An Ultra-light PRNG Passing Strict Randomness Tests and Suitable for Low Cost Tags

Mehmet Hilal ÖZCANHAN[1], Mehmet Suleyman UNLUTURK[2], Gökhan DALKILIÇ[1]
[1]*Department of Computer Engineering, Dokuz Eylul University, Izmir, Turkey*
[2]*Department of Software Engineering, Yasar University, Izmir, Turkey*
dalkilic@cs.deu.edu.tr

***Abstract*—A pseudo-random number generator for low-cost RFID tags is presented. The scheme is simple, sequential and secure, yet has a high performance. Despite its lowest hardware complexity, our proposal represents a better alternative than previous proposals for low-cost tags. The scheme is based on the well-founded pseudo random number generator, Mersenne Twister. The proposed generator takes low-entropy seeds extracted from a physical characteristic of the tag and produces outputs that pass popular randomness tests. Contrarily, previous proposal tests are based on random number inputs from a popular online source, which are simply unavailable to tags. The high performance and satisfactory randomness of present work are supported by extensive test results and compared with similar previous works. Comparison using proven estimation formulae indicates that our proposal has the best hardware complexity, power consumption, and the least cost.**

***Index Terms*—information security, radio frequency identification, random number generation, RFID tags, ubiquitous computing.**

I. Introduction

Consumer goods in supply chains have identification stickers on them, which are used to track their journey up to the consumer bags. The paper barcode stickers are being replaced by Radio Frequency Identification (RFID) stickers (tags) [1]. According to a report, RFID is a booming technology, acting as part of the ubiquitous systems [2]. The boom is due to advantages gained by using RFID, since paper barcodes suffer from wear and tear. Degraded barcodes cause erroneous reading and when intentionally switched or over-written by fakes, big losses occur. On the other hand, RFID tags provide some security in addition to a unique identification number; by employing pseudo random number (PRN) based protocols. PRNs are widely used in computer and mobile device authentication protocols, relying on the very important characteristic of being random. But, producing good quality PRNs in cheap tags is not easy, as it will be revealed next.

A. RFID Technology: Properties, Advantages, and Shortcomings

RFID rests upon wireless technology, where a tiny tag consists of an integrated circuit (IC). The IC is remotely energized by the reader through electromagnetic field of antenna coils, as shown in Fig. 1. The unique and sensitive identification number (ID) information in the memory of the tag is obtained and sent to a remote server by the reader, to be matched to an entry in the database. In a nutshell, RFID "is capable of identifying items of different types and distinguishing between items of the same type, not requiring physical or visual contact" [3]. While paper barcodes are read one at a time, up to 1000 tags/sec can be read. The reading distance of a barcode is a few centimeters, whereas a UHF tag's is a few meters. The low-cost, passive, UHF tags are the most popular [2] and take part in the pervasive networks that surround us [4]. However, their computational capacity and power electronics are limited, forcing little hardware to be spared for security; hence, causing vulnerabilities [5-6]. Therefore, UHF tags have been defined as resource constrained devices and the cryptographic solutions offered to remedy security vulnerabilities have been studied, in detail [7].

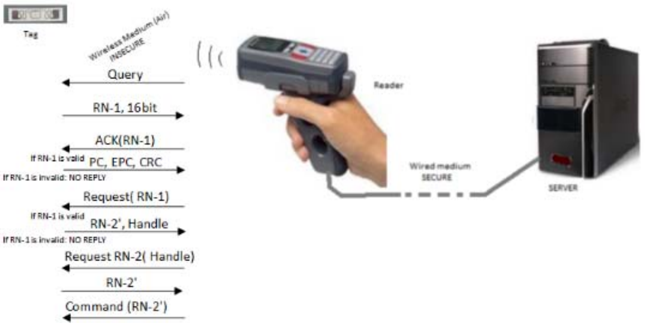

Figure 1. The flow of RFID authentication process

The communication between the reader and the server shown in Fig. 1 is considered secure; whereas communication through air, between the reader and the tag is assumed insecure. Security issues arise when the tag tries to pass its sensitive ID or Electronic Product Code (EPC). The properties ratified in ISO-18000-6 [8] and recently released EPCglobal Class-1 Generation-2 version 2 (Gen-2) standards [9] fail to provide the necessary security. According to the standards, every queried tag transmits 16 bits of generated random numbers (RNs), starting with RN-1. The reader acknowledges by returning a reply with the same RN and the tag replies with its EPC. The commands consist of words XORed with random number RN-2. A malicious listener can capture the RNs and extract the EPC. The only effort to decrypt a command is to XOR it with the recorded RN-2 [3]. Once the ID is captured, the least damage can be the exposure of personal privacy. The UHF tags should not be confused with the high cost tags, such as those used in e-passports.

Digital Object Identifier 10.4316/AECE.2016.03012

1582-7445 © 2016 AECE

B. Random Number Generation and Randomness Tests

Pseudo random number generator (PRNG) function is a fundamental primitive of most cryptographic algorithms and security protocols. The Gen-2 standard supports 16-bit PRNs basically for collision prevention. But, neither the period (2^{16}), nor the random number generation standard of Gen-2 is satisfactory for cryptographic randomness. In addition, low-cost tags have limited capacity and can dedicate only a few thousand gates for security [3, 10]. The above disadvantages are worsened with the following Gen-2 PRN property guidelines:

1. The probability that any value j of a generated 16-bit random number (RN16), where RN16 = j shall be bounded by: $0.8/2^{16} < P(RN16=j) < 1.25/2^{16}$.
2. For a tag population of up to 10,000 tags, the probability that any of two or more tags simultaneously generate the same sequence shall be less than 0.1%, regardless of when the tags are energized.
3. An RN16 drawn from a tag's PRNG 10 ms after the end of tag power-up rise time, shall not be predictable with a probability greater than 0.025%.

Many random number generation proposals made for tags, consider meeting the above guidelines as adequate. However, Gen-2 guidelines are not as strict as the criteria of well-established PRN tests. More serious effort is needed to provide good quality random numbers. Efforts of generating random numbers can be divided into three categories. The first category is true random number (TRN) generators (TRNGs), where randomness is extracted from physical phenomena like thermal noise, frequency instability of an oscillator or the unstable power-up state of SRAM memory cells [11, 12]. For example, work [11] uses chaotic oscillations of a designed electronic circuit. Although the proposed solution passes statistical randomness tests, the hardware implementation exceeds allowed power consumption restrictions, with 26100 nW performance. TRNGs rely on physical characteristics and mostly fail to provide good quality random numbers; even tend to output the same numbers - if physical conditions are reproduced [13]. Therefore, TRNGs are widely regarded as sources of entropy, only.

The second category is PRNGs, where a deterministic algorithm is used [3, 10]. PRNGs use mathematical formulae or pre-calculated tables to generate sequences that appear random. The third category is a blend of the first and second categories; where output obtained from a TRNG is used to seed a PRNG [12-14]. This strategy has been developed because, if the assumed seed of a PRNG is guessed, the generated RN of the mathematically deterministic algorithm can be predicted. Therefore, a PRNG by itself is sometimes declared insecure without good sources for seeding [15, 16]. But, work [14] demonstrates weakness in a previous PRNG design, even though the TRNG seeds are truly random. It follows that the PRNG algorithms providing the final output must be strong. To obtain high quality random numbers, another option is the entropy amplification of low entropy TRN seeds. For example, work [15] tries to increase the quality of generated TRNs, via hardware based multi-phase timing of bistables.

