A LCG-based Secure Protocol for Wireless Sensor Networks

Bo Sun, Chung-Chih Li
Dept. of Computer Science
Lamar University
Beaumont, TX, USA 77710
{sunbx, licc}hal.lamar.edu

Kui Wu
Dept. of Computer Science
University of Victoria
BC, Canada V8W 3P6
wkui@cs.uvic.ca

Yang Xiao
Dept. of Computer Science
The University of Memphis
Memphis, TN, USA 38152
yangxiao@ieee.org

ABSTRACT - In this paper, based on a Linear Congruential Generator (LCG), we propose a new block cipher that is suitable for constructing a lightweight secure protocol for resource-constrained wireless sensor networks. Based on the Plumstead's inference algorithm, we are motivated to embed the generated pseudo-random numbers with sensor data messages in order to provide security. Specifically, the security of our proposed cipher is achieved by adding random noise and random permutations to the original data messages. The analysis of our cipher indicates that it can satisfy the security requirements of wireless sensor networks. We demonstrate that secure protocols based on our proposed cipher satisfy the baseline security requirements: data confidentiality, authenticity, and integrity with low overhead. Performance analysis demonstrates that our proposed block cipher is more lightweight than RC5 in terms of the number of basic operations.

Keywords - Wireless Sensor Networks, Linear Congruential Generator, Security

I. INTRODUCTION

Wireless Sensor Networks (WSNs) have been used for a wide variety of applications [1]. In hostile and un-trusted environments such as battlefield surveillance, an adversary can eavesdrop on traffic, inject new messages, and replay old messages. Therefore, it is necessary to incorporate appropriate secure mechanisms into WSNs. However, given the stringent constraints on processing power, memory, bandwidth, and energy consumption, it is very difficult to design suitable secure mechanisms for WSNs.

The constraints posed by the sensor hardware make it impossible to deploy most of the traditional security primitives and protocols, such as the RSA [17] and Diffie-Hellman algorithm [20], because they require expensive computations and long messages that could easily exhaust the sensor's resources. Symmetric cryptography can be used in WSNs. For example, SPINS [10] used RC5 [14] as the block cipher. TinySec [12] used Skipjack [18] as the default block cipher. The performance of the proposed security protocols depends heavily on the encryption primitives themselves.

In this paper, we take a different step to tackle the security problems for WSNs. We aim at proposing a more lightweight block cipher that is suitable for WSNs. We are motivated by the fact that a suitable lightweight block cipher can significantly reduce the overhead of the security protocols built on it. Therefore the overall performance can be improved dramatically.

Specifically, we propose a lightweight block cipher that is based on a Linear Congruential Generator (LCG) [20]. In theory, cryptosystems based on a pseudo-random number generator (PRNG) (for example, LCG) are not suggested because they are predictable [3]. However, after properly arranging the use of numbers generated by a LCG, we can not only achieve the desirable security properties but also enjoy the high efficiency provided by a LCG. Utilizing the simplest form of a LCG and based on the experiment from the Plumstead's algorithm [5], we demonstrate that it is impossible to significantly enhance the security of the system simply by increasing the size of the modulus. By adding random noise generated by a LCG and random permutations to sensor data messages, we demonstrate that our proposed cipher is secure enough for WSNs. At the same time, it can also reduce the cost of security provision. We compare the number of basic operations of our proposed cipher with that of RC5, which is one of the most commonly used algorithms in security protocols for wireless sensor networks [10]. Analytical results demonstrate that our proposed block cipher is more lightweight than RC5.

II. SECURITY GOALS

- *Confidentiality*: One goal of our protocol is that sensor readings/data cannot be disclosed to attackers. Another stronger requirement, which is also our goal, is *Semantic Security*, which ensures that an adversary has no information about the plaintext, even if it sees multiple encryptions of the same plaintext [21].
- *Integrity*: It makes sure that if an adversary modifies a data message from an authentic sender, the receiver should be able to detect this tampering.
- *Authenticity*: It ensures that data messages come from the intended sender. Authenticity can prevent some third party from injecting falsified messages into the network.

III. KEYING MECHANISMS

Our protocol assumes the existence of a key management scheme and can work well with any of key management protocols. The easiest key management scheme is to use a network-wide shared key among all the nodes. However, the compromise of any single node can paralyze the whole network.

A more robust approach is for groups of neighboring nodes to share a key. In this way, a compromised node can only decrypt the messages from nodes in its group, and cannot decrypt messages from other groups and cannot inject falsified messages into other groups.

