An Efficient Lightweight Stream Cipher Algorithm for Wireless Networks

Soumyadev Maity
ACM Unit
Indian Statistical Institute
Kolkata, India
Email: soumya@ieee.org

Koushik Sinha
Dept. of Computer Science
Southern Illinois University
Carbondale, IL, USA
Email: koushik.sinha@cs.siu.edu

Bhabani P. Sinha
ACM Unit
Indian Statistical Institute
Kolkata, India
Email: bhabani@isical.ac.in

Abstract*—A significant number of applications in mobile transactions and wireless sensor networks (WSNs) are characterized by short duration sessions. Stream ciphers are a popular choice for ensuring the security of data communication sessions in such applications. In this paper, we propose a lightweight variant of the popular RC4 algorithm that is highly suitable for energy and computational resource constrained devices in wireless networks, as compared to RC4 and its variants (e.g., HC-128), Grain-128, etc., while being secure enough for a large class of short duration wireless application scenarios. We replace the PRGA keystream generation algorithm of RC4 by our proposed *Lightweight Pseudo-Random Generation Algorithm* (LPRGA). The test results for LPRGA on the NIST statistical test suite show that the output keystream generated by LPRGA is as random as the output generated by the PRGA of RC4, for keystream sequence lengths smaller than 30,000 bits - a requirement easily satisfied in nearly all mobile transactions, WSNs and ad hoc networks. The structural test results also prove the security strength of the proposed LPRGA. Furthermore, measurements of the period lengths of the LPRGA show that the proposed algorithm has a much larger period than the PRGA of RC4. We have also proved analytically the unpredictability of the proposed algorithm. Finally, for resource constrained wireless devices, we present the outline of a high performance hardware implementation with a significantly lower cost as compared to the conventional hardware implementations of RC4 for wireless networks. Our proposed hardware implementation is capable of throughput as high as one cipher byte per cycle of a 1 GHz clock using 65 nm fabrication technology.

***Index Terms*—Stream cipher, lightweight cryptography, RC4, hardware implementation, wireless communication, mobile transactions, wireless sensor networks, ad hoc networks.**

I. Introduction

The computing devices used in a large class of wireless communication networks, such as smartphones, wireless sensor networks (WSNs), body area networks (BANs), mobile ad hoc networks (MANETs), vehicular ad hoc networks (VANETs), Internet of Things (IoT) etc., are small and resource constrained. Stream ciphers algorithms have been a popular choice to ensure the security of data communication sessions in such networks [6]–[11], [14]. Stream ciphers can be broadly classified into two types, depending on the platform most suited to their implementation - namely, software stream ciphers and hardware stream ciphers. Among the stream ciphers, RC4 [5] is one of the widely used stream ciphers that is mostly implemented in software, even though hardware implementations of RC4 are much more efficient in terms of keystream throughput, than their software implementations [1]. However, the cost, in terms of the NAND gate equivalent area requirement, associated with the hardware implementation of a cryptographic algorithm is usually prohibitively expensive for the resource constrained computing devices of wireless networks. RC4 is used in network protocols such as SSL, TLS, WEP, and WPA. While other more efficient and secure stream ciphers have been discovered after RC4, it is still the most popular stream cipher algorithm due to its simplicity, ease of implementation, and speed. Notwithstanding several cryptanalysis attempts on RC4 [15], [16], the cipher remains secure, if used appropriately.

Although the energy and computational overhead for a simpler cryptographic algorithm is usually lower, the security, in terms of effort required by an adversary to break the encryptions generated by the algorithm, is also less. However, in wireless networks, such as cellular networks, WSNs, BANs or VANETs, most communication sessions are short and encryption keys are renewed in each session - making the task for an adversary more difficult. Hence, in such networks, satisfactory level of security could be achieved even with a relatively simpler cryptographic algorithm. This motivated us to design a simple stream cipher which is lightweight for hardware implementation in small computing devices, yet efficient in terms of keystream generation throughput, and secure enough for use in short term wireless communication scenarios.