Obviously, the randomness of the sequences generated by PRNGs has to be proven. There are various mathematical and statistical proofs for randomness, but today's popular method is to get approval from universally accepted randomness checkers (tests). A large amount of input is given as seed to the proposed generators and their output number sequences are submitted to the randomness tests. Each test suite has its own submission rules, randomness checks and result interpretations. Discussing the rules, test suits and interpretation of randomness tests are beyond the scope of this paper. The ENT, Diehard (versions 1, 2) and NIST randomness tests (detailed in Section 4.2) are highly esteemed, by the random number community. But, any non-obscure algorithm producing output that passes the NIST tests is widely accepted. Therefore, our results have been tested using the above three tests. Today's most randomness tests are designed for 32-bit RNs, because the use of 16-bit RNs in computers is now considered to be too primitive. Hashing and encryption algorithms like SHA-1, DES, AES etc. are known to produce good quality random numbers; however, they overwhelm the low-cost RFID tags [6, 17].

In the rest of this paper, Section 2 accounts for the previous related work in PRNG design and the properties of our inspired algorithm. In Section 3, there are the details of our proposed scheme. Section 4 contains the performance and testing results, followed by evaluation and comparison of the results with similar works. We conclude and list future work, in Section 5.

II. Related Works

There aren't many works on PRNGs, employed in tags. Among the few, - only in [3] and [10] - the proposed schemes are supported with popular randomness tests and provide detailed design-performance information. Two other works [13-14] do not provide the same information, but propose a linear feedback shift register (LFSR) with a TRN bit, as input. In fact, in [13] the authors attack work [14] and try to correct the suggested weak LFSR.

Our work falls into the third category described in Section 1.B, which uses the power-up contents of the SRAM memory of a tag to form a TRN input to its PRNG function. It has been demonstrated that some SRAM memory bits always settle to the same "0" or "1" voltage level, while others settle to either low or high level, randomly. The randomness is due to semiconductor lithography variations and thermal noise. This SRAM cell characteristic has led to the proposal of Fingerprint Extraction and Random Numbers in SRAM (FERNS) [12]. A length of memory containing random bits provides a low source of entropy in FERNS, which by themselves fail the randomness tests. The biggest advantage of FERNS is that it requires no dedicated circuitry. In [12] the TRNs are input to a well-known hash function for privacy amplification, to attain good quality random numbers. Randomness test results indicate that the randomness of the original TRNs is not acceptable, but the hashing algorithm produces random numbers that pass randomness tests. Our proposal replaces the hashing algorithm of the work [12] with an affordable, ultra-light scheme (described in detail in Section 3). In tags, ultra-light means a scheme or protocol where only bitwise AND, OR, XOR, shift and addition modulo 2^m are used [3, 5]. The main reason of replacing the hash function is due to the fact that present hash functions cannot fit in low-cost tags.

Moreover, hashing needs considerable amount of input bits. Both factors raise the cost of the tags. One of the most known works on tag PRNGs is Lamed [3], but it has some undesired characteristics that can be summarized as follows:

- Lamed is a computation intense algorithm with 8 addition, 6 XOR and 10 rotation operations, requiring a 32-bit parallel architecture to finish computations, within required time. Its clock cycle delay for PRN generation is compared to our design in Section 4.A.
- The authors of Lamed declare that the public initialization vector (IV) should never be used with the same pre-shared secret key (PSK). In addition, the notion of programming a constant PSK into the tag - at the time of manufacture - is not defined in Gen-2. Meanwhile, the probability of using the same key with a known public IV is merely $1/2^{16}$. Therefore, the key space of different IVs against different keys is not enough for low-cost tags, produced in millions.
- The authors of Lamed declare the security weakness of the scheme's state variables as tied down to only 32, related to the public IV.
- Apart from using random numbers from a popular source as seeds to the PRNG in their tests, the authors also change the fitness function and produce invalid results. The presented explanation given for the change is unsatisfactory.

A. The Inspired PRNG: Mersenne Twister

Our proposal is based on the Mersenne Twister (MT) [18]; which has properties that are feasible in low-cost tags. MT is a well-known algorithm, classified as a good PRNG [19]. A Mersenne prime is used as the period of the algorithm for fast generation of pseudo random numbers, free from long-term correlations. The algorithm is based on linear recurrences in $\boldsymbol{F}_2$ (finite field with two elements, 0 and 1) and arithmetic modulo 2^m operations [20]. Incidentally, binary recurrences and bitwise operations are easily implementable in resource stricken, low-cost tags. Briefly, MT is a specially twisted, generalized feedback shift register (TGFSR) that takes an incomplete array to realize a Mersenne prime, as its period. It uses an inversive-decimation method for testing the primitivity of a characteristic polynomial of linear recurrence with a computational complexity $O(p^2)$, where p is the degree of the polynomial.

Mathematical arguments show that MT is a special case of well equidistributed long-period linear (WELL) generators [20]. Omitting the details of the argument, MT has a long period of $2^{19937}-1$, with a 623 dimensional equidistribution up to 32-bit accuracy. In fact, MT has better equidistribution and "bit-mixing" properties than its predecessor PRNGs with equivalent period length and speed. Many variants of MT have been introduced for cryptographic security at better speeds [19-20].

The successful randomness test results of MT draw the attention of PRNG researchers. The steps of MT are also of interest to tag producers, because a tag can accommodate their simple bitwise operations. Looking at the steps closely, MT works in a recurring part and a tempering part. The simplified MT steps and the tempering stage are:

```
Step 0: Create bitmask for upper and lower bits,
Step 1: Initialize an x[i] array with nonzero seed values,
Step 2: Concatenate the upper bits of previous array x[i] with the lower bits of iterated array x[i+1],
Step 3: Calculate the next state array x[i],
Step 4: Carry out tempering as follows:
$y \leftarrow x[i]$
$y \leftarrow y \oplus (y >> u)$
$y \leftarrow y \oplus ((y << s)\ \&\ b)$
$y \leftarrow y \oplus ((y << t)\ \&\ c)$
$z \leftarrow y \oplus (y >> l)$
('<<' is a bitwise left shift, '>>' is a bitwise right shift, and '&' is a bitwise AND),
Step 5: Increment i by 1:
$i \leftarrow (i+1) \bmod n$, where n is the degree of recursion,
Step 6: Go to step 2, repeat until i equals n.
```

The first three steps, initializes and concatenates an array. Tempering is carried out in Step 4 to improve the distribution of the sequences generated from the recursions. Parameters *u*, *s*, *t*, *l* are tempering bit shifts, *b* and *c* are tempering bit-masks. The parameters are experimentally tested values for maximally-equidistributed generators [20].

The MT algorithm has some known disadvantages, which were eliminated in our work. Firstly, the initial state of MT has too many zeros, therefore the generated sequences also contain many zeros for more than 10000 generations. This is the reason of the problem indicated previously, in the fitness function of [3]. Our work removes this weakness by supplying non-zero, random initial inputs and by completely removing the matrix recurrences. For seeds chosen systematically as 0, 20, 30 a second weakness appears as correlated output sequences; which does not happen in our proposal. Finally, MT is not preferred for cryptographic purposes because it is easy to predict the next state if the present output is known.

To fix the above weaknesses many variations of MT have been proposed. One of the suggestions is to have the outputs of MT go through a function. For example, TRNs can be fed into a hash function [14]. It is clear that if the generator is initialized with uniform random bits, the probability of getting many zero bits or correlated output is quite small. Thus, seeding MT with TRNs and fine-tuning tempering parameters can improve the distribution of the generated PRNs [20]. This is the achievement of the present work, obtained without using a burdensome hash function.

