A Lightweight Secure Protocol for Wireless Sensor Networks

Bo Sun[a]*, Chung-Chih Li[b], Kui Wu[c] and Yang Xiao[d]

[a]Dept. of Computer Science, Lamar University,
Beaumont, TX, USA 77710, bsun@cs.lamar.edu

[b]Dept. of Computer Science, Lamar University,
Beaumont, TX, USA 77710, licc@hal.lamar.edu

[c]Dept. of Computer Science, University of Victoria
BC, Canada V8W 3P6, wkui@cs.uvic.ca

[d]Dept. of Computer Science, The University of Memphis,
Memphis, TN, USA 38152, yangxiao@ieee.org

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. From the cryptanalysis point of view, our building block is considered secure if the attacker cannot obtain the pseudo-random numbers generated by the LCG. The Plumstead's inference algorithm for a LCG with unknown parameters demonstrates that it is impossible to significantly enhance the security of the system simply by increasing the size of the modulus. Therefore, 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. We also adopt the Hull and Dobell's algorithm to select proper parameters used in the LCG. The analysis of our cipher indicates that it can satisfy the security requirements of wireless sensor networks. We further 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, a commonly used cipher in wireless sensor networks, in terms of the number of basic operations.

Keywords - Wireless Sensor Networks, Linear Congruential Generator, Security

1. Introduction

Wireless Sensor Networks (WSNs) are usually built with a large number of inexpensive, small, and battery-powered devices. They have been used for a wide variety of applications such as environment monitoring, health monitoring, military sensing and tracking, etc [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 wireless sensor networks. However, given the stringent constraints on processing power, memory, bandwidth, and energy consumption, it is very difficult to design suitable secure mechanisms for wireless sensor networks. For example, Mica2 Motes [2] consist of an 8 MHz 8-bit Atmel ATMEGA128L CPU with only 4KB of RAM space for data, 128KB of program memory, and 512KB flash memory. This leaves very limited resources for the necessary security components in WSNs.

The constraints posed by the sensor hardware make it impossible to deploy most of the traditional security primitives and protocols. For example, it is too expensive to apply asymmetric cryptography to wireless sensor networks, such as the RSA [24] and Diffie-Hellman algorithm [25], because they require expensive computations and long messages that could easily exhaust the sensor's resources. Sym-

*Corresponding author

metric cryptography can be used in wireless sensor networks. Some popular symmetric encryption and hashing function schemes include RC5 [21], MD5 [26], SHA1 [27], Skipjack [28], and many existing security protocols for WSNs are based on these schemes. For example, SPINS [12] used RC5 as the block cipher. TinySec [19] used Skipjack as the default block cipher. However, a close look at the security of WSNs demonstrates that most of them only focus on the existing encryption primitives in order to construct the secure protocols. Therefore, 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. Instead of focusing on the potential performance improvement of security protocols with existing block ciphers, we aim at proposing a more lightweight block cipher that is suitable for wireless sensor networks. 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) [31]. In theory, cryptosystems based on a pseudo-random number generator (PRNG) (for example, a LCG) are not suggested because they are predictable [5]. 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 [7], 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. We also adopt the Hull and Dobell's Theorem [38] to select the proper parameters of the LCG. In this way, 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 [12]. Analytical results demonstrate that our proposed block cipher is more lightweight than RC5.

1.1. Related Work

There are two aspects of related work: security in wireless sensor networks and security analysis of LCGs.

Many research efforts have been devoted to security is wireless sensor networks. Perrig *at al.* [12] provided a suite of security building blocks that are optimized for resource constrained wireless sensor networks - SNEP and μTESLA. Liu. *et. al* [29] proposed an efficient distribution of key chain commitment for μTESLA. Hu *et. al* [30] studied the secure aggregation problem if one node is compromised. Park *et al.* [23] proposed LiSP - an efficient lightweight protocol that makes a trade-off between security and resource consumption. Karlof *et al.* [19] presented the fully-implemented link layer security architecture for wireless sensor networks.

There is also much work devoted to the key distribution and management in wireless sensor networks [12] - [17]. Our work depends on the key predistribution protocol to set up the initial shared secret between sensor nodes.

That all sequences generated by the LCG are predictable was first argued by Knuth [5]. Boyar [7] gave a rather complete treatment on the predictability of some of the widely used LCGs. Krawczyk [8] 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 [9], 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. We have also presented a LCG-based encryption protocol for email encryptions [33].

The rest of the paper is organized as follows. Section 2 presents the goals of our proposed security protocol. In Section 3, we introduce the keying mechanisms in developing the lightweight security protocol for wireless sensor networks. In Section 4, we present our LCG-based security protocol in detail. In Section 5, performance of the proposed block cipher is presented and compared with RC5. Finally, in Section 6, we conclude the paper and point out future work.

2. Security Goals

We focus on the following basic security goals:

- *Confidentiality*: Many applications of WSNs, such as military monitoring, require secured sensor readings/data so that they cannot be disclosed to attackers. This is also one of the goals for our security protocol. *Confidentiality* is typically achieved through encryption. Another stronger requirement is *semantic security*, which ensures that an adversary has no information about the plaintext, even if it sees multiple encryptions of the same plaintext [32]. A basic technique to achieve *semantic security* is to use randomization. We will show that our protocol also achieves *semantic security* later.

