LEE: Light-Weight Energy-Efficient Encryption Algorithm for Sensor Networks

Nikos Komninos, Hamed Soroush, and Mastooreh Salajegheh
Algorithms & Security Group
Athens Information Technology
GR-190 02 Peania (Attiki), Greece
{nkom, hsor, msal}@ait.edu.gr

***Abstract*—Data confidentiality in wireless sensor networks is mainly achieved by RC5 and Skipjack encryption algorithms. However, both algorithms have their weaknesses, for example RC5 supports variable-bit rotations, which are computationally expensive operations and Skipjack uses a key length of 80-bits, which is subject to brute force attack. In this paper we introduce a light-weight energy-efficient encryption-algorithm (LEE) for tiny embedded devices, such as sensor network nodes. We present experimental results of LEE under real sensor nodes operating in TinyOS. We also discuss the secrecy of our algorithm by presenting a security analysis of various tests and cryptanalytic attacks.**

***Index Terms*—Feistel network, block cipher, wireless sensor networks, encryption, cryptanalysis.**

I. Introduction

Wireless sensor networks (WSN) have recently attracted many computer engineering and computer science researchers due to their wide range of applications ranging from surveillance and industrial control to inventory tracking and military. A wireless sensor network is made of tiny embedded systems which are employed in the processing of sensor data. These small devices are extremely constrained in terms of their basic components. They usually consist of sensors (for light, temperature, etc.), a low-power communication device (radio transceiver), small amount of memory and a power supply. Consequently, this unique set of resource constraints (such as finite on-board battery power and limited network communication bandwidth) results in very different design trade-offs than those in general-purpose systems.

Since WSN in-network packet transmission consumes more energy than computation, there has been a relatively rich body of work on designing communication protocols that are customized for use in wireless sensor networks; coping with WSN inherent limitations which make conventional schemes a poor fit for the newly emerged technology. However, the need for effective, energy-efficient computational algorithms should not be ignored. According to an observation made by Karlof and Wagner [1] WSNs will more likely ride Moores Law downward, i.e. instead of relying on the computing power to double every 18 months, we are bound to seek ever cheaper solutions leading to systems operating at a fixed performance point, rather than having a constant price for the system and improving performance over time.

As sensor networks are usually deployed in hostile environments, many of their applications require that data must be exchanged in a secure and authenticated manner. Being an essential part of any security architecture, efficient yet sufficiently secure cryptographic primitives can serve as a means of conserving resources, i.e. saving small amount of storage and consuming little energy. The necessary cryptographic primitives for WSN are block ciphers, message authentication codes (MACs) and hash functions. The already available hash functions (such as SHA1 or MD5) are relatively cheap and can be used in sensor nodes without significant overhead. In addition, message authentication codes can be constructed from block ciphers [2]. Therefore, we have focused on designing a secure and lightweight block cipher algorithm suitable for use in sensor network nodes.

In this paper, we introduce the design and implementation of a Light-weight Energy-Efficient Encryption Algorithm (LEE) that can be used in tiny constrained devices especially sensor nodes to provide security services (such as confidentiality) for the communications. This encryption algorithm does not use complicated operators or any s-Boxes. We have implemented our algorithm in both java and nesC, the programming language used in TinyOS [7], which is the de facto event-driven operating system for sensor networks.

The organization of the paper is as follows: In Section II, we briefly consider the suitability of currently available security primitives for the sensor-net platform. Section III discusses our design goals and rationale. In section IV we provide the details of our algorithm. Section V presents our security analysis. We provide the implementation results of our block cipher in section VI. Section VII concludes the paper.

II. Related Work

Public-key cryptography has long been considered inappropriate for wireless sensor networks because of its expensive computational operations [3]. Therefore, symmetric-key schemes which are either stream ciphers or block ciphers using modes of operation are more suitable schemes for these networks.

Although the fastest stream ciphers are faster than the fastest block ciphers [4] which can propose them as a candidate in resource-constrained environments, they have the drawback that the same IV value should not be used more than once in order to encrypt two or more different packets (otherwise the scheme will not be acceptably secure). This means that, a relatively long IV value should be employed, imposing an unacceptable packet overhead (at least 8 bytes in a usually 30-byte long packet of WSN). Since acquiring short IVs and accepting the occurrence of IV reuse violates the robustness of the security scheme [8] using block ciphers and applying proper modes of operation is the most suitable way of constructing the encryption primitive.