The most robust but the most complicated approach is for WSN nodes to set up pairwise keys on the fly. It can effectively defend against node capture attacks.

We assume that there exists a key management sub-system that makes it possible for WSN nodes to negotiate the key setup and bootstrap the corresponding trust relationship. This is a reasonable assumption given the fact that research regarding the group key and pairwise key setup has been carried out extensively [10] [11]. They could be utilized to provide a security service to our protocol.

IV. LCG-based Security Protocols

A. Why selecting LCG

Almost every cryptosystem needs a source of random numbers either in constructing keys for encryption algorithms or in generating enough randomness for scrambling the sensitive information. Many Pseudo-Random Number Generators (PNRG) have been introduced for practical purposes.

It is easy for us to think of linear algorithms when *efficiency* and *simplicity* come to our top priorities. However, a close examination of some widely used linear PRNGs listed in [2] shows that they are all proven to be cryptographically insecure. For example, most commercial Linear Congruential Generators (LCG) do not intend to be used for cryptographic purposes [2]. A series of investigations of LCGs in late 80's and early 90's have been done and raised a substantial doubt about using LCGs in any cryptosystem.

However, this is based on the assumption that an enough amount of sequences generated by a fixed PRNG is known to the attacker. If we can use the information itself to protect the random sequences, we can use the linear PRNGs as an efficient mechanism to protect the data transmission in wireless sensor networks. Motivated by this, we pick up the LCG in its simplest form to produce pseudo-random numbers. The reason we select the LCG is because it is the simplest, most efficient, and a well-studied pseudo-random number generator.

B. Linear Congruential Generators

The simplest form of a LCG uses the following equation:

$$X_{n+1} = aX_n + b \mod m, \quad n = 0, 1, 2, \ldots \quad (1)$$

where a is the *multiplier*, b is the *increment*, and m is the *modulus*. X_n and X_{n+1} are the n^{th} and $(n+1)^{st}$ numbers, respectively, in the sequence generated by the LCG. X_0 is called the *seed* of the LCG. X_0, a, b, and m are the parameters. The statistical properties of the pseudo-random numbers generated by an LCG depend on the selection of its parameter [3].

1) Predictability of LCGs: In order to properly arrange the use of pseudo-random numbers generated by a LCG, we need experimental results to decide how many numbers are actually needed to successfully infer the entire sequence. Because of this, we implement the Plumstead's inference algorithm [5] against the LCG in its easiest form as shown in Equation (1). We implement the algorithm to observe how many pseudo-random numbers are actually needed for successfully recovering the parameters of an unknown LCG, so we can adequately adjust our cipher to meet the security requirements.

2) Plumstead's Algorithm: Assume Equation (1) is a LCG with the fixed parameters a, b, m, and X_0, where $m > max(a, b, X_0)$. The algorithm will find a congruence $X_{n+1} = \hat{a}X_n + \hat{b} \mod m$, possibly with a different multiplier and increment but generating the same sequence as the fixed congruence does. The inference consists of two stages as follows.

Let $Y_i = X_{i+1} - X_i$.

- **Stage I:** In this stage, we find $\hat{a}$ and $\hat{b}$ as follows:
  1. Find the least t such that $d = gcd(Y_0, Y_1, \ldots Y_t)$ and d divides Y_{t+1}.
  2. For each i with $0 \leq i \leq t$, find u_i such that
  $$\sum_{i=0}^{t} u_i Y_i = d.$$
  3. Set $\hat{a} = \frac{1}{d}\sum_{i=0}^{t} u_i Y_{i+1}$, and $\hat{b} = X_1 - \hat{a}X_0$.
  This stage will give $X_{i+1} = \hat{a}X_i + \hat{b} \mod m$ for all $i \geq 0$.
- **Stage II:**
  In this stage, we begin predicting X_{i+1} and, if necessary, modifying m. When a prediction X_i is made, the actual value will be available to the inference algorithm. Initially, we set $i = 0$ and $m = \infty$ and assume X_0 and X_1 are available (we can reuse the numbers used in the previous stage). Repeat the following steps:
  1. Set $i = i + 1$ and predict
  $$X_{i+1} = \hat{a}X_i + \hat{b} \mod m.$$
  2. If X_{i+1} is incorrect, $m = gcd(m, \hat{a}Y_{i-1} - Y_i)$.
  X_i can be inferred *in the limit*. Please refer to [5] for a detailed proof.

