus \begin{bmatrix} k_{00} & k_{01} & k_{02} & k_{03} \\ k_{10} & k_{11} & k_{12} & k_{13} \\ k_{20} & k_{21} & k_{22} & k_{23} \\ k_{30} & k_{31} & k_{32} & k_{33} \end{bmatrix}$$

$$= \begin{bmatrix} x_{00} \oplus k_{00} & x_{01} \oplus k_{01} & x_{02} \oplus k_{02} & x_{03} \oplus k_{03} \\ x_{10} \oplus k_{10} & x_{11} \oplus k_{11} & x_{12} \oplus k_{12} & x_{13} \oplus k_{13} \\ x_{20} \oplus k_{20} & x_{21} \oplus k_{21} & x_{22} \oplus k_{22} & x_{23} \oplus k_{23} \\ x_{30} \oplus k_{30} & x_{31} \oplus k_{31} & x_{32} \oplus k_{32} & x_{33} \oplus k_{33} \end{bmatrix} \quad (5)$$

4 Sprott 94 G chaotic attractor-based TRNG unit

The equation sets for the Sprott 94 G nonlinear autonomous chaotic system used in the proposed system are listed in Eq. (6). Any change of the parameter Ψ in this equation set alters the dynamic behavior of the chaotic system. Therefore, values of these parameters are very important. In this study, the parameter Ψ is chosen to be 0.40. The initial conditions (ICs) of chaotic systems determine their chaotic behavior. These systems are defined as sensitively dependent on initial conditions. Thus, chaotic systems may not exhibit chaotic behavior close to the initial condition values. In other words, chaotic behavior only is only exhibited at certain values. For instance, when the initial conditions of the proposed chaotic model were $x_0 = 0.6, y_0 = 0.6, z_0 = 0.6$, system showed no chaotic behavior; however, when these values were $x_0 = 0.5, y_0 = 0.5, z_0 = 0.5$, the system showed chaotic behavior. This study used initial condition values that cause the system to exhibit chaotic behavior, namely, $x_0 = 0.05$, $y_0 = 0.05$, $z_0 = 0.05$.

$$\begin{aligned} dx/dt &= \psi \cdot x + z \\ dy/dt &= z \cdot x - y \\ dz/dt &= -x + y \end{aligned} \quad (6)$$

The TRNG unit used in the system is designed in hardware manner to operate on FPGA chips in accordance with 32-bit IEEE 754-1985 floating point number standard. The designed chaotic oscillator unit has been coded in the hardware description language VHDL and synthesized with the design platform Xilinx ISE 14.2 for the Virtex-6 FPGA chip. Adders, dividers, multipliers, and other modules used during the design have been derived by using Xilinx IP CORE Generator Tools. A block diagram of the chaos-based TRNG unit designed with the RK5-Butcher algorithm can be seen in Fig. 4 [24]. 32-bit τ signal in chaotic TRNG units represents the steps of the RK-5 Butcher algorithm. In τ step further values of the chaotic system $x(k+1)$, $y(k+1)$, and $z(k+1)$ were calculated using $x(k)$, $y(k)$, and $z(k)$ values in a discrete model which was developed by employing RK-5 Butcher algorithm with initial condition values $x_0 = 0.05$, $y_0 = 0.05$ and $z_0 = 0.05$. At the end of each iteration, the output values of the system $x(k+1)$, $y(k+1)$ and $z(k+1)$ were considered as outputs and also initial conditions for the next iteration. The τ signal has to be a small enough value to show chaotic behavior when the system is modeled numerically, and it has to be a large enough value to avoid radical changes in the time series of a chaotic system. Thus, in this study, a τ value of 0.025 was used to calculate discrete chaotic signals $x(k+1)$, $y(k+1)$, and $z(k+1)$ in each iteration. The 1-bit *Clk* and *Start* signals are used to ensure synchronization between the chaotic oscillator and the system to which it is connected. The $k_0, k_1, k_2, k_3, k_4, k_5$, and *ys* units of the oscillator calculate the coefficients of the RK5 Butcher algorithm. The *ys* unit multiplies $k_0, k_1, k_2, k_3, k_4, k_5$ values with floating point values in the algorithm to obtain the output signals of the oscillator and then sends these values to a filter unit. The filter unit is used to filter error signals that can occur on the outlet of the chaotic oscillator. The 32-bit *X_S*, *Y_S*, and *Z_S*, output signals of the *filter* unit, are the signals generated by the chaotic oscillator. In addition, these signals are transferred again to the *MUX* unit as $x(k+1)$, $y(k+1)$, and $z(k+1)$ signals, as they will be used to form the initial conditions for the output signals to be generated by the oscillator at the moment of $t+1$. The duty of the *MUX* unit is to ensure the initial conditions required by the system are assigned when the Start signal is received. In other words, it makes a choice between the initial conditions (*t0_x*, *t0_y*, and *t0_z*) assigned to the system by the user and the initial conditions obtained by the output of the system [($x(k+1)$, $y(k+1)$, and $z(k+1)$] and transfers these signals to the system. The 1-bit *R_XYZ* signal shows the signals generated by the chaotic oscillator are ready. When the chaotic oscillator generates output signals, the *R_XYZ* signal is asserted as “1”, and in other cases, it transmits “0” to the output. The *Quantizer* unit uses the values received from the filter unit for comparison. The Sprott 94 G chaotic oscillator transmits the generated values to the *Quantizer* unit. The values σ_x, σ_y and σ_z were adaptively calculated using Adder, Counter, Divider, and Comparator units in the Quantizer. The Adder unit collects signal values generated by the chaotic oscillator in each signal generation. The Counter unit calculates generated value of oscillator by using *R_XYZ* signals retrieved from the chaotic