Investigating the suitability of the already proposed secure block ciphers is a prerequisite of designing a more efficient one for the sensor networks platform. Rijndael [5] stands out as a potential candidate, especially since it has been selected as the Advanced Encryption Standard (AES) by NIST. However, as pointed out in [6], its suitability for sensor-net platform is not obvious, due to the fact that its average performance on a range of standard platforms (and not the specific WSN platforms) has been higher comparing to other block ciphers. As a matter of fact, designers of TinySec [8], which is a basic security platform available in TinyOS, have not found AES a suitable cipher for sensor networks[1]. Furthermore, it ranks lower than other ciphers such as Skipjack [9] and RC5 [10] based on its performance on WSN nodes [6]. On the other hand, hardware implementations of AES - though expected to appear with the advent of IEEE 802.15.4 or ZigBee - might not be an appropriate choice for WSN due to cost considerations [6].

Skipjack is used in TinySec, SenSec [11] and TinyKeyMan [12], well-known security platforms for sensor networks. Although since its controversial declassification by NSA in 1998, Skipjack has resisted years of cryptanalysis, its short key length of 80 bits makes it susceptible to the exhaustive key search attack; leading to a security margin[2] of 2013. In order to make Skipjack stronger against this attack, SenSec Designers propose Skipjack-X[3], a scheme similar to DES-X originally devised by Rivest in 1984. However it has been shown that such an strategy is not a sound investment of memory making Skipjack-X not a proper replacement for Skipjack in WSN [6].

RC5 is a very flexible cipher parameterised in word size, number of rounds and key length. It has been used in different sensor network security schemes like [15] and [16] and proposed by the authors of TinySec as another proper candidate to be used in WSN. Although RC5 is faster than Skipjack it is patented. Furthermore, for good performance it requires the key schedule to be precomputed which uses extra bytes of RAM per key [8]. Another discouraging factor against the use of RC5 is that most embedded processors do not support the variable-bit rotation instruction like ROL of the Intel architecture [17], which RC5 is designed to take advantage of.

[1]The authors point out that AES has performed very slow in their initial experiments, though they have found later that an efficient implementation is possible. No such implementation has yet been made available.

[2]Suppose an attacker who could afford sufficient number of computations to break DES in 1982 can also afford sufficient number of computations to break a cipher C in year y. Thus, the security of cipher C in year y is computationally equivalent to the security of DES in 1982. y is called the security margin of C which can be derived from the following formula [13]: $y = 1982 + 30/23(k - 56)$ where k is the key length of C.

[3]Suppose K is the 80-bit Skipjack key, K_1 and K_2 are two 64-bit keys and P is the plain text. According to this scheme we have: $SkipjackX_{K,K_1,K_2}(P) = K_2 \oplus Skipjack_K(K_1 \oplus P)$. Although the apparent key length of the scheme is $80 + 64 + 64 = 208$, it has been shown that the effective key length of this scheme is 111 bits [6].

TABLE I
SENSOR NETWORK NODE HARDWARE SPECIFICATIONS

Platform	SmartDust	Mica	Mica2	Tmote
CPU(bits,MHZ)	8, 4	8, 4	8, 4	16, 8
Flash (KB)	8	128	128	48
RAM(KB)	0.512	4	4	10
Frequency(MHZ)	916	916	916	2400
Band width(kbps)	10	40	38.4	250

According to the survey and benchmark of block ciphers for wireless sensor networks presented in [6] which analyzes and compares the performances of a a wide range of ciphers based on code memory, data memory, encryption/decryption efficiency and key setup efficiency, skipjack -which is the default encryption primitive in TinyOS- is the best performer. Interested readers can refer to [6] for further information on the advantages and disadvantages of using other available block ciphers in WSN.

III. DESIGN GOALS & RANTIONALE

LEE has been designed with the following goals in mind:

- **Simplicity:** Since LEE is designed to be used on sensor network nodes, it has to cope with the limitations that exist on their platforms as well as the requirements of the various sensor-net applications. *High speed* and *small memory usage* as well as the *ease of correct implementation* are the major of these requirements. A simple design, aids to satisfy these requirements. Table I provides the various characteristics of the currently available hardware platforms for WSN nodes.
- **High Security Margin:** Obviously the cipher should provide strong resistance against exhaustive search attacks. A relatively large key size (128 bits), therefore, is chosen for LEE.
- **Acceptable Resistance Against Known Attacks**: We have tried to design LEE in a way that it blocks or represents strong resistance against known weaknesses and security attacks. LEE is specifically designed to avoid having equivalent keys and providing high confusion and diffusion.