A. RC4 Stream Cipher

In 1987, Ron Rivest designed the RC4 stream cipher for RSA Data Security. RC4 uses an S-box S which is an array of length N (typically, $N = 2^n$, where n is usually 8). Each location of S stores a value between 0 and $N-1$. An l byte secret key k (typically, $5 \leq l \leq 16$) is used to scramble a permutation of $\{0, 1, ..., N-1\}$ on S. An array K of length N is used to hold the main key, where the secret key k is repeated as $K[x] = k[x \text{ mod } l]$, for $0 \leq x \leq N-1$.

The two main constituent algorithms of RC4 are *i)* the Key Scheduling Algorithm (KSA) and *ii)* the Pseudo-Random Generation Algorithm (PRGA). Objective of the KSA algorithm (Algorithm 1) is to generate a pseudo-random permutation S

of $\{0, 1, ..., N-1\}$ using the key K. The PRGA (Algorithm 2) generates a sequence of pseudo-random keystream bytes using the pseudo-random permutation produced by KSA. All of the arithmetic additions used in the context of RC4 are 'addition modulo N', in general. The bytes of the output keystream Z, produced by PRGA, are XOR-ed with the bytes of a plaintext message M to generate the ciphertext message C at the sender end ($C = M \oplus Z$). At the receiver end, the ciphertext message C is XOR-ed back with the keystream Z in order to get back the plaintext message M ($M = C \oplus Z$).

B. Our Contribution

In this paper, we propose a novel stream cipher algorithm which is as efficient as RC4, but with significantly lower computational overhead cost as compared to the conventional stream cipher algorithms like RC4, HC-128, Grain-128, etc. [12]–[14], and is secure enough for use in most short term wireless communication application scenarios. In the proposed approach, we adopt the KSA algorithm (Algorithm 1) from RC4 for the generation of the random initial permutation S, but replace the PRGA of RC4 by a *Lightweight pseudo-random Generation Algorithm* (LPRGA) which is used for keystream generation from the input permutation S. The proposed LPRGA is as shown in Algorithm 3.

The security of the proposed algorithm has been tested thoroughly. We have tested the randomness property of the keystream generated by our proposed LPRGA using the NIST statistical test suite [2]. The results obtained have shown that the output keystream generated by LPRGA is as random as the output generated by the PRGA of RC4 as long as the keystream sequence length is limited to 30,000 bits and a new key is used to generate each keystream sequence. These requirements are satisfied in nearly all mobile transactions and secure communication applications in WSNs, BANs, MANETs, and VANETs. In addition to the randomness test, the results of the structural tests also confirm the security of the proposed LPRGA. We have measured the period lengths of the LPRGA, and compared with those of the PRGA. The results demonstrate that the proposed algorithm has much larger period than the RC4 algorithm. The unpredictability of the proposed algorithm is also proved analytically. Finally, our proposed hardware solution demonstrates that it is possible to implement LPRGA in hardware with low overhead, yet high throughput keystream generation capability with one cipher byte per clock cycle using a clock frequency of 1 GHz with 65 nm fabrication technology.

```
procedure: KSA(Secret Key $K$)
1 begin
2     Initialize: $S \leftarrow \{0, 1, ..., N-1\}$, $j \leftarrow 0$;
3     for $i = 0, ..., N-1$ do
4         $j \leftarrow j + S[i] + K[i]$;
5         $S[i] \leftrightarrow S[j]$ ;                    // swap $S[i]$ and $S[j]$
6     end
7     return $S$-box $S$;
8 end
```

Algorithm 1: Key Scheduling Algorithm

```
procedure: PRGA($S$-box $S$)
1 begin
2     Initialize: $i \leftarrow 0$, $j \leftarrow 0$;
3     while generate key-stream do
4         $i \leftarrow i + 1$ ;
5         $j \leftarrow j + S[i]$;
6         $S[i] \leftrightarrow S[j]$ ;                    // swap $S[i]$ and $S[j]$
          output   : $Z \leftarrow S[S[i] + S[j]]$;
7     end
8 end
```

