An efficient scheme for key pre-distribution in wireless sensor networks

Manel Boujelben[1], Habib Youssef[2], Mohamed Abid[1]

[1]CES research unit, National school of engineering, Sfax, 3038, Tunisia
[2] PRINCE research unit, ISITC of Hammam Sousse, 4011, Tunisia

Wireless sensor networks (WSN) are the subject of widespread deployment in commercial and military environments that call for security. Since sensor networks pose unique challenges, traditional security techniques used in traditional networks cannot be directly applied. A previous work defines TinySec, a link layer security protocol for TinyOS to provide integrity and confidentiality of messages for WSN. However, TinySec employs a simple group key management where a single shared key is stored in each sensor node. This makes the network unsecured and vulnerable to attacks. Therefore, TinySec must be enhanced with a more robust key management scheme. This paper presents the integration of a proposed key management scheme as a more robust key management solution for TinyOS. The more secured TinyOS is evaluated in terms of key computation time, memory, and energy consumption overhead. Experimental results demonstrate that the secure key management protocol introduces negligible overhead and does not affect the system performance. We also present performances comparison with two well known key distribution protocols, implemented specially for sensor networks namely, ECDLP and LEAP.

***Index Terms*— Key pre-distribution, Security, Sensors networks**

I. INTRODUCTION

Research advances in Micro Electro Mechanical Systems (MEMS), highly embedded operating systems and wireless communications have enabled the realization of wireless sensor networks. Typically, WSN are composed of a large set (hundreds to a few thousand) of homogeneous nodes with extreme resource constraints [1]. Each node is equipped with a limited power unit which must aliment processing, sensing, storage, and radio communication units. These nodes are usually scattered over the area to be monitored to collect data, process it, and forward it to a central node for further processing.

The deployment of WSN is becoming more common in a wide range of applications ranging from home/health monitoring and remote environment observation to vehicle tracking and management of commercial inventory.

Security is one of the most difficult problems facing these networks. For certain applications of sensor networks, like military applications, security becomes very important. First, wireless communication is difficult to protect since it is realized over a broadcast medium. In a broadcast medium, adversaries can easily eavesdrop on, intercept, inject, and alter transmitted data. Second, since sensor networks may be deployed in a variety of physically insecure environments, adversaries can steal nodes, recover their cryptographic material, and pose as authorized nodes in the network. Third, Sensor networks are vulnerable to resource consumption attacks. Adversaries can repeatedly send packets to drain a node battery and waste network bandwidth. In these and other vital or security-sensitive deployments, secure transmission of sensitive digital information over the sensor network is essential. The use of encryption or authentication primitives between two sensor devices requires an initial link key establishment process, which must satisfy the low power and low complexity requirements.

Currently, the MICA2 mote devices represent the state of the art in wireless sensor networks technology based on commercial off-the-shelf hardware components [21]. They offer an 8-bit, 7.3 MHz ATmega 128L processor, 4 KB of primary memory (RAM) and 128 KB of program space (ROM) and 512 KB secondary memory (EEPROM), and a ChipCon CC1000 radio capable of transmitting at 38.4 Kbps powered by 2 AA batteries. Each MICA2 node has a default operating system called TinyOS. TinyOS is an open-source operating system designed for embedded WSN. It features a component-based architecture which enables rapid innovation and implementation while minimizing code size as required by the severe memory constraints inherent in sensor networks [20]. TinySec is a fully implemented protocol for link-layer cryptography in sensor networks. It is incorporated into the official TinyOS release and provides message integrity and confidentiality. However, TinySec employs a simple group key management, so if a sensor node is compromised by an adversary, overall sensor nodes in the network are likely to be compromised.

In this paper, we focus on the integration of a key establishment protocol called the "multiple space random key pre-distribution scheme" [2] within the TinyOS operating system. This protocol provides very good network resilience while respecting the WSN constraints. Therefore, it is integrated with the TinySec protocol to avoid relying on a single network shared secret key. This protocol implementation is evaluated and validated in terms of time for pairwise key generation, memory utilization, and especially in terms of energy consumption. Experimental results show that the security upgrade version of TinyOS performs better than

978-0-7695-3393-3/08 $25.00 © 2008 IEEE
DOI 10.1109/WiMob.2008.41

two well known key distribution protocols, implemented specially for sensor networks namely, ECDLP and LEAP.