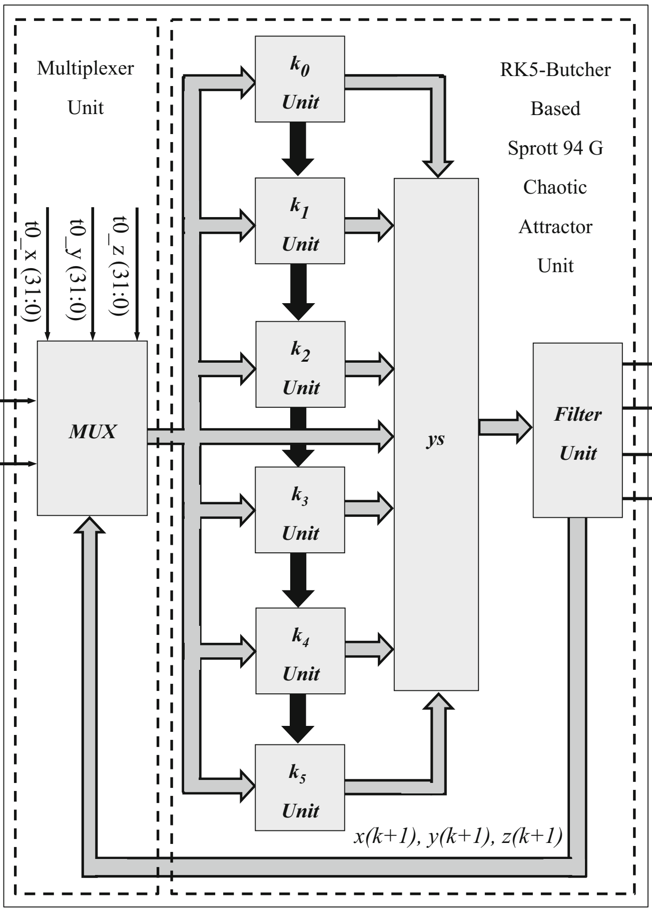
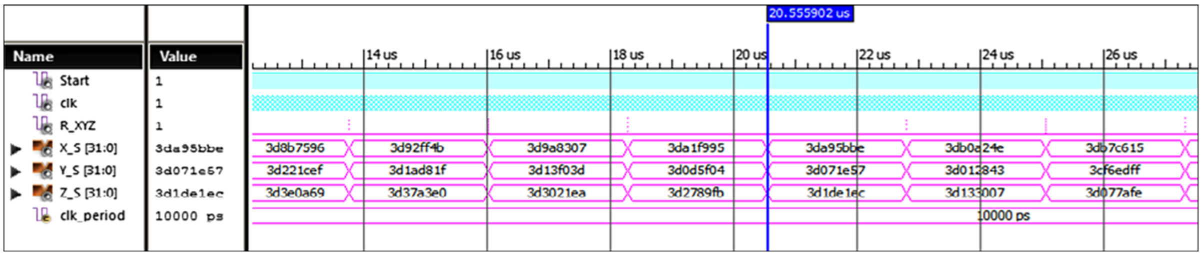