III. The Proposed Scheme

Although the promising FERNS technique is the starting point of our proposal, any scheme qualifying as a TRNG can be the input stage of our proposed PRNG. Our design requires a TRN to be present in one of the registers, before the PRN generation starts. However, it is not difficult to meet this important pre-requisite. In both [12] and [21], Gen-2 compliant Wireless Identification Sensing Platform (WISP) [22] tags are used. WISP uses 16-bit Texas Instruments, reduced instruction set computer (RISC) architecture, MSP430 microcontrollers, which have 256/512 bytes of RAM. Both works use external tools to read the contents of the internal memory of the microcontroller. Our work however, involves the reading of the SRAM memory into one of the 12 registers of the microcontroller, during the initialization of the tag. Prior to answering the request of a reader, the microcontroller can read an unused memory location into one of the 12 registers, with the MOV instruction. Next, the value in the register is rotated and XORed with the next memory contents. After a number of recursions, the result is a non-zero TRN, in one of the registers. The probability of a zero TRN value is $1/2^{16}$.

```
x:=TRN          ;initialization
      z:=x            ;x is copied to z
x:=ROTl(x,z)    ;for improving the seed
y:=x            ;x is copied to y
z:=u            ;u is loaded into z
y:=ROTr(y,z)    ;MT tempering 1
x:=x⊕y          ;end of MT tempering 1
y:=x            ;x is copied to y
z:=s            ;s is loaded into z
y:=ROTl(y,z)    ;MT tempering 2
z:=b            ;b is loaded into z
y:=y AND z      ;MT tempering 2
x:=x⊕y          ;end of MT tempering 2
y:=x            ;x is copied to y
z:=t            ;t is loaded into z
y:=ROTl(y,z)    ;MT tempering 3
z:=c            ;c is loaded into z
y:=y AND z      ;MT tempering 3
x:=x⊕y          ;end of MT tempering 3
y:=x            ;x is copied to y
z:=l            ;l is loaded into z
y:=ROTr(y,z)    ;MT tempering 4
x:=x⊕y          ;end of MT tempering 4
y:=x            ;x is copied to y
z:=b            ;b is loaded into z
y:=y AND z      ;added tempering 5
z:=p            ;p is loaded into z
y:=ROTl(y,z)    ;added tempering 5
x:=x⊕y          ;end of added tempering 5
```

$u = 7F_H, s = 07_H, t = 1F_H, l = 3FFFF_H, p = 7F_H,$ $b = 9D2C5680_H$, $c = EFC60000_H$.

Figure 2. The proposed scheme

The proposed scheme is shown in Fig. 2. The deterministic, iterative part of MT is replaced with the TRNG seeding. The remaining operations are basically step 4 of Mersenne Twister, described in Section 2.1. The main challenge is to find the tempering parameters and mask values (*u*, *s*, *t*, *l*, *p*, *b*, *c*) that yield the best randomness results. The scheme consists of only simple bitwise XOR, AND and rotation (circular shift) operations, which replaces the hash function of previous works.

At the start, the TRN seed is already in the main register *x*. The value in *x* is copied to secondary register *z*, for preservation of the original TRN value until the first XOR operation. The first rotation to the left (ROTl) improves the entropy of the seed, as the TRNs are known to have low entropy. The ROTl (*x*, *z*) operation is a simple bitwise operation, which shifts *x* to the left, depending on the number of "1" values in *z*; where the most significant bit (MSB) is wrapped around to the least significant bit (LSB), through carry bit. The opposite is true for the rotate right (ROTr) operation, as well. Our scheme employs rotation operations, instead of shift operations, because rotation removes the misbalance introduced by the XOR operation between left-most and right-most bits [23]. In the next step, first the improved TRN in *x* is copied into y, then *u* is loaded into *z*. In the second rotation, as *u*'s hamming weight (number of "1" values in *u*) is 7, *y* is rotated 7 times to the right. The rotated value is XORed with the value in the main register *x*, ending the first tempering. In the next two steps, the value in the main register *x* is copied to *y*, and parameter *s* is loaded to *z*. In the next four steps, tempering 2 is carried out. The next six steps process the third tempering. Next step is MT tempering step 4, where the previously tempered value is rotated right by 18 times (*l*'s hamming weight) and the result is XORed with the unrotated value. Finally, tempering 5 is our contribution which is intended for increasing the randomness of the output.

Overall, the rotation operation executes permutation and XOR-AND operations provide substitution effect, on their operands. Thus, the input goes through a sequence of permutations and substitutions, as in modern hashing and encryption algorithms. In the steps labeled as MT tempering 1, 2, 3, 4, 5 the original TRN is transformed. After our additional tempering, the obtained PRN is in the main register, ready to be used in the authentication of the tag. Different number of rotation operations and directions against different *u*, *s*, *t*, *l*, *p*, *b* and *c* values have been tested. The scheme with the best results has been chosen. Since Mersenne Twister is explained abundantly in mathematical literature, no further mathematical discussion will be pursued, in present study.

The scheme has a sequential format which can be implemented as sequential code inside the tag, where the coefficients are given as immediate constant operands. This removes the need for a complex finite state machine (FSM), inside the tag. The registers and operators (XOR, AND, ROT) can also be used in the authentication algorithm; which in fact are not necessarily additional circuitries needed for producing a PRN. This is a true argument for the WISP tag's MSP430.

Running time for algorithms is modeled as a power law, given as $T(N) = a \times N^b$ (where a and b are constants, N is the input data size) [24]. The CPU, memory, cache, compiler, interpreter, garbage collector, operating system, network and other applications determine the constant a, in power law equation. Algorithm design and input data determines exponent b. In general, the measure of the running time involved in the algorithm designs is equal to the number of multiplications and recordings, because most of the work consists of multiplications and recording the numbers. In our algorithm presented in Fig. 2, most of the work comes from the 6 rotations. For the x:=ROTl (*x*, *z*) operation, the code snippet is given below:

```
int i;
for(i=0;i<N;i++)
  if(z[i]==1)
   ROTl (x,1) // Rotate left 1 time only
```

Running time for the above rotate function is O(N). Since there are 6 rotations in the proposed algorithm, the approximate model of our algorithm is ~6N and the order of growth is linear. Therefore, the proposed algorithm can be an effective tool to generate PRN for RFID tags.

Our design can be simulated and tested, easily. A set of numbers is fabricated as a file, to be used as seeds to our PRNG scheme. It is important that the numbers are manipulated, so that their overall entropy is low. The proposed design steps are coded in the order they appear in Fig. 2, as a PRNG sub-routine. Three registers are defined where all the calculations take place. Simulation consists of reading values from the fabricated input table, running them in the PRNG sub-routine and storing the outputs into another file. The output file contains the PRNs generated by our design, which is used as input to the randomness test suites. The code is available upon request from the authors.

IV. Performance, Testing and Evaluation of Results

The hardware efficiency of a proposed scheme - measured in length of data path, die area, clock cycles spent, consumed power, throughput etc. - is a good indicator of its performance. Our proposal uses 16-bit tag architecture to obtain a 32-bit PRN because of a number of reasons. The latest 32 or 64-bit, state of the art technologies for microprocessor production cannot be used in low-cost tag production. One proof of our choice is the aforementioned MSP430 microcontroller of the WISP family of tags. Besides, the common comparison ground is the 16-bit design results, provided by previous works. We assume that 16-bit PRNs of Gen-2 can be obtained from our 32-bit PRNs by either taking the lower 16 bits, or by XORing the higher 16 bits with the lower 16 bits, as in previous works [3, 10]. But it can be claimed that 32-bit, good quality PRNGs that pass the popular randomness tests are possible in tags, as shown in Section 4.A. The “access” and “kill” commands of Gen-2 are 32-bit and require multi steps to finish. The availability of our 32-bit RNs is an advantage that can simplify the commands. Moreover, only 32-bit randomness tests are accepted, by the community. According to Gen-2 specifications, the tag is expected to provide only a 16-bit PRNG whose period is much shorter than that of a 32-bit generator. The short period of a 16-bit PRN means reduced randomness that causes the security of Gen-2 to be classified as inadequate [3, 10].