LEE was designed with as simple as possible computational operations in mind, suitable for sensor devices. The small set of elementary operations that LEE uses makes it efficient on a larger number of software platforms. The absence of tables, variable rotations, and multiplications makes LEE small and efficient in hardware as well.

Similar to XTEA [18], the round function of LEE combines integer addition (ADD) and exclusive-or (XOR) operations alternately to provide nonlinearity. Most ciphers use lookup tables to provide the necessary non-linearity. In LEE the

TABLE II
NOTATIONS USED IN THE DESCRIPTION OF LEE

Symbol	Description
.	Concatenation
$\hookrightarrow$	Bitwise Rotation to Right
$\leftarrow$	Bitwise Shift to Left
$+$	Addition Modulo 2^{32}
$\oplus$	XOR

nonlinearity comes from the mixing of XORs and ADDs. Neither of these operations can be approximated well within the group of the other. In addition, we have combined fixed rotations and shifts operations at the beginning of each round so as to mix all bits of plaintext and key repetitively. While multiplication is an effective mixer in block ciphers, we have not considered them since efficient multipliers are complex and large in size, slow in speed, and consume much power. Moreover, the key scheduling technique uses fixed rotations and bitwise-and (AND) operations to mix the subkeys with some constant. Following this design rationale, LEE is defined in the next section.

IV. ALGORITHM SPECIFICATIONS

Motivated by XTEA [18], LEE is a 64-bit block Feistel Network with a 128-bit key and a suggested 32 rounds. The Feistel Function is based on fixed length rotatation and shift operations, XOR and addition modulo 2^{32}. Table II presents the operations notation that we use to describe our algorithm precisely.

In LEE, as in other Feistel type of ciphers, the plain text block is split into two halves, L_0 and R_0. Each half is used to encrypt the other half over 32 rounds of processing and then combine to produce the cipher text block. Therefore, the original input of the algorithm (i.e. plain text) is $P = L_0.R_0$ and the final cipher text is $C = L_{32}.R_{32}$. The relation between the output $L_{i+1}.R_{i+1}$ and the input $L_i.R_i$ for the ith round of LEE is thus defined as follows:

$$L_{i+1} = R_i$$
$$R_{i+1} = L_i \oplus F(R_i, S_i, \delta_i)$$

where function F operates as follows:
Let:

$$R_i^{'} = (R_i \leftarrow 4) \oplus (R_i \hookrightarrow 5)$$

then:

$$F(R_i, S_i, \delta_i) = ((R_i^{'} + \delta_i) \oplus R_i) + (\delta_i \oplus S_i)$$

S_i is the round subkey generated by the key scheduling algorithm for the ith round, and δ is a value determined by the round number and a constant Δ as follows:

$$\Delta = (\sqrt{5} - 1) * 2^{31} = 0x9E3779B9$$
$$\delta_i = \lceil (i-1)/2 \rceil * \Delta$$

The key schedule algorithm produces sub-keys S_i for the ith round of the encryption/decryption as presented in Algorithm 1.

Algorithm 1 Key Schedule

```
Require: 128-bit Cipher Key $K$
Ensure: The Sub-key used in the $i$th Round
1: Split the 128-bit key $K$ into four 32-bit blocks ($K = K_0.K_1.K_2.K_3$)
2: for $i = 1$ to 32 do
3:     if $i$ is odd then
4:         return $K_{\delta_i \& 3}$
5:     else if $i$ is even then
6:         return $K_{((\delta_i \hookrightarrow 11) \& 3)}$
7:     end if
8: end for
```

The decryption is essentially the same as the encryption process; in the decode routine the cipher text is used as input to the algorithm, but the sub keys S_i are used in the reverse order.

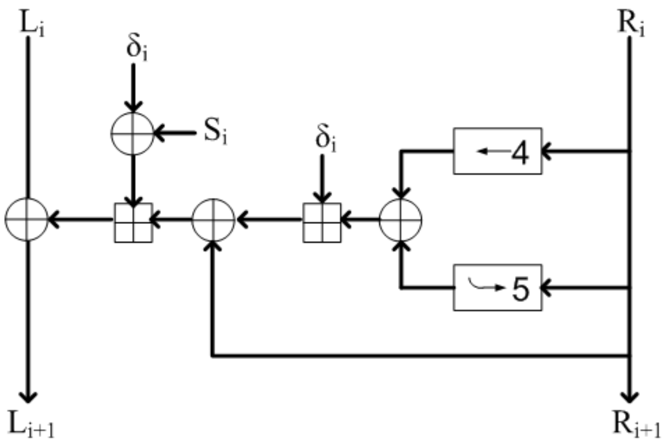

Fig. 1. LEE Stucture

Figure 1 shows the internal structure of the LEE cipher.