Fig. 4 Block diagram of chaos-based TRNG unit

oscillator and transmits this value to Divider unit. The Divider unit conducts the division process using signals retrieved from Adder and Counter units. The output of Divider unit transmits the values σ_x, σ_y, and σ_z to the Comparator unit, and the comparison process is done with Eq. (7). This approach helps to calculate the threshold values of σ_x, σ_y, and σ_z adaptively. This unit makes a comparison including the received 32-bit signal in order to perform the operation in Eq. (7). In this operation, σ is referred to as the threshold value, which varies depending on the characteristics of the chaotic system in use. The *Q_RN* signal as the output of this unit carries the random numbers. *QRN_R* signal indicates that random signals are received at the output of the unit. In the final step, *Q_RN* and *QRN_R* signals are transmitted to the *ART* unit, where they are subjected to



Name	Value
Start	1
clk	1
R_XYZ	1
X_S [31:0]	3da95bbe
Y_S [31:0]	3d071e57
Z_S [31:0]	3d1de1ec
clk_period	10000 ps

20.555902 us
14 us
16 us
18 us
20 us
22 us
24 us
26 us
3d8b7596 3d92ff4b 3d9a8307 3da1f995 3da95bbe 3db0a24e 3db7c615
3d221cef 3d1ad81f 3d13f03d 3d0d5f04 3d071e57 3d012843 3cf6edff
3d3e0a69 3d37a3e0 3d3021ea 3d2789fb 3d1de1ec 3d133007 3d077afe
10000 ps

Fig. 5 Time series generated by Xilinx ISE Simulator of FPGA-based Sprott 94 G chaotic oscillator

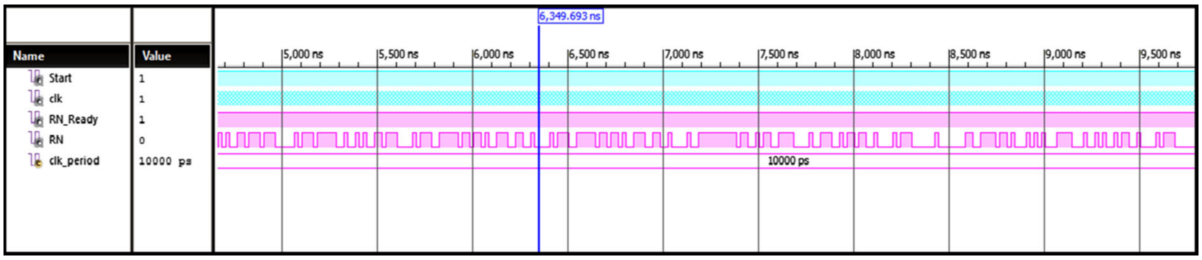


Fig. 6 Results of FPGA-based chaotic TRNG obtained from Xilinx ISE Simulator

Table 1 FPGA chip statistics for the chaotic TRNG unit

FPGA chip	Slice regs. number/%	LUTs number/%	Occupied slices number/%	IOBs number/%	Min. period (ns)
Virtex-6	68,665/10	71,346/21	21,643/25	4/1	2.948

an XOR operation, and then the random numbers from this unit are transferred to the outlet.