- *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. By achieving authenticity, we can prevent some third party from injecting falsified messages into the network.

3. Assumption

We assume that wireless sensor nodes are constrained in resources. We assume that every sensor node has space to store several hundreds of bytes of keying information. We do not put any assumption on the time synchronization of sensor nodes.

We assume that an adversary can eavesdrop on all traffic. Therefore, the adversary can perform cryptanalysis to deduce the secret.

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. Any communication can be encrypted and authenticated using this key. The advantage is its simplicity. However, the compromise of any single node can paralyze the whole network because the adversary can eavesdrop on traffic and inject falsified messages anywhere in the network.

A more robust approach is for groups of neighboring nodes to share a key. That is, a key is *locally shared* by a node and its neighbors. In this keying mechanism, a compromised node can only decrypt the messages from nodes in its own group. It cannot decrypt messages from and 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. However, how to set up pairwise keys on the fly is a non-trivial task.

We assume that there exists a key management subsystem that makes it possible for wireless sensor network 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 [12] - [17]. They could be utilized to provide a security service to our protocol. Based on the key pre-distribution protocol, each sensor node could share a secret key with other nodes when necessary. As we will demonstrate later, the key in our context is the set of parameters used by a LCG.

4. LCG-based Security Protocols

To provide a hop-by-hop guarantee on confidentiality, integrity, and authenticity of data messages, the processing overhead incurred by encryption primitives is the main concern. At the same time, the introduction of security should not incur expensive energy consumption. This is especially important given the extremely limited processing capability of sensor hardware. A large overhead will inevitably increase the processing delay and consume more energy. Therefore, it is necessary to design a fast and secure building block that can be used in wireless sensor networks.

4.1. 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. While a rigorous mathematical definition for true randomness is still an open research topic, 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 [3] shows that they are all proven to be cryptographically insecure. Let's take Linear Congruential Generators (LCG) as an example. Most commercial LCGs do not intend to be used for cryptographic purposes [3] [4]. 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. In conclusion, LCGs are cryptographically insecure in the sense that an attacker can practically recover the entire sequence with a limited observation on the sequence.

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 Linear Congruential Generator (LCG) in its simplest form to produce pseudo-random numbers. The reason that we select the LCG is because it is the simplest, most efficient, and a well-studied pseudo-random number generator.

4.2. Linear Congruential Generators

The simplest form of a Linear Congruential Generator (LCG) uses the following equation:

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

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 of the LCG. The statistical properties of the pseudo-random numbers generated by a LCG depend on the selection of its parameter [5].

4.2.1. 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 [7] against the LCG in its easiest form as shown in Equation (1).

The Plumstead's algorithm [7] is intended to discover the hidden parameters of an unknown LCG by observing its outputs as shown in Fig 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.

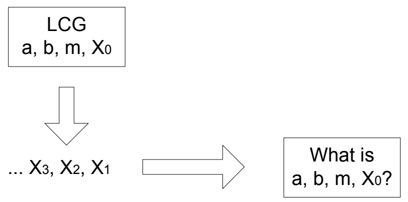

Figure 1. Plumstead's Algorithm.

4.2.2. Plumstead's Algorithm

Assume that 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 that 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 [7] for a detailed proof.

4.2.3. 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 bounded by $2 + \log_2 m$. Therefore, the algorithm is optimal with a sample complexity $O(\log_2 m)$ in the worst case.

4.2.4. Empirical Results of Plumstead's Algorithm

We carried our experiments to measure the impact of m on the security performance of the LCG. 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 [10], 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 1, in which μ is the average number of samples needed to successfully infer the pseudo-random number sequence while δ is the standard deviation. The theoretical analysis of the Plumstead's algorithm is based on the worst case. In reality, however, the worst case rarely occurs. 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 1 contains the best case (min) and the worst case (max) for each size. The values of δ in Table 1 indicate that the worse case occurs rarely.

Based on the results illustrated in Table 1, we can see that the size of m does not prolong the inference process significantly. This is because, from the theoretical point of view, the size of m does not affect the number of internal states [4]. 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 1, 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.

4.3. Key Selection

Based on the results illustrated in Table 1, the moduli that 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, as we show later. 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 $\frac{1}{\ln 2^{128}} = 0.0127$. Here, ln is the natural logarithm whose base is e. Therefore, on average we can successfully pick up a prime within about 100 random selections. Then, we randomly assign numbers less than m to X_0 without further imposing any restriction except for some trivial values such as 0 or 2^k. There is no concern about the size of the cycle in the sequence generated, since a 16-byte prime as the modulus is very likely to generate unrepeated numbers within the length of a regular data message, which is usually short in WSNs.

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, though, in order to achieve the maximum security using the LCG. In this respect, we apply Hull and Dobell's Theorem [38] as follows.

Table 1
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

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. That is, 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 m is divisible by 4, then $(a-1)$ is divisible by 4.

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 set up a system that chosen-plaintext attack cannot 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 predistribution is relatively easier. For example, the Blom key predistribution scheme [39] 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.* [14] 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 5. 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.