V. SECURITY ANALYSIS

In order to measure the secrecy of our block cipher and the level of *diffusion* and *confusion* that it provides, we have performed several tests. These tests are aimed at providing measures of

1) Independence of plain text and ciphertext
2) Mathematical properties of the algorithm
3) Avalanche effect
4) Cryptanalysis

In the following sub-sections we give a brief description of each test and provide the results of performing each test on our algorithm.

A. Independence of Plaintext and Ciphertext

The aim of the test performed in this section is to test the hypothesis that patterned plain text will produce random ciphertext blocks, using a fixed key. To test this hypothesis a large number of patterned plain text blocks is selected together with the corresponding ciphertext output blocks for a large number of fixed keys. Thus, the following tests are performed to the ciphertext:

- **Frequency Test**. The number of *n*-bit ciphertext blocks whose count of ones is as expected based on the chi-square, χ^2, distribution.
- **Binary Derivative Test**. The frequency test is applied to a new ciphertext formed by the exclusive-or of successive bits in the ciphertext. By taking the binary derivative of each of the ciphertext blocks to be examined, a set of ciphertext blocks of length *n*-1 are formed. The randomness of the binary derivative ciphertext blocks is compared using the χ^2, goodness-of-fit test.
- **Subblock Test**. Subblocks of *n*-bit length are chosen from each ciphertext block and determined whether ciphertext bits of different positions are independent or dependent between them.
- **Block Entropy Estimate**. The number of independent bits of the ciphertext block is estimated using the number of repetitions in the subblock test.

It was found that patterned plain text blocks were producing random ciphertext blocks, even when several *weak* keys were tested.

B. Mathematical Description of Block Cipher Properties

The tests described in this section are performed in order to estimate the strength of our block cipher. Note that by satisfying any of these properties, the strength of the block cipher is decreased.

- **Affine Property**. The method used to test for the affine property was performed by encrypting four known plain text blocks randomly chosen and by checking whether one derives from the other by single exclusive-or calculation.
- **Complementation Property**. The resulting complement of the ciphertext, which is derived from the complement of the plain text and key block, is examined with the original ciphertext.
- **Linear Relationship Property**. For a linear relationship to hold between subsets of plain text and ciphertext bits, then the sum of these plain text and ciphertext bits would always equal 0 or 1. The method of detection of this relationship involves a $(2n + 1) \times (2n + 1)$ matrix A. Twenty-one samples of 2*n*+1 plain text-ciphertext pairs of bit-vectors are concatenated and these vectors are placed in columns below an initial row of 2*n*+1 ones in *A*. If a row of all zeros exist, then the history rows involved in the row-reduction is kept. If a row of all zeros exists, then the history of the rows involved to obtain the result gives the bits involved in the resulting linear equation under exclusive-or operation. If linear equations exist for all twenty-one matrices then these equations are applied to the initial data to check that they are true for all input-output pairs.

The above tests were performed for more than 1GByte of plain text-ciphertext pairs and none of these properties hold for LEE.

C. Avalanche Effect

The plain text avalanche effect and the key avalanche effect were measured. By definition, a block cipher satisfies the plain text (key) avalanche effect if for a fixed key (or plain text) a small change in the plain text (or key) causes a large change in the resulting ciphertext block [10]. Having a strong level of the avalanche effect is desired, indicating a high level of diffusion.

In order to test the plain text-ciphertext avalanche criterion, we generated a 1GByte of random plain text blocks and using a large number of keys, including weak keys, we generated ciphertext vectors. The hypothesis that the entry of a particular ciphertext position changes was examined and it was found that the entry in the plain text (or key) position changes with probability $1/2$ for more than 4 rounds of LEE.

The distribution of the number of *changes* in ciphertext bits was also investigated and compared with the expected frequency of *changes* in the χ^2 distribution. The Kolmogorov-Smirnov statistic [18] was applied to determine whether the *changes* significance probabilities satisfy a normal distribution. For each plain text bit complemented, a record of the pairs of avalanche variables whose chi-squared statistic yielded a significance probability less than 0.01. It was found that a change in a bit in the plain text exhibits a random distribution of the changes in ciphertext.