3) Analysis of Plumstead's Algorithm: It is clear that every step in both stages is polynomial-time computable in terms of the size of m. Plumstead proves that in Stage I t is bounded by $t \leq \lceil \log_2 m \rceil$. The number of incorrect predictions made in Stage II is bound by $2 + \log_2 m$. Therefore, the algorithm is optimal with a sample complexity $O(\log_2 m)$ in the worst case.

4) Empirical Results of Plumstead's Algorithm: We tested the module, m, from 1 byte and double its size up to 32 bytes. For $m \geq 2$ bytes, we used the Miller-Rabin Test [8], a very efficient randomized algorithm for primality tests, to select and determine prime numbers with an error rate less than $(\frac{1}{2})^{\lceil \log_2 m \rceil}$. Given m, we select 1000 sets of different parameters (a, b, m, and X_0). For each set of parameters, we generated the sequence of pseudo-random numbers $X_1, X_2, \ldots ....$ We ran the *Plumstead*'s algorithm to decide how many X_i are needed to recover the set of parameters (a, b, m, and X_0).

The results of our experiments are shown in Table I, in which μ is the average number of samples needed to successfully infer the pseudo-random number sequence while δ is the standard deviation. Experimental results show that the Plumstead's algorithm is much more powerful than what the theoretical analysis has suggested. We observe that the number of samples needed in average is far fewer than that of the worst case. Also, Table I contains the best case (min) and the worst case (max) for each size. The values of δ in Table I indicate that the worse case occurs rarely.

TABLE I
RESULTS OF PLUMSTEAD'S ALGORITHM

$\|m\|$ Bytes	μ	δ	min	max
1	5.438	0.939	5	12
2	5.617	1.221	5	17
4	5.554	1.082	5	15
8	5.586	1.114	5	16
16	5.802	1.764	5	31
32	6.105	3.149	5	57

Based on the results illustrated in Table I, we can see that the size of m does not prolong the inference process significantly. Therefore, for a LCG, instead of increasing the size of m, we need to hide the numbers generated. Also, from the results illustrated in Table I, we can see that *if we can find a way to prevent the adversary from retrieving five or more consecutive numbers from the sequence, our cipher based on the LCG will be secure.* Our design follows the above principle by using the transmitted information to protect the sequence of random numbers and by using a re-keying mechanism.

C. Key Selection

The moduli we choose is a 16 byte prime. This could also facilitate the selection of suitable X_0, a, b, and m that satisfy the security requirements. By the Prime Number Theorem that the number of positive prime less than n is asymptotic to $n/\ln n$, the density of 16 byte primes is about $\frac{1}{\ln 2^{128}} = 0.0127$. Therefore, on average we can successfully pick up a prime within about 100 random selections. Then, we randomly assign numbers less than m to a, b, and X_0 without further imposing any restriction except for some trivial values such as 0 or 2^k. In our scheme, we only keep X_0 as the secret shared between two nodes. a, b, and m can be made open. They could be treated as the WSN parameters. Careful selections of a, b, and m are needed to achieve the maximum security using the LCG.

In this respect, we apply Hull and Dobell's Theorem [23].

a) Hull and Dobell's Theorem: : The linear congruential sequence $X_0, X_1, X_2, \ldots$ generated by

$$X_{n+1} = aX_n + b \mod m \tag{2}$$

has a period (the number of integers before the sequence repeats) of length m if the following conditions hold:

1) $\gcd(c, m) = 1$: The only positive integer that (exactly) divides both m and c is 1, i.e., c is relatively prime to m.
2) $p|(a-1)$, for every prime p such that $p|m$: If p is a prime number that divides m, then p divides $(a-1)$.
3) if $4|m$, then $4|(a-1)$: If 4 divides m, then 4 divides $(a-1)$.

Since the results of Plumstead's algorithm suggest that the LCG can be broken almost in a constant number of observed random numbers, our system is not more secure if we keep all parameters a, b, m, and X_0 in secret. In this respect, we make them public except X_0. Our goal is to hide all random numbers from the adversary and setup a system that chosen-plaintext attack can't be conducted. The security of our system then does not rely on the cryptographic strength of the LCG (which is extremely weak). Instead, we rely on the LCG's statistical randomness, i.e., uniformality and period of repetition. Besides the LCG, such statistical properties of any PRNG can be easily tested. Based on Hull and Dobells Theorem, the LCG can reach such maximal statistical randomness under the conditions listed above, which are rather easy to achieve. When the period of the LCG reaches its maximum value, the chance to guess a right X_0 is $1/m$. Also, in practice, the chance that two nodes have their sequence overlapped is slim when m is sufficiently large. In our case, m has at least 128 bits.