4.4. Basic Hop by Hop Message Transmission

In this section, we introduce our secure data transmission scheme. Our scheme makes use of an efficient and lightweight LCG-based approach. Compared to traditional approaches, our approach is more lightweight and more suitable for wireless sensor networks. We remedy the theoretical fault of the LCG and make our block cipher satisfy the security requirements in WSNs.

In the following, we use secure data aggregation [1] as an example to illustrate the operations of our LCG-based security protocol. The reason we choose data aggregation as an example is because it has been widely used to reduce the amount of traffic to the sink node. Nevertheless, our proposed security mechanism is general enough and is not limited to data aggregation only.

We use the following notation to describe our security protocol and cryptographic operations:

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

X_0: *Seed* of the LCG

a, b, m: Parameters of the LCG. They are open and could be treated as the parameters of the WSNs.

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

The overall scheme is illustrated in Fig. 2.

In Fig. 2, sensor nodes, such as nodes A, B, C, and D have monitored some events and transferred 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. Note that K_{AH}, K_{BH}, K_{CH}, and K_{DH} can be decided based on different keying mechanisms, as discussed in Section 3. After H receives the readings, it uses the corresponding secret to decrypt and authenticate the received messages. It then 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.

This transmission scheme is rather standard. We can see, one of the most critical parts in this process is how to efficiently design the encryption $E(P, K)$ and compute the MAC $MAC(P, K)$.

Results illustrated in Table 1 indicate that the Plumstead's algorithm can correctly predict the entire sequence with a few numbers in the sequence. In most cases, five or six numbers are enough. This will prevent a direct embedding of the numbers in the data message, because otherwise some classical cryptanalysis such as the *dictionary-attack* [31] can be used to provide enough information for inference if the adversary possesses enough amount of possible messages in his dictionary. For example, in a typical application of forest fire detection, the data message is used to transmit the temperature change from that of last moment. In this case, if the encoding format is known to the adversary, the elements in his dictionary could be all possible changes of the temperature. Since all these possible changes could repeatedly appear in the data messages, the attacker could try to match them in the dictionary and recover enough amount of pseudo-random numbers to infer the parameters of the LCG that generate them. This problem is considered in **Step 3** of our encryption scheme illustrated in the following.

In Fig. 2, the actual message format is $E(P|MAC(P, K), K)$. One alternative is $E(P, K)|MAC(P, K)$. We will compare them in Section 4.4.2.

4.4.1. Message Encryption

The goal of encryption is to prevent an attacker from recovering all the random numbers (i.e., X_i) generated by the LCG, and thus keep the plaintext message secure. We assume that the message to be encrypted is delimited into segments of 1 byte as we find that it is typical in the context of wireless sensor networks.

The encryption depends on the underlying block cipher. One general requirement is that the block cipher should be secure and lightweight for wireless sensor networks. It cannot involve too many complex multiplications and modulo operations because they are too expensive. 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**). Specifically,

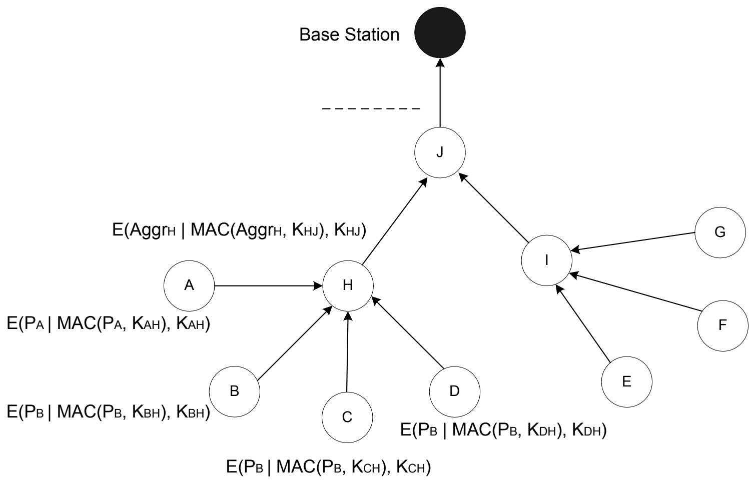
Base Station
J
I
G
F
E
H
A
B
C
D
E(AggrH | MAC(AggrH, KHJ), KHJ)
E(PA | MAC(PA, KAH), KAH)
E(PB | MAC(PB, KBH), KBH)
E(PB | MAC(PB, KCH), KCH)
E(PB | MAC(PB, KDH), KDH)

Figure 2. Hop By Hop Security Protocol.

for a 16-byte packet, once the parameters (a, b, and m) and the seed (X_0) of our LCG are decided, the LCG generates X_1 and the block cipher embeds each byte of the random number in the packet. We also introduce the noise permutation to further scramble results. The general strategy of using our proposed block cipher to encrypt a 16 byte packet is illustrated in Fig. 3:

- **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.

- **Step 2 - Stage I**: Suppose that 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

  $Wireless\ sensor$

  be the message to be encrypted. So $p_1 = Wireless$, and $p_2 = \ sensor\ $. Note that there is a space at each side of p_2. There are 16 bytes in the message, and they can be encrypted using one block cipher.