D. Cryptanalysis

We have designed LEE with a view to reducing or avoiding any vulnerabilities that might arise from the following possible weaknesses and attacks.

- **Dictionary Attack**. As the block size is 64 bits, a dictionary attack will require 2^{64} different plain text blocks to allow the attacker to encrypt or decrypt arbitrary messages under an unknown key. This attack applies to any deterministic block cipher with 64-bit blocks regardless of its design. It is worth observing that with a cipher running at the rate of one terabit per second, or 10^{12} bits/second, the time required for 50 computers working in parallel to encrypt 2^{64} blocks of data is more than a year; to encrypt 2^{80} blocks of data is more than 98,000 years; and to encrypt 2^{128} blocks of data is more than 10^{19} years.
- **Modes of Operation**. After encrypting about 2^{32} plain text blocks in the CBC or CFB mode, one can expect to find two equal ciphertext blocks. This enables an attacker to compute the exclusive-or of the two corresponding plain text blocks. With progressively more plain text blocks, plain text relationships can be discovered with progressively higher probability. In addition, when the algorithm is used in feedforward mode as a hash function, a collision can be found with an effort somewhat more than 2^{32}. This attack applies to any deterministic block cipher with 64-bit blocks regardless of its design.
- **Key-Collision Attacks**. For key size *k*, key collision attacks can be used to forge messages with complexity only $2^{k/2}$. This attack applies to any deterministic block

cipher, and depends only on its key size, regardless of its design.

- **Linear Cryptanalysis**. To mount a linear cryptanalytic attack, there appear to be two different operations. The first might be to find a linear approximation over several rounds that uses a linear approximation across the shifts, rotations operations and δ_i values. Since there appear to be some very suitable linear approximations using the least significant bits, the bias of these approximations drops rapidly as more rounds are added, and the amount of data required for a successful attack exceeds the amount of data available. Moreover, it is possible to find a two-round iterative linear approximation that does not use an approximation across the combination of the shifts and rotations operations and δ_i values. Using basic but established techniques, we observed that the data requirements to exploit this appromixation over a version of LEE with 12 rounds is about 2^{130} known plain texts. This provided our rationale for choosing 32 rounds for LEE.
- **Higher Order Differential Cryptanalysis**. It is well known that a *d*th order differential of a function of nonlinear order *d* is constant, and this can be exploited in higher order differential attacks [18]. The fixed shifts and rotations in LEE have nonlinear order 3 so one would expect that the nonlinear order of the output bits after *r* rounds is about 3^r, with the maximum value of 127 reachable after 5 rounds. Therefore, we are convinced that higher order differential attacks are not applicable to LEE.
- **Truncated Differential Cryptanalysis**. For some ciphers it is possible and advantageous to predict only the values of parts of the difference after each round. This notion of truncated differential attack [18] seems best applicable to ciphers where all operations are done on larger block of bits. Because of the strong diffusion over many rounds, we believe that truncated differential attack is not applicable to LEE and we are under investigation to proof our claim.
- **Related Key Attack**. As the key schedule uses fixed rotations and as we exclusive-or the round number, into the subkey, it is highly unlikely that keys can be found that allow related key attacks [18]. Moreover, different rounds of LEE use different δ_i and subkey, so even if related keys were found, related-key attacks would not be applicable.
- **Other - Timing Attack**. The number of instructions used to encrypt or decrypt does not depend on either the data or the key, and even cache access cannot help the attacker as we do not access different locations in memory for different plain texts or keys. If follows that timing attacks [18] are not applicable.

VI. Experimental Results

We have implemented our encryption scheme in TinyOs [7] which is an event-driven operating system commonly used in WSN nodes (motes). A program written in nesC, the programming language used for TinyOS, consists of one or more reusable components assembled or *wired*, to form an executable application. This wiring mechanism is independent of the implementation of components; allowing each application to customize the set of components it uses. Consequently, unused components or services can be excluded from the application. We present the results of our experiments performed on both the internal TinyOS simulator called TOSSIM and on MICA2 motes.

High speed, as we mentioned in section III is an important requirement of any block cipher to be used in wireless sensor network applications. In link layer security platforms such as [8] block cipher operations have to be completed quickly in order to make the data needed for the radio available. This is of high importance since the cryptographic operations are overlapped with the radio operations (which are quite energy consuming). Furthermore, faster block ciphers consume less energy. Table III provides the time that it takes to perform a single cipher operation (encryption or decryption) using various rounds of LEE. A comparison to Skipjack and RC5 is also provided.