Since X_0 is the only shared secret, key pre-distribution is relatively easier. For example, the Blom key predistribution scheme [24] can be used to allow *any* pair of nodes to compute one secret shared key (*single* key space) (It is worth noting that, based on the Blom key predistribution scheme, Du. *et al.* [11] proposed a pairwise key predistribution scheme using *multiple* key spaces). In this paper, we focus on the discussion of a LCG-based scheme. X_0 can be any number in $\mathbb{Z}_m = 0, 1, \ldots, m-1$. If the environment is detected more hostile, our idea is still workable but a more complicate yet more cryptographically secure PRNG should be used to replace the LCG. Therefore, in this respect, the system is not more secure if we keep a, b, and m the shared secret.

In order to speed up our modulus operation and reduce the computing overhead for each sensor node, we make the following requirement for the multiplier a and the modulus m:

$$2^{63} < a < 2^{64} \quad \text{and} \quad 2^{127} < m < 2^{128}.$$

We will discuss the benefits we can obtain by setting this extra requirement in Section V. It is worth noting that because a, b, and m are open, these extra requirements will not cause extra computation overhead to each sensor node.

D. Basic Hop by Hop Message Transmission

In this section, we use secure data aggregation [1] as an example to illustrate the operations of our LCG-based security protocol. Our proposed security mechanism is general enough and is not limited to data aggregation only.

Notations:

A, B, C...: Sensor nodes

E(P, K): Encryption of plaintext message P using key K

$P_1|P_2$: Concatenation of message P_1 and P_2

MAC(K, P): Message Authentication Code (MAC) of message P using key K

X_0: *Seed* of the LCG

a, b, m: Parameters of the LCG. Together with X_0, they are used to generate secret keys for the message transmission.

K_{AB}: Shared secrets between node A and B. It is X_0 of the LCG.

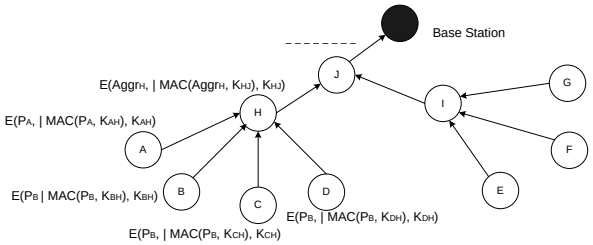

Fig. 1. Hop By Hop Security Protocol.

In Fig. 1, sensor nodes, such as nodes A, B, C, and D transfer the readings to their immediate aggregator, node H. Each sensor node appends a MAC to the plaintext message P and uses their shared secret keys with H to encrypt the whole message. After H receives the readings, it uses the corresponding secret to decrypt and authenticate the received messages, computes and sends out the aggregated result. This time, node H appends a new MAC to the aggregated result and uses its shared secrets with its immediate aggregator, node J, to encrypt the whole message.

1) Message Encryption: The goal of encryption is to prevent an attacker from recovering all the random numbers generated by the LCG. Encryption depends on the underlying block cipher. It cannot involve too many complex operations. The size of the block cipher should not be too large either. Otherwise, given the usually small size of the data messages in WSNs, the message padding will introduce a large overhead.

Our proposed block cipher is 16 bytes in size. For each block cipher, one 16-byte random number X_1 is needed. It is used for the first stage of encryption (**Stage I**). The result of **Stage I** (combine two 8-byte numbers into one 16-byte number) is used for permutations and further encryption (**Stage II**). We also introduce the noise permutation to further scramble results. The general strategy is illustrated in Fig. 2:

a. Step 1 - Random Number Generation: We use the LCG to generate the random number. Given a 16 byte block cipher, one 16 byte random number, X_1, is needed.

b. Step 2 - Stage I: Suppose p_1 and p_2 are the plaintext message to be encrypted using this block cipher. Each p_i is 8 bytes. We embed the pseudo-random number X_1 into the plaintext message in the following way.