The maximum number of gates and clock cycles allocated for security is a few thousands gates and 1800 clocks [7, 25]. These resources cannot be used for only generating a random number, because space and time must be left for other tasks and authentication steps. The above guidelines are commonly used in comparing the performance and randomness test results of similar works.

A. Performance Results

Our scheme (Fig. 2) requires only XOR, AND and circular shift operators. Table 1 shows the operation types used in the previous schemes against ours. Obviously, the multiplication and the finite state machine requirements of Akari-x [10] put overwhelming load on the tag. The Lamed scheme uses three simple operations like ours, but requires input, control and rotation units for iterative work. While Akari-x schemes require memory for an IV, Lamed [3] requires two 32-bit space. Ours requires one 32-bit and five 16-bit space. Memory requirement of each scheme is given in bits, in the final column of Table 1. All schemes’ memory space requirements can be met by WISP.

In order to estimate the die area of integrated circuits, independent of the used technology, gate equivalents (GEs) can be utilized [26]. One GE is equivalent to the area required by a two input NAND gate. The GE of each logic gate and thus the total GE for all bitwise operators are known and widely accepted [27]. The total GE required for our scheme is calculated to be 527 gates, as shown in Table 2. The GE values of our scheme and the declared values of previous works are shown in Table 3a. Our GE value is the lowest among the compared. The lowest GE, at the same time means lowest cost; because every 1000 GE adds $0.01 to the cost [29]. As an example for the calculations, one bit of a shifter uses one flip flop that costs 5.33 GE. For 16 bits, the total GE required for a shifter is 85.28. Our scheme is in the low-cost category, since it is well below 1000 gates [7]. It is important to indicate that present work’s total GE has been increased to accommodate the 3 registers. The heavy cost of hashing algorithms, like the 8,120 GE value of SHA-1 [29] has not been included in our comparisons.

Table I. Comparison of Operation Types and Memory used in Different PRNGS

Scheme	Operation Types Used	Iteration	Memory	Memory Usage (Bits)
Akari1A	SUM, OR, MULTIPLY, SHIFT	For loop, FSM	iv	96
Akari1B	SUM, OR, MULTIPLY, SHIFT	For loop, FSM	iv	96
Akari2A	SUM, OR, MULTIPLY, SHIFT, XOR	For loop, FSM	iv	96
Akari2B	SUM, OR, MULTIPLY, SHIFT, XOR	For loop, FSM	iv	96
Akari2C	SUM, OR, MULTIPLY, SHIFT, XOR	For loop, FSM	iv	96
Lamed	SUM, XOR, SHIFT	Control Units	iv, key	160
Ours	XOR, AND, SHIFT	Sequential	u, s, t, l, p	112

Table II. Gate Equivalents and the Total Gate Equivalents of our Proposal

Operator	Operation Types Used # Used	Logic	GE	16-bit Total
Register	3	FlipFlop	5.33	255.84
Shifter	1	FlipFlop	5.33	85.28
AND	1	Gates	1.33	21.28
XOR	1	Gates	2.67	42.72
Total				405.12
Control	1	Gates	30%	121.54
Grand Total				526.66

Table III. Comparison of our Performance Results with (a) Declared (Upper) and (b) Calculated Results of Previous Similar Works

Scheme	Area (GE)	Die Area (μm^2)	Delay Cycles	Complexity GE×Delay	Pow. Cons. (nW)	Throughput (Kbps)	Op. Types & Ctrl
Akari-1A	1018	3191	66	67,188	89.95	48.48	[2]C
Akari-1B	922	2892	450	414,900	95.71	7.11	[2]C
Akari-2A	1861	5837	51	94,911	109.88	31.37	[2]C
Akari-2B	1650	5173	290	478,500	135.81	5.50	[2]C
Akari-2C	1620	5081	530	858,600	126.02	3.01	[2]C
Lamed	1566	[1]	186	291,276	[1]	8.20	[2]C
Ours	527	1665	118	62,186	113.80	13.56	[3]S
Akari-1A	1018	3217	389	396,002	219.89	4.11	[2]C
Akari-1B	922	2914	389	358,658	199.15	4.11	[2]C
Akari-2A	1861	5881	333	619,713	401.98	4.81	[2]C
Akari-2B	1650	5214	333	549,450	356.40	4.81	[2]C
Akari-2C	1620	5119	333	539,460	349.92	4.81	[2]C
Lamed	1566	4949	186	291,276	338.26	8.20	[2]C
Ours	527	1665	118	62,186	113.80	13.56	[3]S

1: Not provided; 2: Complex; 3: Simple

The die area of a two-input NAND gate is given as 3.16 μm^2, in UMC90 nm semiconductor production technology [30]. Hence the die area in μm^2 of a bitwise operator can be obtained by multiplying the GE of the operator by 3.16. The die area of our design is $3.16 \times 527 = 1665\ \mu m^2$. The die area values of the Akari-x schemes have been quoted (Table 3a) from the 16-bit architecture columns of the authors' own tables [31]. The values are in agreement with the equation (1 GE = 3.16 µm^2), but are not exact. Lamed's die area has not been declared; therefore, it had to be calculated.

The timing metric, on the other hand, is the clock cycles used for performing a specific function. A 16-bit ALU requires one clock cycle to finish a 16-bit register copy, AND or XOR operation. But, 16 clocks are needed for rotation since it requires the testing of the contents of a 16-bit register bit by bit and circularly shifting the contents of another register, depending on the outcome of the test. To calculate the number of clock cycles spent for obtaining a random number in our scheme, the total of copying/loading operations of registers, XOR/AND operations and the rotation (ROT) operations in Fig. 2 are counted. The TRN is already in one of the registers before the PRNG starts. The register copying and XOR/AND operations can be completed in a single cycle. Assuming the constants *u*, *s*, *t*, *l*, *p*, *b*, *c* are immediately after the instructions (immediate addressing), loading them into a register (e.g. MOV Rx, *u*) also takes one clock cycle. ROT operations test the bits of one register and shift the contents of another if the tested bit is a "1"; wrapping around the overflowing bit through carry. Therefore, a clock cycle is spent for each bit test. Thus, clock cycles equal to the number of architectural bit length (16 in our case) are consumed for each rotation. In total, there are 6 register copying, 8 register loading, 5 XOR, 3 AND and 6 ROT operations. Hence, the total clock cycles consumed is 6 + 8 + 5 + 3 + 6 × 16 = 118 clocks. When checked against the declared limit 1800, the proposed 118 clock cycles scheme is definitely in the ultra-light category [7]. This value is shown in Table 3a, together with the declared values of previous works. However, if the same clock cycle calculation method is applied to the previous works, their declared values appear optimistic. For fair comparison conditions, a second table of calculated performance values is necessary. To prove the point, examining the Akari-1 design suffices. Akari-1 has an iteration loop of 64 times. Inside the iteration, there are 2 single shift and 3 addition (modulo 2) operations. Thus, the iteration costs 5 × 64 = 320 clock cycles. Outside the iteration, there are 2 addition, 2 OR, 1 register copying and 2 multiplication operations. Assuming the traditional shift left and add method for multiplication, each multiplication costs 2 × architectural bit length; for 16-bit Akari-1, single multiplication costs 2 × 16 = 32 clocks. Thus, outside the iteration there are 2 + 2 + 1 + 64 = 69 clock cycles. Overall, Akari-1 has a clock cycle delay of 389 clocks, for producing the lower half (16 bits) of its PRN. The Akari-2 design has two 24 round iteration loops. Outside iteration, Akari-2 also uses 69 clocks; but first iteration costs 5 × 24 and the second 6 × 24 clocks. Overall, Akari-2 has a 333 clock cycles delay. Authors of Akari-x have a superseding work with new results for their previous work [31]. Although there are changes in the architectural bit lengths, the declared clock cycles are the same as in the previous work. On the other hand; although Lamed has a parallel architecture, it still has a high declared clock cycle delay of 186. The calculated results of previous work are compared with our results in Table 3b. The area-delay product (GE × clock cycles) is defined as hardware complexity (complexity) of an operator [29]. The complexity of previous works is compared with ours in Tables 3 and 4. For Akari-x works, the GE values of 16-bit architectures have been accepted in Table 3a. From both Table 3a and 3b, it follows that our scheme has the lowest complexity, leaving enough space for other security functions as well. The area-delay product for hashing functions is very high; around 8 to 10 million GE clocks for 32-bit architectures [25, 29]. Their complexity values clearly indicate that encryption and hashing schemes are not in the ultra-light category. Therefore, the work of [12] has not been included in comparisons.