Algorithm 2: Pseudo-Random Generation Algorithm

```
procedure: LPRGA($S$-box $S$)
1  begin
2      Initialize: $i \leftarrow 0$, $j \leftarrow N-1$;
3      Declare temporary variable: $t$;
4      while generate key-stream do
5          $t \leftarrow S[i] + j$;
6          $j \leftarrow i$;
7          $i \leftarrow S[i]$;
8          $S[j] \leftarrow t$;
           output   : $Z \leftarrow S[i] \oplus S[j]$;
9      end
10 end
```

Algorithm 3: Proposed LPRG Algorithm

II. The Proposed LPRGA and its Security Analysis

We describe our proposed Lightweight Pseudo-Random Generation Algorithm (LPRGA) as in Algorithm 3 which will replace the PRGA of RC4 in generating the pseudo-random keystream sequence. The keystreams generated from this LPRGA will be XORed with the succesive bits of a plaintext message to produce the required ciphertext message.

Any pseudo-random generation algorithm to be used by a stream cipher should possess the following two important properties, viz., *randomness*: to approximate the properties of a truly random sequence, and *large period*: to ensure that the length of the generated sequence is not less than plaintext message length. Also, since the output keystreams are generated using deterministic algorithms from some secret input values, for a cryptographically secure stream cipher, it is important to demonstrate using *structural tests* that the correlation between input and output values of the pseudo-random number generator (PRNG), used by the cipher, is negligibly small. In addition, a cryptographically secure stream cipher must use an *unpredictable* PRNG in the sense that, from a sequence of known output keystream bits of the PRNG, it should be computationally difficult to predict the preceding bits or the successive bits in the sequence. In the following subsections, we analyze the above mentioned four security properties of the proposed LPRGA in the context of short term wireless communications.

A. Randomness Test

There are a good number of statistical test suites in the literature [2], [4], each of which defines different sets of test statistic parameters for examining different randomness properties of a pseudo-random number generation algorithm. Among these, the NIST statistical test suite [2] is the most widely used one in the field of cryptography, which we have used for comparing the randomness of the LPRGA with that

TABLE I
RANDOMNESS TEST RESULTS

	Fraction passed with	
Test	**PRGA**	**LPRGA**
Frequency	0.99	0.98
Block Frequency	0.99	0.98
Cumulative Sums	0.98	0.98
Runs	0.99	0.96
Longest Run	0.99	0.99
Rank	0.99	0.99
FFT	0.99	0.99
Non Overlapping Template	0.99	0.98
Overlapping Template	0.99	0.99
Universal	1.00	1.00
Approximate Entropy	0.98	0.97
Random Excursions	1.00	1.00
Random Excursions Variant	1.00	1.00
Serial	0.99	0.99
Linear Complexity	0.99	0.97

of the PRGA. In order to generate each keystream sequence, a randomly chosen 16-byte secret-key is input to the KSA algorithm, and the output permutation produced by KSA is input to the PRGA or LPRGA. We have experimented with 100 keystream sequences, each 30,000 bits long with word-size of 8 bits ($n = 8$) and with a different secret key.

The results of our experiment are shown in Table I, where the leftmost column mention all the 15 statistical tests defined by NIST and the figures under the column headings PRGA and LPRGA show the proportions of the sequences passing the respective tests by the PRGA and LPRGA, respectively. The proportion of sequences passing a test is directly dependent on the value of the significance level (α) used by NIST. For the default value of $\alpha = 0.01$, the expected value of this proportion is 0.99, and the lower bound on the proportions is 0.96015. Table I shows the test results for both of the algorithms. Values for the non overlapping template test, cumulative sums test, random excursions test, random excursions variants test, and serial tests have been averaged. Table I shows that the proportion values for both the LPRGA and PRGA algorithms are almost the same for all the tests except the runs test. However, the proportion of keystream sequences, generated by the LPRGA, passing a test is above the lower bound of 0.96015 (without rounding off the figures) for all of the 15 statistical tests. Hence, for the above mentioned configurations, the sequences generated by the LPRGA have highly satisfactory level of randomness as that of the PRGA algorithm.