For example, let $Wireless\ sensor$ (16 bytes) be the message to be encrypted. So $p_1 = Wireless$, and $p_2 =\ sensor$. The first three characters of p_1 are $W = 87$, $i = 105$, and $r = 114$. The embedding operations are simply the addition modulo 256. If

$X_1 = 10\,5A\,FB\,11\,FC\,BB\,00\,11\,22\,33\,44\,55\,66\,77\,88\,99_h$

The values of the first three bytes are $10_h = 16$, $5A_h = 90$, and $FB_h = 251$. Therefore, the values of the first three ciphertext characters encrypted are:

$87 + 16 \mod 256 = 103$

$105 + 90 \mod 256 = 195$

$114 + 251 \mod 256 = 109$

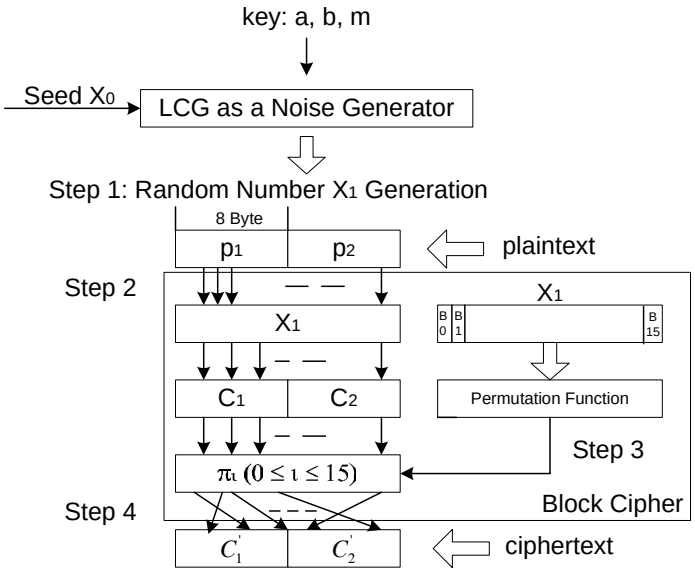

Fig. 2. Message Encryption of a 16 byte Packet.

As illustrated in Fig. 2, C_1, and C_2 are the scrambled text after X_1 is embedded. Each C_i is also 8 bytes.

c. Step 3 - Permutation: X_1 is broken into 16 1 byte random numbers, denoted as $B_0, B_1, \ldots, B_{15}$ respectively. We introduce a permutation function Π over $Z_{16} = \{0, 1, 2, \ldots, 15\}$. Let $\Pi = \pi_0\pi_1\pi_2 \ldots \pi_{15}$ be constructed as follows:

I. $\pi_0 = B_0 \mod 16$;

II. $\pi_i = (n \mod 16)$, for $i = 1 \ldots 15$ with n is the smallest integer such that $n \geq B_i$ and $\pi_i \notin \{\pi_0, \pi_1, \ldots, \pi_{i-1}\}$.

d. Step 4 - Stage II: After we obtain Π, we apply Π to C_1C_2 obtained in **Step 2** in a standard manner, i.e., the i^{th} byte of $\Pi(C_1C_2)$ is the π_i^{th} byte of C_1C_2. Presented by 8 byte segments, let $\Pi(C_1C_2) = C_1'C_2'$, which are our final encrypted message.

Decryption is straightforward. The receiver node could generate the same X_1 that the sender generates. Using X_1, the receiver can obtain p_1 and p_2 following the backward of Fig. 2.

2) Security Analysis: This section analyzes the security protocol in terms of *Confidentiality*, *Authenticity* and *Integrity*.

a. Confidentiality

According to the construction of the permutation function in **Step 3**, the mapping from the random bytes B_i to Π is many-to-one. Under the chosen-plaintext attack, the adversary may successfully obtain a permutation function. However, one permutation function corresponds to

$$\frac{256^{16}}{16!} \approx \frac{2^{128}}{2^{44}} \approx 2^{84}$$

many values for one 16 bytes pseudo-random number. That is, the same permutation function may be constructed based on 2^{84} many different pseudo-random numbers (i.e., B_i). Therefore, it is not feasible to exhaustively search the possible values of the 16 byte pseudo-random numbers.

We only use a half of each byte ($16 = 2^4$) in B_i to construct our permutation function. It follows that the revealing of the permutation function cannot recover the value of B_i. Even from the cryptographic point of view, we consider revealing some

random bits (a half of bits in B_i) a deficiency in a cryptosystem, we have the following analysis. The probability that the values in B_i do not introduce collisions is very low. More precisely, according to the birthday [20] attack, when $n = 16$ and

$$k \approx \sqrt{2n \ln 0.5^{-1}} - 1 \approx 3.7 \quad (3)$$

The probability of $B_k \mod 16 \in \{\pi_0, \pi_1, \ldots, \pi_{k-1}\}$ (the probability of collision) is at least 0.5. Based on Equation 3, starting from π_3, the value of π_i is not likely to be the value of $B_i \mod 16$. As i becomes larger, the chance of collisions becomes larger and the chance that the attacker obtains the right value for B_i becomes smaller.

b. Authenticity and Integrity

We use a Cipher Block Chaining (CBC) MAC to provide authentication and integrity. It has proven that the CBC-MAC is secure if the underlying block cipher is secure [15].