TABLE III
Time Comparison of available encryption algorithms in TinyOS

Algorithm Name	Time	Round
LEE	0.08s	8
LEE	0.125	12
LEE	0.14ms	16
LEE	0.27ms	32
LEE	0.54ms	64
Skipjack	0.18ms	32
RC5	0.20ms	12

Table IV provides a comparison of Skipjack, RC5 and LEE in terms of memory both in RAM and ROM on MICA2 sensor node platform. Note that the small difference of 2 bytes of RAM between Skipjack and LEE is due to the much smaller key-size of Skipjack. The relatively high security margin of LEE, therefore, comes with a relatively small cost of required RAM (comparing to Skipjack) which we believe is acceptable. In addition, LEE is still much smaller in size than RC5, another encryption candidate for WSN as we discussed in section II;

TABLE IV
Comparison of the Required Memory for Skipjack, RC5 and LEE on MICA2 nodes operating under TinyOs

Encryption Algorithm	ROM(Bytes)	RAM(Bytes)
Skipjack	17494	633
RC5	16128	723
LEE	16048	635

VII. Conclusion

In this paper we have introduced a new light-weight energy-efficient encryption algorithm (LEE) suitable to devices with limited computational power, such as sensor network nodes (motes). The results of the tests performed to the cipher imply that LEE reaches to acceptable levels of confusion and diffusion very fast. Furthermore, it uses larger secret key (128-bits) and requires smaller amount of memory comparing to the best known encryption candidates for sensor networks. We are under investigation to measure the performance of this cipher under newer sensor node hardware platforms.

References

[1] C. Karlof, D. Wagner, “Secure Routing in Wireless Sensor Networks: Attacks and Countermeasures”, *Elsevier’s Ad Hoc Networks Journal, Special Issue on Sensor Network Applications and protocols*, 2003, 1-3

[2] B. Preneel, V. Rijmen, “Cryptographic primitives for information authentication - state of the art”, *State of the Art in Applied Cryptography*, 1998

[3] D. Carman, P. Kruus, B. Matt, “Constraints and Approaches for Distributed Sensor Network Security”, Tech. Rep. #00-010, NAI Labs, 2000

[4] R. Venugopalan, P. Ganesan, P. Peddabachagari, A. Dean, F. Mueller, M. Sichitiu, “Encryption overhead in embedded systems and sensor network nodes: Modeling and analysis”, *International Conference on Compilers, Architectures and Synthesis for Embedded Systems*, 2003

[5] J. Daemen, L. Knudsen, V. Rijmen, “AES Proposal: Rijndael”, 1999

[6] check it out

[7] J. Hill, et al, “System architecture directions for networked sensors”, in *Proceedings of ACM ASPLOS IX*, 2000

[8] C. Karlof, N.Sastry, D. Wagner, “TinySec: Link Layer Encryption for Tiny Devices”, *ACM SenSys*, 2004

[9] NIST, “Skipjack and KEA Algorithm Specifications Version 2.0”

[10] R. Rivest, “The RC5 Encryption Algorithm”, *Proceedings of the 7th Annual International Conference on Mobile Computing and Networking*, 1995, Springer-Verlag, 86-96

[11] T. Li, H. Wu, F. Bao, “SenSec Design”, Institue for InfoComm Research, Tech. Rep. TR-I2R-v1.1, 2005

[12] D. Liu, P. Ning, R. Li, “Establishing Pairwise Keys in Distributed Sensor Networks”, *ACM Trans. Inf. Syst. Secur.*, 2005

[13] A. Lenstra, E. R. Verheul, “Selecting Cryptographic Key Sizes”, *Journal of Cryptology*, 2001

[14] J. Kilian, P. Rogaway, “How to Protect DES Against Exhaustive Key Search”, *CryptologyCRYPTO96*, 1996

[15] A. Perrig, R. Szewczyk, V. Wen, D. culler, D. Tygar, “SPINS: Security Protocols for Sensor Networks”, *ACM CCS*, 2003

[16] Q. Xue, A. Ganz, “Runtime Security Composition for Sensor Networks (SecureSense)”, IEEE Vehicular Technology Conference, 2003

[17] Intel Corporation, “Intel Architecture Software Developer’s Manual Volume 2: Instruction Set Reference”, 1997

[18] A. J. Menezes, S. A. Vanstone, and P. C. Van Oorschot, *Handbook of Applied Cryptography*, CRC Press, Inc., 2001.