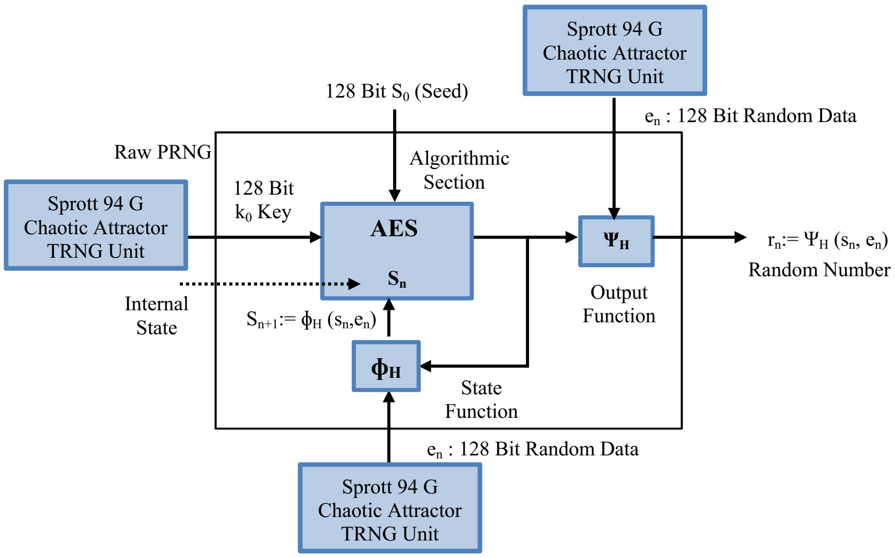
$$\mathrm{RS}(x, y, z) = \begin{cases} 0 & x, y, z < \sigma \\ 1 & x, y, z >= \sigma \end{cases} \tag{7}$$

The Sprott 94 G chaotic oscillator unit, which is designed as FPGA-based with RK-5 Butcher algorithm, is synthesized for XC6VLX550T chip, which is one of the smallest chips of Xilinx Virtex-6 family. Operating frequency of the designed unit is measured as 339 MHz with Xilinx ISE Design Tools 14.2 simulation program. The results of FPGA-based chaotic oscillator obtained from Xilinx ISE Simulator are presented in Figs. 5 and 6. The figure shows the values for the time series of the outputs X_S, Y_S and Z_S, which are discretized equivalents of the signals x, y, and z obtained from the implementation of the chaotic oscillator on an FPGA by means of ISE Design Tools. The simulation results are demonstrated in a hexadecimal format in order to facilitate analysis of the results. The FPGA chip statistics obtained from the Place-Route process which was conducted pursuant to the synthesizing process for the FPGA-based chaotic TRNG unit are presented in Table 1.

5 Developed Hybrid PRNG

Functions such as AES and 3DES provide the complex structure required for raw PRNG. However, the inner value of raw PRNG can be extracted since raw PRNG

Fig. 7 General design of the proposed hybrid PRNG



users and owners are potentially aggressive. Thus, R4 requirements cannot be met even though the structure is complex. The chaotic circuit being used is a nonlinear system that is sensitively dependent to initial conditions. Even though the system is deterministic, since it interacts with its surroundings, it is affected by noise and environmental conditions, and this interaction can cause unpredictable outcomes since the system is sensitively dependent on initial conditions. Since predicting the dynamic behavior of chaos is impossible for the user, unpredictable additional input values were generated using the deterministic circuit. The hybrid PRNG system developed to ensure raw PRNG systems meet the R4 security requirement is shown in Fig. 7. For this purpose, a raw PRNG has been established by means of the AES. The additional input to the system is the random bit sequence obtained from the TRNG implemented on Xilinx FPGA by means of the Sprott 94 G chaotic attractor.

For the developed system, the seed value s_0 to be input to the system was randomly selected first and k_0 was taken from the Sprott 94 G Chaotic Attractor TRNG Unit.

- s_0 : 101010110100010101000101010..
- k_0 : 010101010001010111010101010..

These values have been input to the AES system shown in Fig. 4 to obtain the current state value s_n after ten cycles.

- s_n : 010101010100101111100111001..

This result was combined using XOR with the additional 128-bit input generated by the TRNG unit implemented, as shown in Eq. (8).

- e_n:011111000000011110100101000..

$$r_n := \Psi_H\,(s_n \oplus e_n) \tag{8}$$

- $r_n := \Psi_H(010101010100101111100111001\ldots \oplus 01111100000001111101001010010\ldots)$
- r_n : 0010101010101010100011011110...