The remainder of this paper is organized as follows. First, we discuss background and related work in Section 2. In Section 3 we present an overview of the multiple space random key pre-distribution scheme and we give a brief analyse of its security. Section 4 describes the implementation details. Section 5 presents experimental results. Finally, we conclude in Section 6.

II. RELATED WORKS

When setting up a sensor network, one of the first requirements is to establish cryptographic keys for later use (encryption and authentication). The inherent properties of sensor networks render previous protocols impractical.

Public Key Cryptography (PKC) such as RSA or Elliptic Curve cryptography (ECC) has been proposed for solving the problem of key distribution in WSN. However, it is unsuitable for most sensor architectures due to its high energy consumption and increased code storage requirements. Despite this fact, several researchers have been focusing on developing optimized implementations of PKC algorithms for sensor networks [3], [4].

KDC-based schemes are also proposed for WSN. They rely on the presence of a resource-rich Key Distribution Center (KDC) or base station in the network to act as a trusted arbiter for key establishment. Examples of such schemes include SPINS [5] and Kerberos [6]. Here, each node needs to share only a single key with the base station and sets up keys with other nodes through the base station. The memory resource load on the sensor nodes is low since the heavy burden is placed on the KDC. This arrangement makes the base station a single point of failure, but because there is only one base station, the network may incorporate tamper-resistant packaging for the base station, improving the protection against physical attacks.

Several alternative approaches relying on symmetric key cryptography have also been developed to perform key management on resource-constrained sensor networks. The simplest way is to let the network node share a single secret key. Unfortunately, the compromise of even a single node in a network would reveal the secret key and thus allow decryption of all network traffic. One variant on this idea is to use a single shared key to establish a set of link keys, one per pair of communicating nodes, and then erase the network wide key after setting up the session keys. However, this variant of the key-establishment process does not allow addition of new nodes after initial deployment. Yet another approach is the full pairwise scheme. This approach uses a shared unique symmetric key between each pair of nodes. Therefore, this scheme is memory-intensive and does not scale up. The memory overhead is n-1 cryptographic keys for every sensor node. Zhu et al. proposed a protocol named LEAP [7] to help establish individual keys between sensors and a base station, pairwise keys between sensors, cluster keys within a local area, and a group key shared by all nodes. To fully take advantage of the information available to the sensor networks, schemes using information from the environment were proposed. Deployment knowledge about the environment is frequently used for a more optimized design. For example, Du et al. proposed a key management scheme using deployment knowledge [14]. Liu et al. proposed location-based pairwise key establishment for relatively static sensor networks [15], with the prior knowledge obtained before distributing the sensor nodes. The memory usage per sensor is greatly improved while the connectivity of the sensor network is maintained.

Other recent work focused on using networks with a heterogeneous mix of nodes, and designating nodes with greater inherent capabilities and energy as cluster heads in order to maximize network lifetime [16], [17].

Blom [9] and Blundo et al. [10] proposed respectively a matrix and a polynomial key generation schemes. These schemes guarantee that any two nodes will be able to perform pairwise key, but each of these schemes also involves an $\Omega(n)$ high memory cost if we require that the system be secure against an adversary capable of compromising a fraction λ of the total number of nodes. The solution is λ secure, meaning that coalition of less than $\lambda+1$ sensor nodes knows nothing about pairwise keys of others.