The randomness property of LPRGA as obtained from these test results is realizable in a practical wireless communication system if the size of a data packet is limited to 30,000 bits and each data packet is encrypted with a different key. The first requirement is easily met as the typical size of a data packet used in wireless communication networks following the IEEE 802.11 standards is only 1.5 KB = 12,000 bits. For the second condition to meet, a 16-byte encryption key for each data packet can be generated by combining (XOR-ing) the 16-byte shared secret key of the session (K_{ssn}) with a 1-byte Initialization Vector (IV) for each packet. The IV for a packet can be randomly chosen by the packet sender and can be put into the header of the packet. So, the size of the encryption key space is equal to the size of the session key space which is equal to 2^{128}, although the IV is only one byte long. In wireless ad hoc network communications, the durations of data sessions are usually very short, and hence, the total number of data packets transmitted during a session is also small. So, the probability that the same encryption key would be used for two different data packets of a session is negligible. It can be noted that, there are a large number of efficient key exchange protocols for wireless ad hoc networks using which the session key (K_{ssn}) between a pair of communicating principals can be changed for every communication session.

B. Structural Test

The purpose of a structural test is to measure the correlation between an output keystream and the input values from which the stream is generated. In [3], four types of structural tests have been defined for synchronous stream ciphers, viz., *i*) *key/keystream correlation test*, *ii*) *IV/keystream correlation test*, *iii*) *frame correlation test*, and *iv*) *diffusion test*. The purpose of the first test is to measure the correlation between an input key (session key) and the corresponding output keystream, for a fixed IV value; whereas, the second test is aimed at measuring the correlation between an IV value and the corresponding output keystream, for a fixed session key. The third test measures the correlations among keystreams generated from similar IV values. The fourth test analyzes the diffusion property of each bit of session key and IV on the output keystream.

In our approach, as discussed above, an encryption key for a data packet will be generated by XOR-ing the session key with a randomly chosen IV value. Hence, instead of measuring the correlation between session key and keystream, and correlation between IV and keystream separately, we only measure the correlation between the final encryption key and the corresponding output keystream. Also, in the diffusion test, we analyze the diffusion property of the bits of the final encryption key on the output keystream. Since the IV values for successive data packets are chosen randomly, we omit the frame correlation test for the proposed cipher. The details of the above mentioned tests and the obtained results are discussed as follows.

1) Encryption Key / Keystream Correlation Test: In order to observe the correlation between encryption key and output keystream, a large number (say, m) of different 128-bit encryption keys are chosen randomly. For each encryption key, a keystream of length 128 bits is generated, which is XOR-ed with the corresponding input encryption key. The weight values of the XOR-ed outputs are calculated in each

TABLE II
RESULTS OF THE STRUCTURAL TESTS.

	Key/Keystream Correlation Test	Diffusion Test
Average p-value	0.834	0.757
Pass-proportion	0.98	0.98

of the m cases. The interval $0 - 128$ is divided into a fixed number of subintervals, and the proportion of weight values falling in each of these subintervals is counted. As described in [3], for a secure stream cipher, the probability distribution of the above mentioned weight values should be binomial with Prob(weight = k) = $\binom{128}{k}(1/2)^{128}$. From this, we can compute the expected frequency of weight values in a specific subinterval. The observed proportions are compared with the expected frequencies using the Chi-square goodness of fit test. We repeat the above test process 100 times, with $m = 2^{20}$ in each case, and finally measure the proportion of tests where the observed weight values passes the Chi-square goodness of fit test, for a confidence value of 0.99.

2) Diffusion Test: In order to analyze the diffusion property of encryption key on output keystream, we first take a random 128-bit encryption key and generate an output keystream of length 256 bits. Now, we change each bit of the encryption key one-by-one and regenerate the keystreams, and XOR the regenerated keystreams with the original keystream. With all of these XOR outputs, we obtain a (128×256) matrix of binary values. We repeat the above process for N ($N = 2^{10}$) randomly chosen encryption keys and add all of the obtained N matrices. Now, for a secure stream cipher, the probability distribution of the values for the elements of the above mentioned resultant matrix should be normal with mean $N/2$ and variance $N/4$. We compare the observed values with the expected values using Chi-square goodness of fit test. We repeat the above test process 100 times as mentioned in the previous test.