  The first three characters of p_1 are $W = 87$, $i = 105$, and $r = 114$. The embedding operations are simply the addition modulo 256. A more complicate operation is not necessary in this step. 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:

  $87 + 16 \quad mod\,256 = 103$

  $105 + 90 \quad mod\,256 = 195$

  $114 + 251 \quad mod\,256 = 109$

  As illustrated in Fig. 3, C_1, and C_2 are the scrambled text after X_1 is embedded. Each C_i is also 8 bytes.

- **Step 3 - Permutation**: X_1 is broken into 16 1 byte random numbers (Because X_1 is a 16-byte random number). We use $B_0, B_1, \ldots, B_{15}$ to denote them 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:

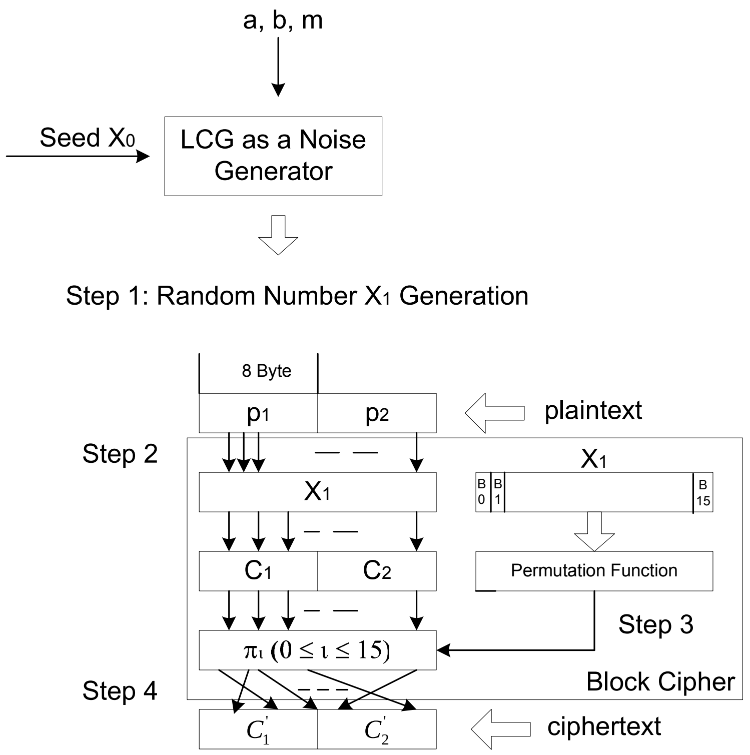

Figure 3. Message Encryption of a 16 byte Packet.

1. $\pi_0 = B_0 \, mod \, 16$;
2. $\pi_i = (n \; mod \; 16)$, for $i = 1 \ldots 15$ with n is the smallest integer such that $n \geq B_i$ and $\pi_i \nsubseteq \{\pi_0, \pi_1, \ldots, \pi_{i-1}\}$.

- **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. Through the key distribution protocol, the receiver node has obtained the same parameter (a, b, m) and the seed (X_0) of the LCG. Therefore, the receiver node could generate the same X_1 that the sender generates. Based on X_1, the receiver can recover the same permutation function. It can then recover C_1 and C_2. Finally, the receiver can obtain p_1 and p_2.

4.4.2. Security Analysis

We use the basic goals discussed in Section 2 - confidentiality, authenticity, integrity to analyze our proposed security protocol.

- **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 [31] 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.

- **Authenticity and Integrity**:

  Authenticity and Integrity are the two basic requirements of any security systems. A MAC mechanism is used here to provide Authenticity and Integrity together.

  We use a Cipher Block Chaining (CBC) MAC to provide authentication and integrity. CBC-MAC is efficient and effective. It has proven that the CBC-MAC construction is secure if the underlying block cipher is secure [22].

  In [19], 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. Therefore, given a 4 byte MAC, an adversary has a 1 in 2^{31} chance in blindly forging a valid MAC for a particular message. Because the adversaries cannot determine off-line if a forgery will be successful or not, the adversary can only test the validity of an attempted forgery by sending it to an authorized receiver. Given a 19.2kbs channel in WSNs, one can only send 40 forgery attempts per second. Therefore 2^{31} packets would take about 20 months, which is not realistic in WSNs. Following the motivation in [19], we also use a 4 byte MAC in our protocol.

  The CBC-MAC using our proposed block cipher scheme is illustrated in Fig. 4. 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,

  As illustrated in Fig. 2, the final format of our transmitted message is $E(P|MAC(P,K),K)$, instead of $E(P,K)|MAC(P,K)$. The encryption computation in $E(P,K)|MAC(P,K)$ involves less operations because its encryption only operates on P, instead of $P|MAC(P,K)$. 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. Therefore, we choose $E(P|MAC(P,K),K)$ as our message format in transit. The encryption of $E(P|MAC(P,K),K)$ can provide one layer of protection for the MAC.