Random key pre-distribution schemes represent another major class of key establishment protocols for WSN. Eschenauer and Gligor [12] proposed a probabilistic key pre-distribution technique. Each sensor node receives a random subset of keys from the key pool before deployment. Any two nodes able to find one common key within their respective subsets can use that key as their shared secret to initiate communication. Chan et al. further extended this idea and proposed the q-composite key pre-distribution [13]. This approach allows two sensor nodes to set up a pairwise key only when they share at least q common keys. Du et al. developed a pairwise key management scheme [2]. This scheme combines the random key pre-distribution scheme [12] and the Blom scheme [9] to substantially improve network resilience against node capture over existing schemes, without increasing the memory overhead. This scheme offers a good security level while respecting the resources constraints of sensor nodes. Therefore, this paper adopts this scheme as a key management solution to be used with TinySec protocol [8] and implements it to validate its efficiency and feasibility for WSN.

III. OVERVIEW OF THE MULTIPLE SPACE RANDOM KEY PRE-DISTRIBUTION SCHEME

Since Du et al. scheme [2] provides good network resilience with a low memory requirement; this paper focuses on the implementation of this protocol for WSN. Next, we briefly describe this scheme and discuss the reasons of this choice.

A. Blom's scheme

In [9], Blom proposes a key pre-distribution scheme that allows any pair of nodes to find a common secret between them. This scheme has a λ secure property: as long as an adversary compromises less than or equal to λ nodes, uncompromised nodes are perfectly secure; when an adversary compromises more than λ nodes, all pairwise keys of the entire network are compromised.

Compared to the full pairwise scheme, this scheme is efficient in terms of memory consumption: each node in a network of size n needs to store only λ+1 elements, with λ<< n, but it is not perfectly resilient against node capture. The size of an element is that of a secret key. To be more resilient, the security parameter λ must be high. However, this can have a big memory overhead. Du et al's scheme builds on Blom's scheme and combines it with the random key pre-distribution method. The goal of this scheme is to improve network resilience against node capture without increasing the memory size.

B. *Key Pre-distribution Phase*

In the key pre-distribution phase, we must select the parameters that decide on the level of security of this scheme: these parameters are λ, ω and τ ($2 < \tau < \omega$). At the end of this phase, each node must carry some information to be able to compute a key after deployment. This phase is composed of three steps:

Step 1 (Generating G matrix): To generate the matrix G of size (λ +1)×n, we must chose a primitive element from a finite field GF(q), where q is the smallest prime larger than 64 bits, the size of TinySec key, and q > n. Figure 1 shows an example of matrix G. Let s be a primitive element of GF(q). A node k in the network must carry the k^{th} column of G. Using this generator matrix each node will only carry the seed s^k.

$$\begin{pmatrix} 1 & 1 & 1 & 1 & 1 & 1 \\ s & s^2 & \dots & s^k & \dots & s^n \\ s^2 & (s^2)^2 & \dots & (s^k)^2 & \dots & (s^n)^2 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ s^\lambda & (s^2)^\lambda & \dots & (s^k)^\lambda & \dots & (s^n)^\lambda \end{pmatrix}$$

Fig. 1. Example of matrix G.

Step 2 (Generating D matrix): This step consists of generating the ω spaces with which we will work. Each space consists of a tuple $S_i=(D_i.G)$, i=1… ω, where D_i is a random symmetric matrix of size (λ+1)×(λ+1) and G is the matrix generated in the first step. Then, ω matrices $A_i= (D_i.G)^T$, i=1… ω, must be computed. Let $A_i(j)$ represent the j^{th} row of A_i.

Step 3 (Selecting τ spaces): Each node must choose τ spaces from the ω spaces. For any chosen space i the node will carry a row from the matrix A_i, and this row must be secret and shouldn't be revealed to other nodes.

In terms of memory usage, each node needs to store (λ+1)×τ elements. Because the length of each element is the same as the length of a secret key, the memory usage of each node is (λ+ 1)×τ times the length of the key.

C. Key agreement Phase

After deployment, each node must carry its id, the different chosen spaces, and the corresponding rows of matrix A_i. According to Blom's scheme, two nodes can find a common secret key if they have both picked a common key space. Therefore, to communicate with his neighbours, each node must broadcast a message containing his id and the spaces it carries. If a neighbour receiving the message finds that it shares a common space A with the sender, they can compute their shared secret key using Blom's scheme. (See Figure 2)

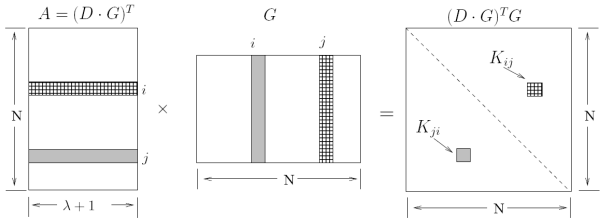

Fig. 2. Example of generating a pairwise key from a common space.