Table II shows the results of the above two tests for the proposed LPRGA. We have listed the average p-values for each test along with the proportion of tests that passes successfully. The results confirm that the output keystreams generated by the proposed LPRGA are secure, in the sense that, they are neither positively nor negatively correlated with the corresponding input values.

C. Period Measurement

We have measured the periods for both the PRGA and the LPRGA for $n = 2$ and 3 with all possible initial states. It may be noted that the states of PRGA or LPRGA are fully determined by the two n-bit index variables i, j and the values in the S-box S. The KSA algorithm gives a permutation of the set $\{0, ..., 2^n - 1\}$ in the S-box S, which is the only input to the PRGA or LPRGA. Hence, the total number of initial states is $2^n!$. (For $n = 2$, the total number of initial states = 24, and for $n = 3$, the total number of initial states = 40320). Table III summarizes the periodicity properties of PRGA and LPRGA for 2 bits and 3 bits word-sizes. We see that the average period of LPRGA is almost 6.5 times larger than that of PRGA for $n = 2$ bits, and almost 563 times larger for $n = 3$ bits. For $n = 3$ bits, both the minimum and the maximum values of period for LPRGA are much larger than those values for PRGA, and also the percentage of deviation of period values from the mean period value is smaller in LPRGA than in PRGA. For $n = 2$ bits, the maximum period of LPRGA is much larger than that of the PRGA, but the minimum period is smaller and the percentage deviation of period values is larger in LPRGA than those in the PRGA.

It is expected that the period with LPRGA will be larger than that with PRGA. This is because in Algorithm 2, we see that in an intermediate state of PRGA, the index variables i and j can take any value between 0 and $2^n - 1$. The S-box S can take any one of the $n!$ permutations of $\{0, ..., 2^n - 1\}$, since the only operation on S-box S is a swap operation. So, the state space size (total number of possible intermediate states) in PRGA is equal to $(2^n)!(2^n)^2$. On the other hand, in Algorithm 3, we see that the index variables i and j can take any value between 0 and $2^n - 1$. Also, each element of the S-box S can take any value between 0 and $2^n - 1$. So, the state space size in LPRGA is equal to $2^{n(2^n+2)}$ which is much larger than $(2^n)!(2^n)^2$, which would lead to a larger period than that with PRGA, and it will be more likely so when n increases to higher values.

D. Unpredictability Analysis

The unpredictability requirement for a cryptographically secure pseudo-random number generation algorithm states that, given a set of x successive outputs ($x > 0$) of an output stream generated by the algorithm, it should be computationally difficult to obtain the subsequent outputs or the previous outputs from the given set. In the following, we prove for the proposed LPRGA that, if an adversary knows x successive outputs $\{Z_t, Z_{t+1}, ..., Z_{t+x-1}\}$ of an output keystream generated by the LPRGA, the probability for the adversary to correctly guess the values of either Z_{t-1} or Z_{t+x} is not greater than $\frac{1}{2^n}$, where n is the word-size. It may be noted here that, since there are only 2^n possible values for an output in the output keystream, an adversary can always guess an output value with a probability not less than $\frac{1}{2^n}$.

The LPRGA can be divided into two component subroutines, viz., *i*) the state change function, and *ii*) the output function. The state change function defines how the internal state, defined by the values of the two index variables i, j, and the S-box S, is changed in an iteration of the algorithm. The output function defines how the output key is generated in an

TABLE III
COMPARISON OF PERIOD VALUES FOR WORD LENGTHS 2 AND 3

	Period Values for Word Length 2		Period Values for Word Length 3	
	PRGA	**LPRGA**	**PRGA**	**LPRGA**
Minimum Period	164	78	24	9322
Maximum Period	196	1734	955496	466272794
Average Period	180	1166	762183	429225232
Standard Deviation of Periods	16	670	343763.11	124033704.89
Percentage of Standard Deviation	8.89	57.44	45.11	28.90

iteration from the values of the state variables i, j and S. By analyzing LPRGA, we can extract the above two subroutines of LPRGA as stated below.