  An adversary can launch the *replay* attack by eavesdropping on legitimate messages sent between two nodes and then replays it at some later time. A common approach is to include a monotonically increasing counter shared by two nodes with the transmitted message. The use of the counter could also help to achieve *semantic security*. However, a counter exchange protocol is necessary to tackle the counter synchronization problem. A replay table is also needed to keep the last value from every sender. Some research efforts [12][19] have discussed

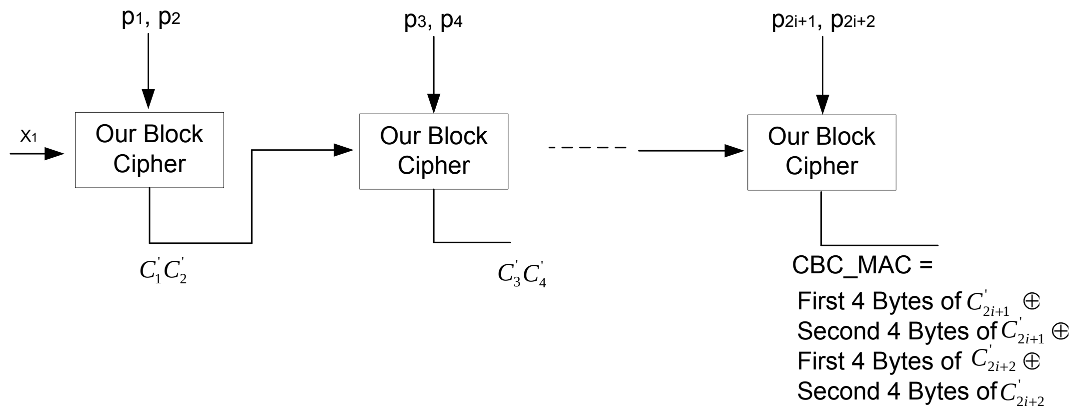
p1, p2
p3, p4
p2i+1, p2i+2
X1
Our Block Cipher
Our Block Cipher
Our Block Cipher
$C'_1C'_2$
$C'_3C'_4$
CBC_MAC =
First 4 Bytes of C'_{2i+1} ⊕
Second 4 Bytes of C'_{2i+1} ⊕
First 4 Bytes of C'_{2i+2} ⊕
Second 4 Bytes of C'_{2i+2}

Figure 4. Integrity and Authenticity.

this aspect before, we do not cover its details here.

Another approach to achieve *semantic security* in our context is an efficient *rekeying* mechanism. In Fig. 3, C'_1 and C'_2 can be assigned as the new seed (X_0). That is, after the transmission of the current data message, X_0 is changed to $C'_1C'_2$ at both the sender and the receiver. 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. In this way, *semantic security* can be achieved. This could also help to prevent the replay attack because bogus messages will be discarded if the computed MAC and the received MAC do not match.

Nevertheless, if an environment suffers from a high rate of message lost and collision, key synchronization will become a serious problem. In such an environment, we can reduce the frequency of rekeying operations to avoid the potential huge number of key synchronization operations. To prevent potential Denial-Of-Service (DoS) attacks, that is, an adversary can keep sending bogus messages to trigger the nodes into performing key synchronization, the nodes can use the similar method as in [12] by sending the keys with each encrypted messages they send.

There is also one trade-off between the communication overhead and the key synchronization: We can add 1 Byte overhead to the data packet. This byte is used as the *counter* and it is transmitted with every data packet. Given a harsh environment, the packet delivery ratio could be very low. However, it still has a very low probability that consecutive 256 packets are lost. Therefore, when the receiver receives the counter, it can compare it with its own stored counter. The difference between the received counter and the stored counter could help the receiver to decide what X_i is used for the current data packet. In this way, a key synchronization could be achieved.

In this way, we can simply delete the CRC of the original message.

4.4.3. Discussion

Our cipher is designed based on the following belief: while the pseudo-random numbers are generated to protect the message, the entropy of the message itself can in turn protect the pseudo-random numbers. Thus, if the message sent out from the sensor is almost flat, i.e., with very low entropy, our encryption in Step 2 alone is insecure since too many random bits can be recovered and, consequently, the size of the possible key space will be largely reduced. For this reason, we introduce the permutation in our cipher in Steps 3 and 4 to guarantee that even if our cipher is applied to a low entropy environment, the security of our cipher will not be significantly compromised.

Moreover, the encryption in Step 2 alone cannot resist *known-plaintext attack* in case the message in a sequence of transmitted packages is known to the adversary. To fix this problem, 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.

So far, we are not aware of any known-plaintext attack against our proposed block cipher. To our knowledge, even a plaintext-ciphertext pair is given, there is no easy way to separate the two factors, noise and permutations, involved in the ciphertext. Likewise, a direct ciphertext analysis does not seem possible.

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. Also, our scheme has a potential to integrate an efficient rekeying scheme - X_i could be changed for each transmission. In this kind of situation, chosen-plaintext attack does not apply.

We can see the size of our block cipher is 16 bytes. For wireless sensor networks, most data messages are usually small. So for a data packet that is less than 16 bytes, we need to pad it and apply our block cipher. For a message that is larger than 16 bytes, one approach called *ciphertext stealing* [31] can be used to ensure that the ciphertext has the same length as the underlying plaintext. Encrypting data payloads of less than 16 bytes will produce a ciphertext of 16 bytes because ciphertext stealing requires at least one block of ciphertext [31]. Similar approaches have also been used in [19]. 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) [19]. 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.