In [12], a choice of a 4 byte MAC is used. This is because in certain applications, it is difficult for the attacker to brute force the key in an off-line manner. Also, given a 19.2kbs channel in WSNs, it is not realistic to send enough data packets to test the MAC. In our design we also use a 4 byte MAC in our protocol.

The CBC-MAC scheme is illustrated in Fig. 3. Here each p_i is 8 bytes and X_1 is 16 bytes. The output of the previous block cipher $C'_1C'_2$ (16 bytes) is used as the input for the next block cipher (i.e., X_1). This process is standard. The final output $C'_{2i+1}C'_{2i+2}$ is 16 bytes. We use $(First\ 4\ Bytes\ of\ C'_{2i+1}) \bigoplus (Second\ 4\ Bytes\ of\ C'_{2i+1}) \bigoplus (First\ 4\ Bytes\ of\ C'_{2i+2}) \bigoplus (Second\ 4\ Bytes\ of\ C'_{2i+2})$ to convert it to a 4-byte MAC,

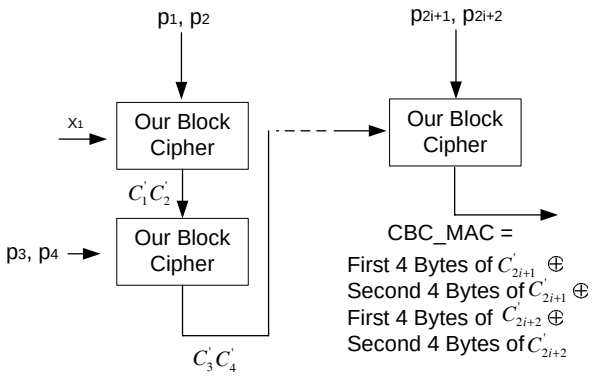

Fig. 3. Integrity and Authenticity.

The final format of our transmitted message is $E(P|MAC(K,P),K)$, instead of $E(P,K)|MAC(K,P)$. The encryption computation in $E(P,K)|MAC(K,P)$ involves less operations because its encryption only operates on P, instead of $P|MAC(K,P)$. However, the MAC code is short in wireless sensor networks. What's more, the MAC function is usually weaker than that of traditional wired networks. The encryption of $E(P|MAC(K,P),K)$ can provide one layer of protection for the MAC.

One approach to achieve *semantic security* in our context is an efficient *rekeying* mechanism. In Fig. 2, C'_1 and C'_2 can be assigned as the new seed (X_0). Therefore, new keys will be generated for the encryption of the next data message at the sender and receiver sides. Note that no message overhead is involved in this process. In doing so, the same plaintext message can be encrypted using different keys, and in this way *semantic security* can be achieved.

If an environment has a high rate of message lost and collision, we can reduce the frequency of rekeying operations to avoid the potential huge number of key synchronization operations. To prevent an adversary from keeping sending bogus messages to trigger the nodes into performing key synchronization, the nodes can send the keys with each encrypted messages.

3) Discussion: The permutation of our cipher in Steps 3 and 4 can guarantee that even if our cipher is applied to a low entropy environment, the security of our cipher will not be significantly compromised. Moreover, for the *known-plaintext attack*, the permutation function takes on in Step 3, in which the random numbers generated by the LCG play an extra role in altering the original order of the content of the message.

For the chosen-plaintext attack, as we mentioned earlier, encrypting two packets and comparing their ciphertexts may reveal about 8 to 12 random bits in a 128-bit random number. This does not provide sufficient information for the adversary to conduct an effective attack in the context of WSNs.

The size of our block cipher is 16 bytes. So for a data packet that is less than 16 bytes, we need to pad it. For a message that is larger than 16 bytes, one approach called *ciphertext stealing* [20] can be used to ensure that the ciphertext has the same length as the underlying plaintext. Note that it is not desirable to send short messages considering the fixed overhead of sending a message (turning on the radio, acquiring the channel, and sending the start symbol) [12]. Also, for data packets that are larger than 16 bytes, we need more than one block cipher to encrypt the whole message. The same X_1 (X_1 used for the first block cipher) is used for the rest of the block ciphers in order to avoid the expensive operations of multiplication and modulo.