The power consumption is critical in remotely energized RFID tags, therefore the power required for generating a random number should be minimal. It is natural to observe that power consumption is proportional to the number of gates; because the more the number of the gates to be powered, the more supply energy is required [28].

$$P = P_{0\rightarrow 1} \mathrm{C_L} \mathrm{V_{DD}^2} \mathrm{f_{clk}} \quad (1)$$

(1) is used to estimate the power consumed by a hardware design and denotes the dynamic power dissipation by estimating the power loss according to the capacitance charge and discharge [33]. Logic state transition of the gates from 0 to 1 or 1 to 0 in one clock cycle is denoted by $\mathrm{p}_{0\rightarrow 1}$. $\mathrm{C_L}$ represents the load capacitance and it is commonly approximated to 3×10^{-15} F. The drain supply voltage ($\mathrm{V_{DD}}$) in our 90 nm design is 1.2 V. The system clock frequency ($\mathrm{f_{clk}}$) is typically 100 kHz, in RFID architectures. The above values are the same in Akari-x and Lamed, making equal conditions of comparison. The total GE of our proposal is 527, since approximately half of these gates are switched in 1 clock cycle, $\mathrm{p}_{0\rightarrow 1}$ value is 263.5. It follows that the P value of our design is 113.8 nW. The power consumption of Akari-x family PRNGs has been quoted in Table 3a, using the new publication [31]. The declared Akari-x consumption values are also estimations, obtained by using a software design tool. Lamed's power consumption was not provided; therefore, it was calculated by using (1). The calculated power consumption values are given in Table 3b. If the number of switched gates is considered as a true indication of the power consumed, then our design has the lowest power consumption, in Table 3b. Our design's power consumption is better than the Akari-2x family of designs, in Table 3a.

The last metric to consider is the throughput of the designed hardware. Throughput is a measure of the number of bits output per second, by the designed scheme. This metric is not as critical as the above considered parameters in low-cost devices like RFID tags [25-27], [29]. Nevertheless, low throughput values of a design show that low power consumption and die area were aimed, instead of high output per second. Throughput can be calculated by (output data size per second) / (the number of clock cycles it takes to output each data × time of each clock cycle). The data size of the compared architectures is 16 bits. The period (1/working frequency) of each clock is 1/100 kHz. Hence, the throughput in kilobits per second (kbps) of a typical 16-bit RFID tag, working at 100 kHz system clock frequency, is simplified to (2). The *nCC* value is the number of clock cycles consumed for outputting a single PRN, by the designed hardware.

$$T = \frac{1600}{nCC} \qquad (2)$$

The throughput of our design is 1600/118 = 13.56 kbps. When compared with the results quoted from the other proposals in Table 3a, it is observed that our design's throughput is positioned in the middle of the compared designs. Among the calculated values in Table 3b, the throughput of our design is the highest, compared to the other 16-bit PRNG versions. Having the highest throughput, yet the lowest GE and power consumption is a solid proof that our scheme is indeed a high-throughput, ultra-lightweight design.

Taking universally accepted performance estimations into consideration, Table 3b shows that the overall performance of our design is the best among the enlisted PRNGs. The 32-bit and parallel architectures of the other designs are not very realistic for RFID tags and their declared performance values are inconsistent. For example, the Akari-1x designs which have lower power consumptions than our design in Table 3a, have GE values two times higher than ours. Akari-2B, has a GE almost 3 times bigger than our design, obviously contradicting condition (1). Our clock delay estimation of 16-bit versions of Akari-x is not in contradiction with the high number of iterations and the multiplication operations, in the design. All of the hardware architectural arguments indicate that our performance results are realistic and not optimistic for 16-bit PRNG designs.

B. Verilog & WISP Implementations

```
module prng;

reg[31:0]  x;
reg[31:0]  z, y;
parameter u = 32'h0000_007F, s = 32'h0000_0007,
t = 32'h0000_001F, l = 32'h0003_FFFF, p=
32'h0000_007F, b=32'h9D2C_5680;
parameter c=32'hEFC6_0000;
integer i;

initial begin
x = 32'hDEAD_BEEF;
z = x;
for (i=0;i<=31;i=i+1)
begin: BLOCK_1
        if(z[i]) x = {x[30:0],x[31]};
end
y = x;
z = u;
for (i=0;i<=31;i=i+1)
begin: BLOCK_2
        if(z[i]) y = {y[0],y[31:1]};
end
x = x ^ y;
y = x;
z = s;
for (i=0;i<=31;i=i+1)
begin: BLOCK_3
        if(z[i]) y = {y[30:0],y[31]};
end
z = b;
y = y & z;
x = x ^ y;
```

Figure 3. Sample Verilog implementation code of proposed PRNG

The popular Verilog hardware description language (HDL) has been used to model the proposed PRNG system. Verilog syntax is very similar to the C programming language syntax. Fig. 3 shows the PRNG code snippet that was simulated using the Icarus Verilog simulation engine. The code is a proof of the simplicity of our design.

```
uint32_t one = 1;
// Returns ith bit of an integer
uint32_t extractBit(uint32_t value, int pos)
{
 return((value & (one<<pos)) >> pos);
}

// Rotates left "n" bits
uint32_t rotl(uint32_t value, int n) {
  return (value << n) | (value >>
(sizeof(value)*8 - n));
}
// Rotates right "n" bits
uint32_t rotr(uint32_t value, int n) {
  return (value >> n) | (value <<
(sizeof(value)*8 - n));
}

int main( void )
{

 uint32_t x, y, z;
 uint32_t u= 127, s=7;
 uint32_t t= 31, l= 262143;
 uint32_t p= 127, b= 2636928640;
 uint16_t i;
 uint32_t c = 4022730752;

 x = 3735928559;
 z = x;

 for (i=32;i>0;i--){
   x=extractBit(z,32-i)?rotl(x,1):x;
                     }

 y = x;
 z = u;

 for (i=32;i>0;i--){
   y=extractBit(z,32-i)?rotr(y,1):y;
  }

 x = x ^ y;
 y = x;
 z = s;

 for (i=32;i>0;i--){
   y=extractBit(z,32-i)?rotl(y,1):y;
  }

 z = b;
 y = y & z;
 x = x ^ y;
```

Figure 4. Sample WISP implementation code of the proposed PRNG

The proposed scheme has also been implemented on WISP to prove that the design requires resources that can be met on real 16-bit tag platforms. The 16-bit, programmable, passive, UHF, WISP version 5.0, RFID tag is a good testing hardware platform. WISP has a MSP430FR5969 low power microcontroller that can easily be programmed through a simple programming interface. Part of the WISP implementation code corresponding to the Verilog implementation code of Fig. 3 is given in Fig. 4. The similarity in the syntax of the two implementations is obvious. Both implementations verify the hardware design shown in Fig. 5.