5. Performance Analysis

The amount of computational energy consumed by a security function on a given microprocessor is primarily determined by the power consumption of the processor, the clock frequency of the processor, and the number of clocks needed by the processor to compute the security function. We assume that energy consumption cannot be significantly reduced via a reduction in clock frequency. The cryptographic algorithm and the efficiency of the software implementation determine the number of clocks necessary to perform the security function [34]. 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.

Thus, we calculate the *Number of Basic Operations* of our cipher and compare it with RC5, which is one of the most popular and efficient block ciphers that is widely used in wireless sensor networks. 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 [21]. The key-expansion routine is the most time-consuming part in running the RC5 algorithm. Because most wireless sensor networks that adopt RC5 as their underlying cipher require the S-Table to be computed in advance to speedup their sensor's operation, we do not consider the cost of computing the S-Table in our analysis.

To make our comparison plausible, 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 (in fact, we need a bitwise AND). In Rivest's algorithm, "shift B bits

($\lll B$)" means "shift ($B\ mod\ 32$) bits". Thus, there is a bitwise AND involved. Also, we use 16 as the average value of ($B\ mod\ 32$). For RC5, we assume that 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. This is because each operation will be broken into 4 operations plus 4 store operations. This may be oversimplified since some necessary bookkeeping such as handling the carry bits may be required, but we ignore them for simplicity.

We restrict the multiplier a and modulus m to

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

Therefore, the modulus operation can be reduced to basic operations as follows:

Let $a_{63} \cdots a_i \cdots a_2 a_1 a_0$ be the bit string representing a. Equation (1) is equal to the following:

$$aX_n + b\ \ mod\ \ m = (aX_n\ \ mod\ \ m) + b\ \ mod\ \ m.$$

Because

$$(2^{i+1} X_n\ \ mod\ \ m) = ((2X_n\ \ mod\ \ m) \times 2^i)\ \ mod\ \ m),$$

and

$$\begin{aligned} aX_n\ \ &mod\ \ m \\ = &\ (a_{63} \cdot 2^{63} \cdot X_n + \cdots + a_i \cdot 2^i \cdot X_n + \cdots + \\ &\ a_0 \cdot 2^0 \cdot X_n)\ \ mod\ \ m \end{aligned}$$

the value of ($aX_n\ \ mod\ \ m$) can be computed by the following algorithm using the following basic operation shifts, additions, and subtractions:

```
1:input $X_n$, $a$, $b$, $m$;
2:   $X_{n+1} \leftarrow 0$;
3:   for ($i = 0$ to 63)
4:      if ($a_i = 1$)
5:         $X_{n+1} \leftarrow mod(X_{n+1} + X_n, m)$;
6:      $X_n \leftarrow mod(2X_n, m)$;
7:   end for;
8:   $X_{n+1} \leftarrow mod(X_{n+1} + b, m)$;
9:output $X_{n+1}$;
```

Note that $2X_n$ needs a shift-left operation. Moreover, since we choose $2^{127} < m < 2^{128}$, it follows that both $X_{n+1} + X_n$ and $2X_n$ are less than $2m$. Thus, each of the two modulus operations can be done by one subtraction.

We assume that X_n and X_{n+1} remain in the registers, and in average a half of the bits in a has value 1. The average number of basic operations performed by the algorithm above approximates to 197 according to the following analysis.

- Line 5: performed 32 times, each time includes one addition and one subtraction.
- Line 6: performed 64 times, each time includes one shift and one subtraction.
- Line 8: performed once, with one addition and one substraction involved.
- In addition, we need to fetch four parameters (X_n, a, b and m) and store one number (X_{n+1}).

5.1. Results

5.1.1. Our Cipher

The message encryption of our mechanism involves two parts: the block cipher (Step 2, 3, and 4) and the generation of the random number X_1 (Step 1). We analyze their number of basic operations respectively.

The breakdown of the number of the block cipher's basic operations is illustrated in Table 2. Our block cipher involves one 128-bits XOR, a Z_{16} to Z_{16} permutation construction, and its application to 16 bytes. In our experiments, we need less than 30 8-bit comparisons and 30 8-bit additions (the addition is just for a while loop index) to construct the permutation in average. We consider an 8-bit comparison as an 8-bit XOR. Of course, we need 16 8-bits fetches and 16 8-bits stores for applying the permutation. Note that, all operations involved in the permutation part are byte-oriented. Thus, no overhead will increase if we implement the algorithm in an 8-bit processor.

In Table 2, the first column is the name of the basic operations. The second column is the number of basic operations needed for a 16 bytes block on a 128-bit processor. This is the ideal case for our block cipher because the size of our block cipher is 16 bytes and the maximum size of the LCG parameters (a, b, m, and X_i) is 128 bits. Because of the popularity of 8-bit Atmega in Berkeley Motes, we also calculate the number of basic operations for a 16 bytes block on an 8-bit processor, as illustrated in the third column. The number of basic operations for a 16 byte packet and a 32 byte packet are illustrated in the fourth and fifth column, respectively. Please note that the number of operations presented in Fig. 2 does not include the operations to generate random numbers.