V. Performance Analysis

The cryptographic algorithm and the efficiency of the software implementation determine the number of clocks necessary to perform the security function [22]. Therefore, the processing overhead in terms of the *Number of Basic Operations* can reflect the implementation efficiency and the energy consumption of the cryptographic computation. We calculate the *Number of Basic Operations* of our cipher and compare it with RC5. We consider Addition, XOR, Shift (1 bit), Fetch (fetch a value from the main memory to a register), and Store (store a value in a register to the main memory) as our basic operations. In particular, we choose RC5-32/12/X, (i.e., 32 bits words, 12 rounds, and X as the key length) based on the algorithm in [14]. We do not consider the cost of computing the S-Table of RC5 in our analysis.

We consider the cost of performing one general n-bits multiplication as $\frac{n}{2}$ additions and $\frac{n}{2}$ shifts in average on n-bit registers. Since a division can be reduced to a multiplication, we use the same estimation for the division. Also, the same estimation is made to the general modulo.

We have some special cases: a multiplication by 2 is a left-shift operation; the operation of $(n \mod 32)$ is considered one XOR operation. "shift B bits ($\lll B$)" means "shift (B

$\mod 32$) bits". . Also, we use 16 as the average value of ($B \mod 32$). For RC5, we assume the values of A and B, the two words to be encrypted, remain in the registers during the course of computation. Finally, we consider that n basic operations on a 32-bit-processor are equivalent to $8n$ basic operations on a 8-bit processor.

A. Results

Combining the number of basic operations in LCG-based cipher, the number of basic operations for generation of one 128-bit LCG pseudo-random number, and the number of basic operations in RC5, we obtain Fig. 4, which depicts the comparison of the number of basic operations to encrypt a packet at different sizes. Fig. 4 clearly demonstrates the advantage of our proposed cipher. Considering an 8-bit processor, for a 16 byte packet, our encryption mechanism needs roughly $3/4$ amount of basic operations of RC5. What's more, our encryption mechanism takes into consideration the generation of random numbers, which may provide many advantages. For a 32 byte packet, the number of basic operations of RC5 is doubles that of 16 byte packets. However, for a 32 byte packet, our encryption mechanism only slightly increases the number of operations. This is because the second 16 bytes do not need the generation of the random number, which significantly reduces the overhead. Similar observations exist for 64 byte and larger packets.

	8-bit Processor 16 byte Packet	8-bit Processor 32 byte Packet	8-bit Processor 64 byte Packet
RC-5/32/12/X	8384	16768	33536
LCG Cipher	6490	6676	7048

Fig. 4. Numbers of Basic Operations in RC5-32/12/X and LCG-based Cipher.

VI. Related Work

Many research efforts have been devoted to security is WSNs. Perrig *at al.* [10] provides a suite of security building blocks that are optimized for resource constrained WSNs - SNEP and μTESLA. Hu *et. al* [19] studied the secure aggregation problem if one node is compromised. Park *et al.* [16] proposed LiSP - an efficient lightweight protocol that makes a trade-off between security and resource consumption. Karlof *et al.* [12] presented the fully-implemented link layer security architecture for WSNs. There is also much work devoted to the key distribution and management in WSNs [10] [11].

That all sequences generated by the LCG are predictable was first argued by Knuth [3]. Boyar [5] gave a rather complete treatment on the predictability of some of the widely used LCGs. Krawczyk [6] gave an inference algorithm that can predict any sequence generated by the LCG in its most general form, which settled a final theoretical viewpoint to the predictability of LCG. In [7], Ritter strongly warned that any attempt to use LCGs for cryptographical purposes is dangerous ***unless the sequence can be isolated from another generator.*** Our work is motivated by this fact and uses the transmitted information to protect the sequence of random numbers.

VII. Conclusions and Future Work

In this paper, based on a LCG, we propose a lightweight block cipher and apply it to WSNs. The security of our proposed cipher is achieved by adding random noise and random permutations to the original data messages. Security analysis demonstrates that our proposed cipher is secure and suitable for WSNs. At the same time, our proposed cipher is much more efficient in terms of the number of basic operations.

We plan to implement our proposed mechanisms on MICA2 sensor nodes and compare the performance of our block cipher with other popular lightweight ciphers.