The state change function of LPRGA can be defined by the eqns. (1), (2) and (3), which show how the values for the state variables i, j and S for the $(t+1)^{th}$ state are determined by the values of these variables for the t^{th} state.

$$i_{t+1} = S_t[i_t] \tag{1}$$

$$j_{t+1} = i_t \tag{2}$$

$$S_{t+1}[k] = \begin{cases} S_t[k]; & \text{if } (k \neq i_t) \\ S_t[k] + j_t; & \text{otherwise} \end{cases} \tag{3}$$

It may be noted that the initial inputs to the LPRGA are $i_0 = 0, j_0 = N-1$, and S_0, which is the output of the KSA algorithm for a given secret key. Let Z_0 indicate the first output byte of the output keystream generated by LPRGA. Hence, the output function of LPRGA can be defined by the following equation:

$$Z_t = S_{t+1}[i_{t+1}] \oplus S_{t+1}[j_{t+1}], \tag{4}$$

where $A \oplus B$ indicates a bitwise XOR operation between the operands A and B.

From eqns. (1), (2), (3) and (4), we get,

$$\begin{aligned} Z_t &= S_{t+1}[i_{t+1}] \oplus S_{t+1}[j_{t+1}] \quad \text{(from eqn. (4))} \\ &= S_{t+1}[i_{t+1}] \oplus S_{t+1}[i_t] \quad \text{(from eqn. (2))} \\ &= S_{t+1}[i_{t+1}] \oplus (S_t[i_t] + j_t) \quad \text{(from eqn. (3))} \quad (5) \\ &= S_t[i_{t+1}] \oplus (S_t[i_t] + j_t) \quad \text{(from eqn. (3))} \\ &= S_t[S_t[i_t]] \oplus (S_t[i_t] + j_t) \quad \text{(from eqn. (1))} \quad (6) \end{aligned}$$

Since Z_0 depends on $S_0[i_0] = S_0[0]$ and $j_0 = N-1$, and S_0 is not known to an adversary, the adversary cannot correctly predict the value of Z_0 with a probability greater than $\frac{1}{2^n}$. Now, from eqn. (5) we get,

$$\begin{aligned} Z_t &= S_{t+1}[i_{t+1}] \oplus (S_t[i_t] + j_t) \\ &= i_{t+2} \oplus (i_{t+1} + j_t) \quad \text{(from eqn. (1))} \\ &= i_{t+2} \oplus (i_{t+1} + i_{t-1}) \quad \text{(from eqn. (2))} \quad (7) \end{aligned}$$

Thus, for $t = 0, 1, 2, 3, \cdots$, we get,

$$\begin{aligned} Z_0 &= i_2 \oplus (i_1 + N - 1) \\ Z_1 &= i_3 \oplus (i_2 + 0) \quad (\because i_0 = 0) \\ Z_2 &= i_4 \oplus (i_3 + i_1) \\ Z_3 &= i_5 \oplus (i_4 + i_2) \\ &\vdots \end{aligned}$$

From a set of such x consecutive known output values $\{Z_t, ..., Z_{t+x-1}\}$, $(\forall t \geq 0, x \geq 1)$, one obtains a set of x linear equations, where the number of unknowns is always greater than the number of equations. Hence, by solving these equations, the correct values of the unknown variables can only be probabilistically found with a probability $p < 1$. Now, to obtain the value of Z_{t+x} using eqn. (7), an adversary requires the values of either i_{t+x+1} and i_{t+x-1}, or $(i_{t+x+1} + i_{t+x-1})$, and the value of i_{t+x+2}. The probability of correctly predicting the values of either i_{t+x+1} and i_{t+x-1}, or $(i_{t+x+1} + i_{t+x-1})$ from the known values of $\{Z_t, ..., Z_{t+x-1}\}$ is p (for some $p < 1$). The variable i_{t+x+2} is not used in any of the x linear equations corresponding to $Z_t, ...Z_{t+x-1}$. Hence, the probability for an adversary to correctly predict the value of i_{t+x+2} from the known values of $\{Z_t, ..., Z_{t+x-1}\}$ is only $\frac{1}{2^n}$. So, the probability for an adversary to correctly predict the value of Z_{t+x} from the known value of $\{Z_t, ..., Z_{t+x-1}\}$ is equal to $(p \times \frac{1}{2^n})$ which is clearly less than $\frac{1}{2^n}$, since $p < 1$.