The breakdown of the number of basic operations of the random number generation is illustrated in Table 3. In our algorithm, one 16 byte LCG random number is needed for each packet to be transmitted. This should be considered as the counterpart of the key expansion stage in RC5. Although a LCG is very efficient compared to the key expansion routing in RC5, we should take its computing cost into account because we ask each sensor node to compute these keys dynamically for each packet (not for each block). It is clear that the most costly operations in the LCG algorithm are the general 128-bit multiplication and 128-bit modulo. Each, as we discussed earlier, is equivalent to 64 128-bit additions and 64 128-bit shifts in average. To generate a LCG random number, we need four fetches, one store, one multiplication, one addition, one modulo, and each performs on 128-bit data.

In Table 3, the first column illustrates the basic LCG operations involved in the random number generation. The second column illustrates the number of corresponding LCG operations. The fourth column lists the corresponding number of the basic operations for a 16 byte block on a 128-bit processor. The fifth column lists the corresponding number of the basic operations for a 16 byte block on an 8-bit processor.

5.1.2. RC5

The breakdown of the number of RC5's basic operations is illustrated in Table 4. Here we do not count the computation overhead of the S-Table's precomputation. Table 4 follows the same format as illustrated in Table 2. As we can see, for RC5, we need less than 17K basic operations to encrypt a 32-byte packet on an 8-bit processor.

5.2. Put Them All Together

Table 2, 3, and 4 illustrate the breakdown of using our proposed block cipher and RC5 as the building block to encrypt a packet. Combine these three tables together, we obtain Table 5, which depicts the comparison of the number of basic operations to encrypt a packet at different sizes.

Table 5 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.

6. Conclusions and Future Work

6.1. Conclusions

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

6.2. Future Work

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

We also plan to integrate our proposed protocol with other existing WSN applications. Some emerging new classes of applications (for example, reprogramming or "re-tasking" of groups of sensors) require reliable data delivery. The data that flows from sinks to sources for the purpose of control or management is sensitive to message loss. In this situation, existent reliable transport protocols for WSNs (for example, Pump Slowly Fetch Quickly (PSFQ) [37]) can

Table 2
Numbers of Basic Operations in LCG-based Cipher

Operation	128-bit Processor 16 byte Block	8-bit Processor 16 byte Block	8-bit Processor 16 byte Packet	8-bit Processor 32 byte Packet
Addition	0	0	0	0
XOR	31	62	62	124
Shift	0	0	0	0
Fetch	31	62	62	124
Store	31	62	62	124
Total	93	186	186	372

Table 3
Numbers of Basic Operations for Generating One 128-bits LCG Pseudo-Random Number

LCG Operations	Number of LCG Operations	Equivalent Operations	128-bits processor 16 byte block	8-bit processor 16 byte block
128-bit Addition	1	Addition	129	4128
128-bit Shift	0	Shift	64	2048
128-bit Fetch	4	Fetch	4	128
128-bit Store	1	Store	1	32
128-bit Multiplication	1			
128-bit Moduli	1			
Total	8		197	6304

be used together with our rekeying mechanism to enhance the security. That is, PSFQ could help our secure protocols to dynamically adjust the correct key for the current data transmission.

When there is no existence of a reliable transport protocol, we can adaptively adjust the mechanism of our proposed scheme according to existing services. In different environments of WSNs, the communication quality can vary dramatically over time. Generally, many of the links are lossy. The loss rate may change dynamically with environmental factors (the network topology, the physical layer coding scheme, the network topology, etc.) [35] or due to the contention arising from the highly correlated behavior of the application. Error rates experienced by these wireless networks can vary widely. In this kind of situation, existing work on the wireless link evaluation can not only help data delivery services select a more reliable path, but also help in our rekeying mechanism - reducing the possible synchronization operations for rekeying. Blacklisting and routing using a metric that reflects path reliability can also help eliminate unreliable, lossy, or asymmetric links from the set of links used for communications [36]. All these techniques could also provide help to the rekeying mechanism.

We expect our proposed secure protocols to have a close relationship with different environments and different applications in order to achieve the maximum security. This is necessary given the application specific nature of wireless sensor networks.

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. Crossbow Technology Incorporation. http://www.xbow.com.
3. K. Entacher, “A Collection of selected Pseudo-random Number Generators with Linear Struc-

Table 4
Numbers of Basic Operations in RC5-32/12/X

Operation	32-bit Processor 8 byte Block	8-bit Processor 8 byte Block	8-bit Processor 16 byte Packet	8-bit Processor 32 byte Packet
Addition	50	400	800	1600
XOR	48	384	768	1536
Shift	384	3072	6144	12288
Fetch	40	320	640	1280
Store	2	16	32	64
Total	524	4192	8384	16768

Table 5
Numbers of Basic Operations in RC5-32/12/X and LCG-based Cipher

	8-bit Processor 16 byte Packet	8-bit Processor 32 byte Packet	8-bit Processor 64 byte Packet
RC5-32/12/X	8384	16768	33536
LCG-Cipher	6490	6676	7048

tures," TR 97-1, University of Vienna, Austria, 1997.