The r_n value obtained was transferred to the output and s_n was input to the system as the new seed value.

As a result, the developed system meets the R4 requirement as no attacker cannot predict the random numbers generated although he knows the seed value or the internal state value because the internal state value is updated with random data in every step. In addition, with the value of key k_0 generated by a TRNG, any distinguishing attack the AES system can be avoided. The

Table 2 Test results of the proposed hybrid PRNG

Test name	*P* value		Result
Frequency test	0.170		Passed
Frequency test within a block test	0.432		Passed
Runs test	0.347		Passed
Test for the longest run of ones in a block test	0.390		Passed
Binary matrix rank test	0.675		Passed
Discrete Fourier transform	0.364		Passed
Non-overlapping template matching test	0.746		Passed
Overlapping template matching test	0.109		Passed
Maurer's universal statistical test	0.138		Passed
Linear complexity test	0.60		Passed
Serial test	0.349		Passed
	0.285		
Lempel Ziv test	1		Passed
Approximate entropy test	0.700		Passed
Cumulative sums test	0.221		Passed
Random excursions test	−4	0.690	Passed
	−3	0.963	
	−2	0.992	
	−1	0.758	
	1	0.831	
	2	0.198	
	3	0.045	
	4	0.044	
Random excursions variant test	−9	0.140	Passed
	−8	0.069	
	−7	0.068	
	−6	0.152	
	−5	0.237	
	−4	0.345	
	−3	0.524	
	−2	0.898	
	−1	0.949	
	1	0.184	
	2	0.176	
	3	0.488	
	4	0.667	
	5	0.547	

Table 2 continued

Test name	*P* value		Result
	6	0.439	
	7	0.648	
	8	0.787	
	9	0.835	

system has been made appropriate for cryptographic applications. Table 2 shows the random 1000000 bit sequences obtained from the system have passed NIST statistical tests, the software of which was implemented in [25].

6 Conclusion

In this study, providing an additional input to the system in order to ensure raw pseudo-random number generators meet all the security requirements and to improve complexity of the functions in use and the corresponding results are discussed. Raw PRNGs meet the first three of the R1–R4 security requirements. For this purpose, a hybrid PRNG system has been developed by adding a TRNG implemented with the Sprott 94 G chaotic attractor, as the additional input to the transition and output functions of a raw PRNG, in order to improve the randomness and security of the system. The developed system guarantees no attacker can predict the random numbers generated although he knows the seed value or the internal state value because the internal state value is updated with random data in every step. In addition, with the value of key k_0 generated by a TRNG, any attack to identify the key to the AES system can be avoided. The bit sequences obtained from the developed hybrid PRNG have successfully passed the NIST-800-22 and FIPS statistical tests and met all the security requirements. The results obtained from the hybrid PRNG have shown that the proposed hybrid system can be used in cryptography.

Acknowledgments We thank the referees contributing to the development of the article with their positive critics.