Using similar arguments as stated above, it follows that the probability of correctly guessing the value of Z_{t-1} from the known values of $\{Z_t, ..., Z_{t+x-1}\}$ is less than $\frac{1}{2^n}$.

III. A HIGH PERFORMANCE HARDWARE DESIGN OUTLINE

The schematic diagram for hardware implementation of LPRGA is shown in Fig.1 where we use 8-bit registers for i, j and S-box, L is an 8-bit latch and the register bank S contains 256 registers. Thus, the hardware requirement is very small and well suited for implementation in wireless devices considering the cost, complexity, and execution time. It may be noted that this hardware overhead is much smaller than that in the hardware implementation for RC4 given in [1].

Since a modern NAND gate fabricated using 65 nm or 90 nm fabrication technology has a delay of only a few picoseconds, the total signal propagation delay to generate a keystream byte in the proposed LPRGA hardware circuit

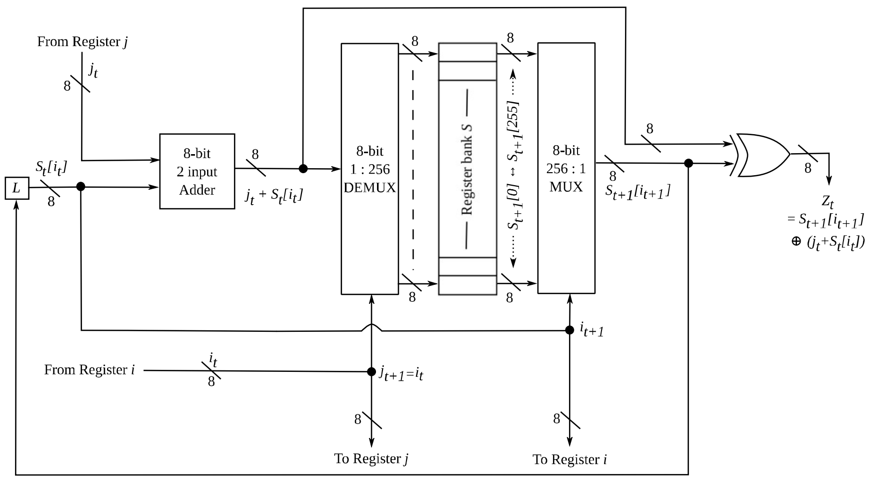

Fig. 1. A simple hardware circuit for the LPRGA algorithm.

will be significantly smaller than a nanosecond so that the proposed LPRGA hardware circuit can generate a throughput of one cipher byte per cycle of a 1 GHz clock using 65 nm fabrication technology. Considering a maximum of 100 Mbps data transmission rate, the time required by a wireless communication device to transmit a single byte of data would be about 80 nanoseconds, which is much more than the time required by the stream cipher to generate one cipher byte. Thus, with considerably lower energy overhead as compared to [1], the proposed hardware can be utilized to produce keystream bytes for generating ciphertext messages in even very high throughput data applications.

IV. Conclusion

We have proposed a novel stream cipher algorithm that is significantly lightweight, yet equally secure as RC4, Grain-128 and other stream ciphers in the context of wireless applications that use short sessions. The strength of LPRGA has been established through theoretical unpredictability analysis, and the NIST statistical test suite. The proposed LPRGA can also be implemented in hardware with a very low cost and complexity and a data rate of 100 Mbps. This makes the proposed LPRGA extremely suitable for implementing secure communication in a wide range of wireless applications, where devices are constrained by either energy, cost or computing capability. Our future work includes developing a complete hardware implementation using VHDL and then comparing its performance in terms of throughput and area with those of the existing stream cipher designs.