4. J. Heinrich, "Detecting a Bad Random Number Generator," Technical Report:CDF/MEMO/STATISTICS/PUBLIC/6850, University of Pennsylvania, 2004.
5. D.E. Knuth, "Deciphering a Linear Congruential Encryption," *IEEE Transactions on Information Theory*, vol. 31, no. 1, pp. 49-52, January 1985.
6. J. Boyar, "Inferring Sequences Produced by Pseudo-Random Number Generators," *Journal of the ACM*, vol. 36, num. 1, pp. 129-141, 1989.
7. 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.
8. H. Krawczyk, "How to Predict Congruential Generators," *Journal of Algorithms*, vol. 13, no. 4, pp. 527-545, 1992.
9. T. Ritter, "The Efficient Generation of Cryptographic Confusion Sequences," *Cryptologia*, vol. 15, no. 2, pp. 81-139, 1991.
10. D. Stinson, "Cryptography: Theory and Practice", Chapman & Hall, 2nd Edition, 2002.
11. S. Zhu, S. Setia, and S. Jajodia, "LEAP: Efficient Security Mechanisms for Large-Scale Distributed Sensor Networks", *Proceedings of ACM Conference on Computer and Communications Security(ACM CCS)*, Washington DC, pp. 62-72, 2004.
12. 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.
13. H. Chan, A. Perrig, and D. Song, "Random Key Predistribution Schemes for Sensor Networks", *IEEE Security & Privacy*, Oakland CA, pp. 197-213, May 2003.
14. W. Du, J. Deng, Y. Han, and P. Varshney, "A Pairwise Key Pre-distribution Scheme for Wireless Sensor Networks", *ACM Transactions on Information and System Security (TISSEC)*, vol. 8, issue 2, pp. 228-258, 2005.
15. L. Eschenauer, and V. Gligor, "A Key-Management Scheme for Distributed Sensor Networks", *Proceedings of ACM Conference on Computer and Communications Security(ACM CCS)*, Washington DC, pp. 41-47, 2002.
16. D. Liu, and P. Ning, "Establishing Pairwise Keys in Distributed Sensor Networks", *Proceedings of ACM Conference on Computer and Communications Security(ACM CCS)*, Washington DC, pp.

52-61, 2003.

17. S. Zhu, S. Xu, S. Setia, and S. Jajodia, "Establishing Pairwise Keys for Secure Communication in Ad Hoc Networks: A Probabilistic Approach", *11th IEEE International Conference on Network Protocols (ICNP)*, Atlanta, GA, pp. 326-335, 2003.
18. D. Stinson, *Cryptography: Theory and Practice*, Chapman & Hall, second Edition, 2002.
19. 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.
20. 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.
21. R. Rivest, "The RC5 Encryption Algorithm", *Proc. 1st Workshop on Fast Software Encryption*, 1995, pp. 86 - 96.
22. 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.
23. 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.
24. 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.
25. W. Diffie and M. E. Hellman, "New Directions in Cryptography", *IEEE Transaction on Information Theory*, IT-22:644C654, November 1976.
26. R. Rivest, "The MD5 Message-Digest Algorithm", IETF RFC 1321, April 1992.
27. D. Eastlake and P. Jones, "US Secure Hash Algorithm 1 (SHA1)",IETF RFC 3174, Sept. 2001.
28. Skipjack and KEA algorithm specifications. http://csrc.nist.gov/encryption/skipjack/skipjack.pdf, NIST, 1998.
29. D. Liu and P. Ning, "Efficient Distribution of Key Chain Commitments for Broadcast authentication in distributed sensor networks", *Proceedings of 10th Network and Distributed System Security Symposium (NDSS'03)*, San Diego, CA, February, 2003, pp. 263-276.
30. L. Hu and D. Evans, "Secure Aggregation for Wireless Networks", Workshop on Security and Assurance in Ad hoc Networks, Orlando, FL, January, 2003.
31. B. Schneier, "Applied Cryptography: Protocols, Algorithms, and Source Code in C", 2nd Edition, John Wiley & Sons, 1996.
32. S. Goldwasser, and S. Micali, "Probabilistic Encryption", *Journal of Computer Security*, vol. 28, pp. 270-299, 1984.
33. C.-C Li and B. Sun, "Using Linear Congruential Generators for Cryptographic Purposes", *20th International Conference on Computers and Their Applications*, New Orleans, LA, March, 2005.
34. D. Carman, P. Kruus, and B. Matt, "Constraints and Approaches for Distributed Sensor Network Security", NAI Labs Technical Report No. 00010, 2000.
35. J. Zhao, and R. Govindan, "Understanding Packet Delivery Performance In Dense Wireless Sensor Networks", *1st international conference on Embedded networked sensor systems (ACM Sensys'03)*, pp. 1-13, Los Angeles, CA, 2003.
36. O. Gnawalix, M. Yarvisz, J. Heidemanny, and Ramesh Govindan, "Interaction of Retransmission, Blacklisting, and Routing Metrics for Reliability in Sensor Network Routing", *The First IEEE Communications Society Conference on Sensor and Ad Hoc Communications and Networks (SECON'04)*, pp. 34-43, Santa Clara, Oct. 4-7, 2004.
37. C.-Y Wan, A.T. Campbell, and L. Krishnamurthy, "PSFQ: A Reliable Transport Protocol for Wireless Sensor Networks", *International Workshop on Wireless Sensor Networks and Applications (WSNA'02)*, pp. 1-11, Atlanta, GA, 2002.
38. D.E. Knuth, "The Art of Computer Programming", Vol 2: Seminumerical Algorithms, Addison-Wesley, 1969.
39. 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.