References

1. Corrigan-Gibbs, H., Mu, W., Boneh, D., Ford, B.: Ensuring high-quality randomness in cryptographic key generation. In: CCS'13 Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security, pp. 685–696 (2013). doi:10.1145/2508859.2516680
2. Wold, K.: Security properties of a class of true random number generators in programmable logic. Thesis submitted to Gjøvik University College for the degree of Doctor of Philosophy in Information Security (2011)
3. Koç, Ç.: Cryptographic Engineering. Springer, New York (2009)
4. Calegari, S., Rovatti, R.: Embeddable ADC-based true random number generator for cryptographic applications exploiting nonlinear signal processing and chaos. IEEE Trans. signal process. **53**(2), 793–805 (2005)
5. Hu, Y., Liao, X., Wong, K., Zhou, Q.: A true random number generator based on mouse movement and chaotic cryptography. Chaos Solitons Fractals **40**(5), 2286–2293 (2009). doi:10.1016/j.chaos.2007.10.022
6. Pareschi, F., Setti, G., Rovatti, R.: A fast chaos-based true random number generator for cryptographic applications. In: Solid-State Circuits Conference. ESSCIRC 2006, pp. 130–133 (2006). doi:10.1109/ESSCIR.2006.307548
7. Tuncer, T., Avaroğlu, E., Türk, M., Özer, A.B.: Implementation of non-periodic sampling true random number generator on FPGA. J. Microelectron. Electron. Compon. Mater **4**(4), 296–302 (2014)
8. Vasyltsov, I., Hambardzumyan, E., Kım, Y.-S., Karpinskyy, B.: Fast digital TRNG based on metastable ring oscillator. In: Proceedings of the 10th International Workshop on Cryptographic Hardware and Embedded Systems (CHES'08), vol. 5154 of Lecture Notes in Computer Science, Springer, pp. 164–180 (2008)
9. Ergün, S., Özoğuz, S.: A chaos-modulated dual oscillator-based truly random number generator. In: Proceedings, International Symposium on Circuits and Systems, pp. 2482–2485 (2007)
10. Beirami, A., Nejati, H., Massoud, Y.: A performance metric for discrete-time chaos-based truly random number generators. In: Circuits and Systems, MWSCAS 2008. 51st Midwest Symposium on, pp. 133–136 (2008)
11. Yalcın, M.E., Suykens, J.A.K., Vandewalle, J.: True random bit generation from a double scroll attractor. IEEE Trans. Circuits Syst-I **51**(7), 1395–1404 (2004)
12. Koyuncu, I., Ozcerit, A.T., Pehlivan, I.: Implementation of FPGA-based real time novel chaotic oscillator. Nonlinear Dyn. **77**(1–2), 49–59 (2014). doi:10.1007/s11071-014-1272-x
13. Wang, X.-Y., Oin, X.: A new pseudo random number generator based on CML and chaotic iteration. Nonlinear Dyn. **70**(2), 1589–1592 (2012). doi:10.1007/s11071-012-0558-0
14. Keller, S.S.: NIST-recommended random number generator based on ANSI X9.31 appendix A.2.4 using the 3-key triple DES and AES algorithms. In: National Institute of Standards and Technology Information Technology Laboratory Computer Security Division, 31 January 2005
15. Prescott, T.: Random Number Generation. Automot.Compil. **8**, 30–34 (2011)
16. Akram, R.N.: Pseudorandom number generation in smart cards an implementation performance and randomness analysis. In: New Technologies, Mobility and Security (NTMS), 2012 5th International Conference on, 7–10 May 2012, pp. 1–7

17. Sobotka, J., Zeman, V.: Design of the true random numbers generator. Elektrorevue **2**(3), 1–6 (2011)
18. Özkaynak, F.: Cryptographically secure random number generator with chaotic additional input. Nonlinear Dynamics (2014). doi:10.1007/s11071-014-1591-y
19. Avaroğlu, E., Tuncer, T., Özer, A.B., Türk, M.: A new method for hybrid pseudo random number generator. J. Microelectron. Electron. Compon. Mater. **4**(4), 311 (2014)
20. Bodanov, A., Khovratovich, D., Rechberger C.: Biclique cryptanalysis of the full AES. In: ASIACRYPT'11 Proceedings of the 17th international conference on the Theory and Application of Cryptology and Information Security, pp. 344–371 (2011). doi:10.1007/978-3-642-25385-0_19
21. Kriptoloji Seminer Notları: Uygulamalı Matematik Enstitüsü, Kriptografi Bölümü, ODTÜ, Türkiye (Şubat 2004)
22. Daemen, J., Rijmen, V.: The Design of Rijndael: AES–The Advanced Encryption Standard. Springer, New York (2001)
23. http://www.cs.bc.edu/~straubin/cs381-05/blockciphers/rijndael_ingles2004.swf
24. Koyuncu, I., Ozcerit, A.T., ve Pehlivan, I.: An analog circuit design and FPGA-based implementation of the Burke-Shaw chaotic system. Optoelectron. Adv. Mater. Rapid Commun. **7**(9–10), 635–638 (2013)
25. Avaroğlu, E.: Donanım Tabanlı Rasgele Sayı Üretecinin Gerçekleştilmesi. Ph.D, Fırat Üniversitesi, Elazığ, Türkiye (2014)