References

[1] S. S. Gupta, A. Chattopadhyay, K. Sinha, S. Maitra and B. P. Sinha, "High performance hardware implementation for RC4 stream cipher," *IEEE Trans. on Computers*, vol. 62(4), April 2013, pp. 730–743.

[2] A. Rukhin, J. Soto, J. Nechvatal, M. Smid, E. Barker, S. Leigh, M. Levenson, M. Vangel, D. Banks, A. Heckert, J. Dray, and S. Vo, "A statistical test suite for random and pseudorandom number generators for cryptographic applications," http://www.nist.gov., 2001.

[3] M. S. Turan, A. Doganaksoy and C. Calik, "Statistical analysis of synchronous stream ciphers," *Proc. SASC 2006: Stream Ciphers Revisited*, 2006.

[4] G. Marsaglia, "DIEHARD Statistical Tests," http://stat.fsu.edu/geo/diehard.html.

[5] R. Basu, S. Ganguly, S. Maitra, G. Paul, "A complete characterization of the evolution of RC4 pseudo random generation algorithm," *Journal of Mathematical Cryptology*, vol. 2(3), 2008, pp. 257-289.

[6] T. Good and M. Benaissa, "Hardware results for selected stream cipher candidates," eSTREAM, ECRYPT Stream Cipher Project, *Proc. SASC*, Report 2007/023, 2007.

[7] F. K. Gurkaynak, P. Luethi, N. Bernold, R. Blattmann, V. Goode, M. Marghitola, H. Kaeslin, N. Felber and W. Fichtner, "Hardware evaluation of eSTREAM candidates: Achterbahn, Grain, MICKEY, MOSQUITO, SFINKS, Trivium, VEST, ZK-Crypt," eSTREAM, ECRYPT Stream Cipher Project, Report 2006/015, 2006.

[8] V. A. Ghafari, H. Hu, and C. Xie, "Fruit: ultra-lightweight stream cipher with shorter internal state," eSTREAM, ECRYPT Stream Cipher Project, https://eprint.iacr.org/2016/355.pdf.

[9] C. Manifavas, G. Hatzivasilis, K. Fysarakis and Y. Papaefstathiou, "A survey of lightweight stream ciphers for embedded systems," *Security and Communication Networks*, vol. 9(10), July 2016, pages 1226-1246.

[10] T. Eisenbarth, S. Kumar, C. Paar, A. Poschmann and L. Uhsadel, "A survey of lightweight-cryptography implementations," *IEEE Design & Test of Computers*, vol. 24(6), Nov. 2007, pp. 522–533.

[11] M. B. Shemaili, C. Y. Yeun, K. Mubarak and M. J. Zemerly, "A new lightweight hybrid cryptographic algorithm for the internet of things," *Intl. Conf. for Internet Technology and Secured Transactions*, London, 2012, pp. 87–92.

[12] M. Hell, T. Johansson, A. Maximov and W. Meier, (2006, July). "A stream cipher proposal: Grain-128," *Proc. IEEE International Symposium on Information Theory*, 2006, pp. 1614–1618.

[13] R. L. Rivest, "The RC5 Encryption Algorithm," *Proc. 2nd International Workshop on Fast Software Encryption* (FSE), 1994, pp. 86-96.

[14] eSTREAM: the ECRYPT Stream Cipher Project, http://www.ecrypt.eu.org/stream/.

[15] G. Paul and S. Maitra, "On biases of permutation and keystream bytes of RC4 towards the secret key," *Cryptography and Communications - Discrete Structures, Boolean Functions and Sequences*, vol. 1(2), 2009, pp. 225-268.

[16] I. Mantin, "Predicting and distinguishing attacks on RC4 keystream generator," *EUROCRYPT 2005*, LNCS, vol. 3494, 2005, pp. 491-506.