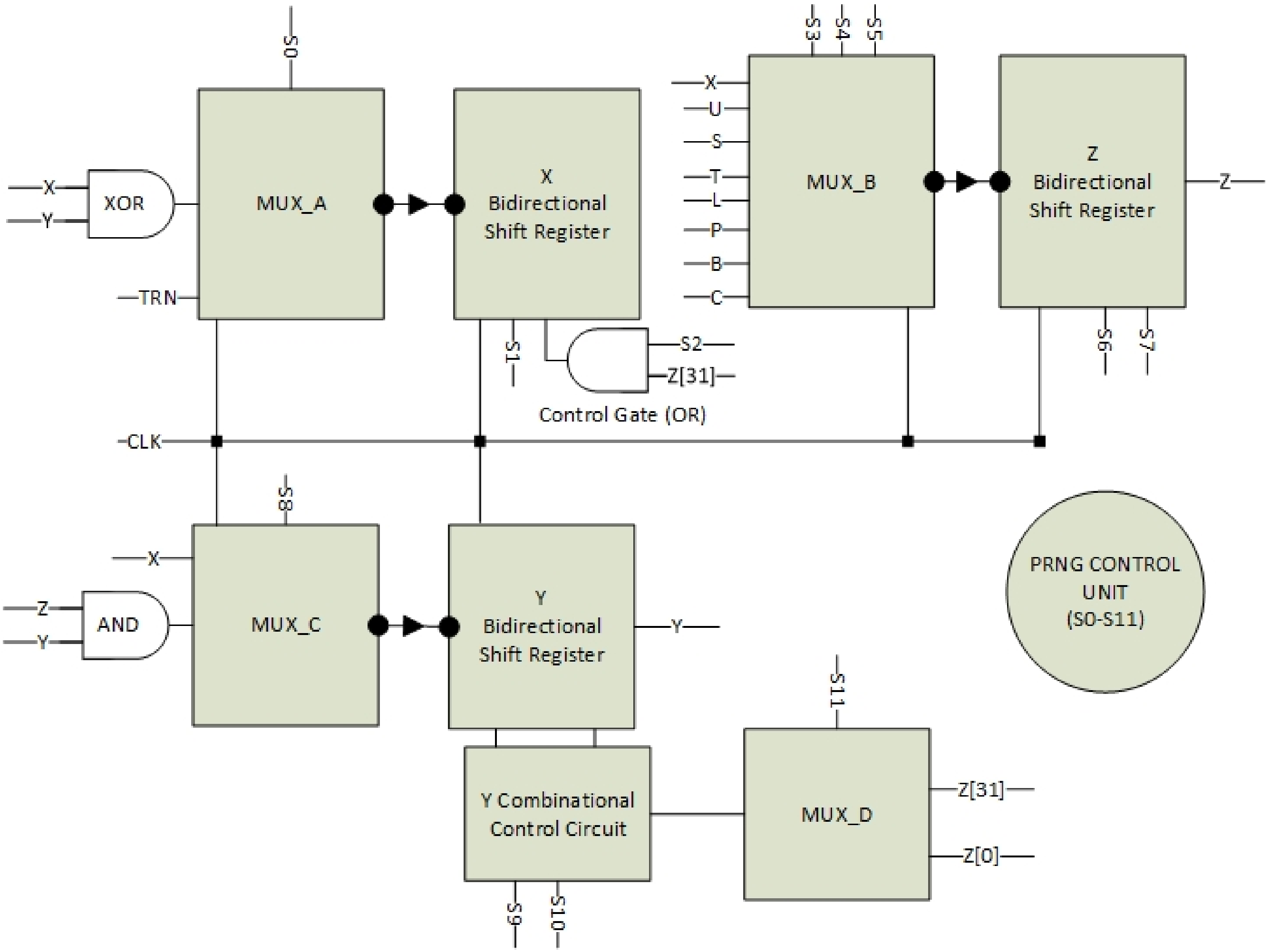

Figure 5. Hardware implementation of the proposed PRNG scheme

C. Hardware Implementation

The block diagram of a hardware implementation of the proposed PRNG scheme of Fig. 2 is given in Fig. 5. The tempering constants *u*, *s*, *t*, *l*, *p*, *b*, and *c* are shown as distinct inputs of multiplexer MUX_B, timed by the PRNG Control Unit through the S3-S4-S5 control path. In fact, the PRNG Control Unit generates the necessary control signals S0 through S11 to control the multiplexers, bidirectional shift registers and other control circuits of the proposed scheme. For example, when S0=0, S1=1 and S2=1, TRN is loaded into Register X (x:=TRN step of Fig. 2). In the next step when S3=0, S4=0, S5=0, MUX_B selects input X. When S6=1 and S7=1, X is loaded into Register Z (z:=x step of Fig. 2). The combinational control circuit for Register Y and multiplexer MUX_D are controlled via S9-S10 and S11 respectively, to decide the direction of rotation of the bidirectional rotational shift Register Y, as either left or right direction. For example, when S11=0 and the tested MSB Z[31]=1, Register Y is one bit left shifted and the MSB of Register Y is rotated to LSB. Hence, the shift register is used as a rotational register. When S11=0 and Z[31]=0, then there is no change in Register Y (Hold). When S11=1 and Z[31]=1, then Register Y is one bit right shifted and the LSB of Register Y is rotated to MSB. The direction of the rotation of Register X is controlled in a similar method, using control path S1, S2 and the tested Z[31] bit.

D. Testing Results

Starting with the three requirements of generating random numbers given in Gen-2 (Section 1.2), it can be observed that they are met by our proposed scheme. The results satisfying the Gen-2 requirements can be summarized, as follows. The first requirement is satisfied in the constraint $0.923/2^{16} < P(RN16 = j) < 1.071/2^{16}$. The second requirement is satisfied with a result value of 0.04. The third requirement is satisfied with a 0.000008 serial correlation result, which ensures that the probability of predictability is not greater than 0.025% [3]. The inputs, detailed generated random number files and the results of the calculations can be found on the web page at http://srg.cs.deu.edu.tr/ publications/2012/prng/.

In the randomness tests, two types of inputs were used to reach the best scheme. At first, an input-set from http://random.org was used to identify the schemes which fail to produce good random numbers. Then, a second set of inputs with low entropy (0.00) was used to equate the entropy of the inputs to that of work [12]. But, many schemes that performed well with RN inputs produced poor test results with low entropy inputs, so they were dropped too. Only those schemes that passed the randomness tests with low entropy inputs were selected and further improved, finally to reach the best solution. It should not be missed that Lamed and Akari-X use RNs from http://random.org inputs for obtaining RNs, but RN seeds are simply not readily available, in tags.