References

[1] I. F. Akyildiz, W. Su, Y. Sankarasubramaniam, and E. Cayirci, "Wireless sensor networks: a survey", Computer Networks, 38(4): 393-422, 2002.

[2] K. Entacher, "A Collection of Selected Pseudorandom Number Generators with Linear Structures," TR 97-1, University of Vienna, Austria, 1997.

[3] D.E. Knuth, "Deciphering a Linear Congruential Encryption," *IEEE Transactions on Information Theory*, vol. 31, no. 1, pp. 49-52, January 1985.

[4] J. Boyar, "Inferring Sequences Produced by Pseudo-Random Number Generators," *Journal of the ACM*, vol. 36, num. 1, pp. 129-141, 1989.

[5] J.P. Plumstead (Boyar), "Inferring a Sequence Generated by a Linear Congruence," *Proceedings of the 23rd Annual IEEE Symposium on the Foundations of Computer Science*, pp. 153-159, 1982.

[6] H. Krawczyk, "How to Predict Congruential Generators," *Journal of Algorithms*, vol. 13, no. 4, pp. 527-545, 1992.

[7] T. Ritter, "The Efficient Generation of Cryptographic Confusion Sequences," *Cryptologia*, vol. 15, no. 2, pp. 81-139, 1991.

[8] D. Stinson, "Cryptography: Theory and Practice", Chapman & Hall, 2nd Edition, 2002.

[9] S. Zhu, S. Setia, and S. Jajodia, "LEAP: Efficient Security Mechanisms for Large-Scale Distributed Sensor Networks", *ACM CCS*, Washington DC, pp. 62-72, 2004.

[10] A. Perrig, R. Szewczyk, V. Wen, D. Culler, and J. D. Tygar, "SPINS: Security Protocols for Sensor Networks", *ACM Wireless Networks*, 8(5):521-534, Sept. 2002.

[11] W. Du, J. Deng, Y. Han, and P. Varshney, "A Pairwise Key Pre-distribution Scheme for Wireless Sensor Networks", *ACM CCS*, Washington DC, pp. 42-51, 2004.

[12] C. Karlof, N. Sastry, and D. Wagner, "TinySec: a link layer security architecture for wireless sensor networks", *Proceedings of the 2nd international conference on Embedded networked sensor systems*, Baltimore, MD, USA, pp. 162 - 175.

[13] D. Wagner, "Resilient aggregation in sensor networks", *Proceedings of the 2nd ACM workshop on Security of ad hoc and sensor networks*, Washington DC, USA, 2004, pp. 78 - 87.

[14] R. Rivest, "The RC5 encryption algorithm", *Proc. 1st Workshop on Fast Software Encryption*, 1995, pp. 86 - 96.

[15] M. Bellare, J. Kilian, and P. Rogaway, "The Security of the Cipher Block Chaining Message Authentication Code", *Journal of Computer and System Sciences,* vol. 61, no. 3, December 2000, pp. 363-399.

[16] T. Park and K.G. Shin, "LiSP: A lightweight security protocol for wireless sensor networks", *ACM Transactions on Embedded Computing Systems (TECS)*, Volume 3, Issue 3, pp. 634 - 660, August 2004.

[17] R. L. Rivest, A. Shamir, and L. M. Adleman, "A method for obtaining digital signatures and public-key cryptosystems," *Communications of the ACM*, 21(2):120-126, 1978.

[18] Skipjack and KEA algorithm specifications. http://csrc.nist.gov/encryption/skipjack/skipjack.pdf, NIST, 1998.

[19] L. Hu and D. Evans, "Secure Aggregation for Wireless Networks", Workshop on Security and Assurance in Ad hoc Networks, Orlando, FL, January, 2003.

[20] B. Schneier, "Applied Cryptography: Protocols, Algorithms, and Source Code in C", 2nd Edition, John Wiley & Sons, 1996.

[21] S. Goldwasser, and S. Micali, "Probabilistic encryption", *Journal of Computer Security*, vol. 28, pp. 270-299, 1984.

[22] D. Carman, P. Kruus, and B. Matt, "Constraints and approaches for distributed sensor network security", NAI Labs Tech. Report No. 00010, 2000.

[23] D.E. Knuth, "The Art of Computer Programming", Vol 2: Seminumerical Algorithms, Addison-Wesley, 1969.

[24] R. Blom, "An Optimal Class of Symmetric Key Generation Schemes", *Advance in Cryptography EUROCRYPT*, 1985, Lecture Notes in Computer Science, Vol, 209, pp. 335-338, Springer-Verlag, 1985.