TABLE IV. NIST TEST VERSION 2.1, "PROPORTION" RESULTS COMPARED WITH PREVIOUS

Test Name	Lamed	Ours
Frequency	0.98	1.00
Block-frequency	0.98	0.99
Cumulative-sums	0.98,0.98	1.00,1.00
Runs	1.00	1.00
Longest-run	1.00	0.98
Rank	0.98	0.00*
Fft	0.99	0.99
Overlapping-Templates	0.98	1.00
Universal	0.96	1.00*
Apen	0.99	0.98
Serial	0.97, 1.00	1.00,0.99
Linear-complexity	0.99	0.99
Random-excursions	0.97, 0.98, 1.00, 0.97, 1.00, 1.00, 0.97, 1.00	1.00, 1.00, 1.00, 1.00, 1.00, 0.92*, 1.00,1.00
Random-excursions-variant	1.00, 1.00, 1.00, 0.98, 1.00, 1.00, 1.00, 1.00, 1.00, 1.00, 1.00, 1.00, 1.00, 0.98, 0.97, 0.98, 1.00, 0.97	1.00, 1.00, 1.00, 1.00, 1.00, 1.00, 1.00, 1.00, 1.00, 1.00, 1.00, 1.00, 1.00, 1.00, 1.00, 1.00, 1.00, 1.00, 1.00, 1.00, 1.00
Test Name	**Akari 1A/B**	**Akari 2A/B/C**
Frequency	0.98	0.98
Block-frequency	0.99	0.99
Cumulative-sums	0.97, 0.97	0.99, 0.97
Runs	0.99	0.99
Longest-run	1.00	0.99
Rank	0.99	0.99
Fft	0.99	0.99
Overlapping-Templates	1.00	0.99
Universal	0.99	0.97
Apen	1.00	0.99
Serial	1.00, 0.99	0.99, 0.97
Linear-complexity	1.00	1.00
Random-excursions	1.00, 0.99, 1.00, 0.97, 1.00, 0.99, 1.00, 1.00	1.00, 0.98, 1.00, 1.00, 0.99, 0.99, 0.98, 1.00
Random-excursions-variant	1.00, 0.99, 0.99, 0.99, 0.99, 0.99, 0.97, 0.99, 1.00, 1.00, 1.00, 0.99, 1.00, 1.00, 1.00, 1.00, 0.99, 0.99	1.00, 1.00, 1.00, 1.00, 1.00, 0.99, 0.99, 1.00, 0.99, 0.99, 1.00, 0.99, 0.99, 0.99, 0.98, 0.98, 0.98, 0.99

1: Not provided; 2: Complex; 3: Simple

The ENT [34] and Diehard [34] are preliminary, relaxed tests acting as indicative values, prior to running the NIST suite [36]. To summarize the posted results briefly, the random number outputs of our design pass the ENT and Diehard tests with satisfactory results. Normally, the random numbers that pass the NIST suite can easily pass the ENT and Diehard tests.

To expose the differences in designed schemes, one has to consider the testing results of the stricter NIST suite of tests. The NIST tests' output results called "p-values" and "proportion" values are expected to be greater than 0.01 and 0.96, respectively. The "proportion" result is the proportion of the binary sequences that passed the test (p-values > 0.01). Any undesirable result is marked with a "*" next to the proportion value. It is acceptable for a scheme to fail a few tests out of 188 tests; i.e. a scheme failing one or two individual tests cannot be considered as not passing the overall NIST test [35]. In our work, to test the scheme given in Fig. 2 the NIST test version 2.1. has been used. The NIST test results are very long and detailed reports. Therefore, our full results are posted on http://srg.cs.deu.edu.tr/publications/2012/prng/. The NIST test results of [10] are on http://www.lightweightcryptography.com/research/akari/akari.html. Only the proportion results are given, in publication [3]. Our proportion value results are summarized in Table 4.

Only two tests individually fail the criteria. But the NIST tests are known to be the strictest tests intended for computers and not for tags. Failing one or two tests does not prove our scheme to be unsuitable for tags. Work [35] openly states that "It is acceptable for a few individual tests to fail". To defend the argument, first our Universal test result is discussed. In NIST's official document, the technical description of Universal test is given as a compression-type test, where a significantly compressible sequence is considered to be non-random [36]. Although our Universal test result fails, the ENT test results posted on our web site shows that our proposed scheme's output compression rate is 0%. The conflicting results cast doubt whether our output sequence is easily compressible, or not.

E. Limitations

Failure in Binary Matrix Rank Test (Rank for short, in Table IV) is regarded as an indication for non-randomness [37]. Rank test "constructs binary matrices from the analyzed data and checks for linear dependence among the rows or columns of the constructed matrices". Hence, failure to pass the Rank test can be accepted as evidence of non-randomness due to linearity problems. This is the case for the proposed PRNG scheme of Fig. 2, as well. However, the tested output values of the scheme are the result of very low entropy inputs (entropy of TRN = 0.00 in Fig. 2). In other words, the input values already have a huge linearity vulnerability and fail the Rank test, badly. Our proposed scheme passes the Rank test if input values from random.org are used, as in the previous works. But, such a test or its results do not present a valid argument, because it would mean to defend obtaining random numbers from already proven random numbers. Even the null function can pass the Rank test with proven random number inputs.

Hence, there is a critical input entropy threshold when the input values push our proposed scheme to pass the Rank test. Although an entropy threshold of 0.20 is a good starting point where our proposed scheme passes the Rank test, laboratory tests have not revealed an exact threshold for input values. This is a limitation which can be stated as "the higher the random distribution of the input TRN values, the lower the linearity vulnerability of the proposed PRNG scheme in Fig. 2". In other words, as the seeding of the proposed PRNG scheme improves, the statistical linearity vulnerability decreases.

V. CONCLUSION

A new random number generator that is feasible in low-cost RFID tags has been presented. The performance results of the present proposal are the best in terms of simplicity of design, power consumption, die area, cost and complexity. The results indicate that the proposed scheme does not exceed the resource limits of the ultra-light tags. The presented scheme takes low-entropy TRNs as seeds, without requiring special circuitry, and produces random numbers that passes well-known randomness test suites. This is a critical superiority over previous works, which use random numbers as inputs (not available in RFID tags) to produce random numbers. Our proposal's performance results and the favorable randomness test results are supported by the utilized universally accepted measurement methods. Our scheme is available now, until a hashing or encryption algorithm is offered at the right cost, for low-cost tags.

Future work involves the design and implementation of the proposed scheme in an integrated circuit prototype. Theoretical work is encouraging for the hardware design, because the scheme is sequential and ultra-light.

REFERENCES

[1] C. M. Robert, "Radio frequency identification," Computers and Security, vol. 25, pp. 18–26, 2006. doi:10.1016/j.cose.2005.12.003.

[2] R. Das, P. Havrop, "RFID forecasts, players and opportunities 2011-2021," IDTechEX, 2011. [Online]. Available: http://www.idtechex.com/research/reports/rfid_forecasts_players_and_opportunities_ 2011_2021_000250.asp.

[3] P. Peris-Lopez, J. C. Hernandez-Castro, J. M. Esteves-Tapiador, A. Ribagorda, "LAMED a PRNG for EPC Class-1 Generation-2 RFID Specification," Computer Standards & Interfaces, vol. 31, pp. 88-97, 2009. [Online]. Available:http://dx.doi.org/10.1016/j.csi.2007.11.013.

[4] A. Manzalini, et al. "Self-optimized cognitive network of networks," The Computer Journal, vol. 54, pp. 189-195, 2011. doi:10.1093/comjnl/bxq032.

[5] H. Y. Chien, "SASI: A New Ultralightweight RFID authentication protocol providing strong authentication and strong integrity," Trans. on Dependable and Secure Computing, vol. 4, p. 337–340, 2007. doi:10.1109/TDSC.2007.70226.

[6] P. Peris-Lopez, J. C. Hernandez-Castro, J. M. Esteves-Tapiador, A. Ribagorda, "An ultra-light authentication protocol resistant to passive attacks under the Gen-2 specification," J. of Information Science and Engineering, vol. 25, pp. 33-57, 2009.

[7] J. H. Kong, L. M. Ang, K. P. Seng, "A comprehensive survey of modern symmetric cryptographic solutions for resource constrained environments," J. of Network and Computer Applications, vol. 49, pp. 15-50, 2015. doi:10.1016/j.jnca. 2014.09.006.

[8] ISO/IEC 18000-6:2010, 2010. [Online]. Available: http://www.iso.org/iso/iso_catalogue/catalogue_tc/catalogue_ detail.htm?csnumber= 46149.

[9] Class-1 generation 2 UHF air interface protocol standard "Gen-2", Version 2.0.0, 2013. [Online]. Available: http://www.gs1.org/sites/default/files/docs/uhfc1g2/uhfc1g2_2_0_0_standard_ 20131101.pdf.

[10] H. Martin, et al. "AKARI-x: A pseudorandom number generator for secure lightweight systems," in Proc. 17th Int. On-Line Testing Symposium (IOLTS), pp. 228-233, 2011.

[11] M. Park, J. C. Rodgers, D. P. Lathrop, "True random number generation using CMOS Boolean chaotic oscillator," Microelectronics J., vol. 46, pp. 1364-1370, 2015. doi:10.1016/j.mejo.2015.09.015.

[12] D. E. Holcomb, W. P. Burleson, K. Fu, "Power-Up SRAM state as an identifying fingerprint and source of true random numbers," Transactions on Computers, vol. 58, pp. 1198-1210, 2009. doi:10.1109/TC.2008.212.

[13] J. M. Segui, J. G. Alfaro, J. H. Joancomarti, "Analysis and improvement of a pseudorandom number generator for EPC Gen2 tag," in Proc. Financial Cryptography and Data Security 2010 Workshops, pp. 34-46, 2010.

[14] J. Chen, A. Miyaji, H. Sato, C. Su, "Improved lightweight pseudo-random number generators for the low-cost RFID tags," in Proc. IEEE Trustcom/BigDataSE/ISPA, vol. 1, pp. 17-24, 2015. doi:10.1109/Trustcom.2015.352.

[15] P. Z. Wieczorek, "Lightweight TRNG based on multiphase timing of bistables," IEEE Transactions on Circuits and Systems I, vol. 63, pp.1043-1054, 2016. doi:10.1109/ TCSI.2016.2555248.

[16] A. J. Menenez, P. C. Oorschot, S. A. Vanstone, Pseudorandom bits and sequences. Handbook of Applied Cryptography CRC Press, pp. 169-187, 1996.

[17] B. Alomair, L. Lazos, R. Poovendran. "Passive attacks on a class of authentication protocols for RFID," in Proc. Int. Conf. on Information Security and Cryptology – ICISC'07, pp. 102-115, 2007.

[18] M. Matsumoto, T. Nishimura, "Mersenne twister: a 623-dimensionally equidistributed uniform pseudo-random number generator," Transactions on Modeling and Computer Simulation, vol. 8, pp. 3–30, 1998. doi:10.1145/ 272991.272995.

[19] M. Matsumoto, et al. Cryptographic Mersenne twister and Fubuki stream/block cipher, 2005. [Online]. Available: http://eprint.iacr.org/2005/165.

[20] F. Panneton, P. L'Ecuyer, M. Matsumoto, "Mersenne twister: improved long-period generators based on linear recurrences modulo 2," Transactions on Mathematical Software, vol. 32, pp. 1–16, 2006. doi:10.1145/1132973.1132974.

[21] N. Saxena, J. Voris, "Data remanence effects on memory-based entropy collection for RFID systems," Int. J. of Information Security, vol. 10, pp. 213-222, 2011. doi:10.1007/s10207-011-0139-0.

[22] A. P. Sample, et al. "Design of an RFID-based battery-free programmable sensing platform," IEEE Transactions on Instrumentation and Measurement vol. 57, pp. 2608-2615, 2008. doi:10.1109/TIM.2008.925019.

[23] D. Khovratovich, I. Nikolic, "Rotational cryptanalysis of ARX," Fast Software Encryption, pp. 333-346, 2010.

[24] R. Sedgewick. Algorithms in C, Parts 1-5 (Bundle): Fundamentals, Data Structures, Sorting, Searching, and Graph Algorithms, 3/e, Addison-Wesley Professional, pp: 55-86, 2002.

[25] M. Feldhofer, S. Dominikus, J. Wolkerstorfer, "Strong authentication for RFID systems using the AES algorithm," in Proc. Cryptographic Hardware and Embedded Systems-CHES 2004, pp. 357-370, 2004.

[26] A. Moradi, A. Poschmann, "Lightweight cryptography and DPA countermeasures: a survey," in Proc. 14th Int. Conf. on Financial Cryptography and Data Security, pp. 68-79, 2010. doi:10.1007/978-3-642-14992-4_7.

[27] C. Paar, A. Poschmann, M. J. B. Robshaw, "New designs in lightweight symmetric encryption," RFID Security: Techniques, Protocols and System-on-Chip Design, pp. 349-371, 2009.

[28] P. Peris-Lopez, P. T. Lim, T. Li, "Providing stronger authentication at a low-cost to RFID tags operating under the EPCglobal framework," in Proc. Embedded and Ubiquitous Computing Conference, pp. 159-167, 2008.

[29] M. Feldhofer, J. Wolkerstorfer, "Hardware implementation of symmetric algorithms for RFID security," RFID Security: Techniques, Protocols and System-on-Chip Design, vol. 3, pp. 373-415, 2009.

[30] H. Martin, P. Periz-Lopez, J. E. Tapiador, E. San Millan, "An estimator for the ASIC footprint area of lightweight cryptographic algorithms," IEEE Trans. on Industrial Informatics, vol. 10, pp. 1216-1225, 2014. [Online]. Available: http://dx.doi.org/10.1109/TII.2013. 2288576.

[31] H. Martin, E. San Millan, P. Periz-Lopez, J. E. Tapiador, "Efficient ASIC implementation and analysis of two EPC-C1G2 RFID authentication protocols," Sensors, vol. 13, pp. 3537-3547, 2013. [Online]. Available: http://dx.doi.org/10.1109/JSEN.2013.2270404.

[32] J. Melia-Segui, J. Garcia-Alfaro, J. Herrera-Joancomarti, "Multiple-polynomial LFSR based pseudorandom number generator for EPC Gen2 RFID tags," in Proc. 37th Annual Conference on IEEE Industrial Electronics Society, pp. 3820-3825, 2011. doi:10.1109/IECON.2011.6119932.

[33] J. Walker. Randomness battery, 1998. [Online]. Available: http://www.fourmilab.ch/random/.

[34] G. Marsaglia, T. Marsaglia, "Random number CDROM including the DIEHARD battery of tests of randomness, Diehard version 1," 1996. [Online]. Available: http://stat.fsu.edu/pub/diehard. Diehard version 2 2003. [Online]. Available: http://i.cs.hku.hk/~diehard/.

[35] P. Kohlbrenner, K. Gaj, "An embedded true random number generator for fpgas," Proc. 12th International symposium on Field programmable gate arrays, pp. 71-78, 2004.

[36] A. Rukhin, et al. A statistical test suite for random and pseudorandom number generators for cryptographic applications, 2010. [Online]. Available: http://csrc.nist.gov/rng/.

[37] J. Melià Seguí, "Lightweight PRNG for low-cost passive RFID security improvement," Doctoral thesis, Universitat Oberta de Catalunya, 2011. [Online]. Available: http://openaccess.uoc.edu/ webapps/o2/handle/10609/29341.