This scheme results in a connected graph, rather than the complete graph provided by Blom's scheme. In the case where the graph is only connected, each sensor node needs to carry less key information. To make it possible to find pairwise keys, all we need is to have a connected graph with high probability. According to [2], we have this equality:

$$p_{actual} = 1 - \frac{((\omega-\tau)!)^2}{(\omega-2\tau)!\,\omega!}. \qquad (1)$$

Where p_{actual} is the actual probability that any two neighbouring nodes share at least one space. It depends on selected values of ω and τ. So, we have interest in choosing a high p_{actual} to increase the probability that our network is connected. Figure 3 show the values of p_{actual} when ω varies from τ to 100 and τ = 2, 4, 6, 8. For example, when we choose τ = 4, ω must not exceed 11 to achieve a high probability of connectivity (>0,9).

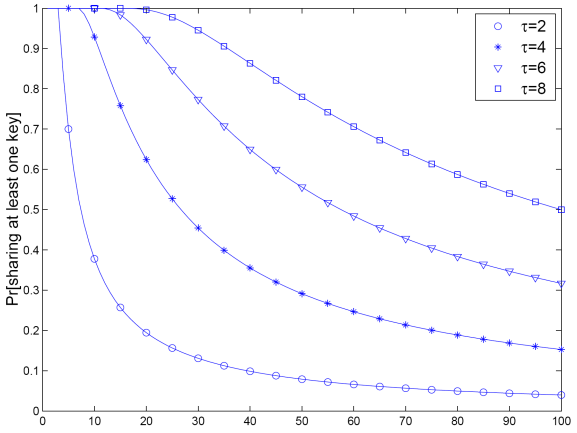

Fig. 3. Probability of sharing at least one space when each of two neighbouring nodes randomly selects τ spaces from ω spaces.

The multiple-space key pre-distribution scheme is evaluated in terms of its resilience against node capture. The evaluation metric is: when x nodes are captured, what is the probability that at least one key space is broken? Each space is considered as the basic λ -secure Blom scheme, so to break a key space, an adversary needs to capture λ +1 nodes that contain this key space's information; otherwise, the key space is still perfectly secure. This analysis shows when the network starts to become insecure.

Fig. 4 shows both simulation and analytical results. For example, ω is set to 50, τ is set to 4, and the value of λ for each space is 49, an adversary needs to capture about 380 nodes in order to be able to break at least one key space.

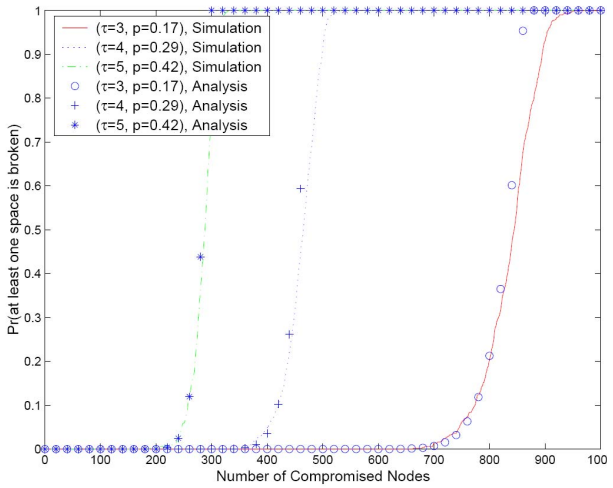

Fig. 4. The probability of at least one key space being compromised by the adversary when the adversary has captured x nodes (λ=49, ω = 50). p in the figure represents p_{actual}.

IV. Implementation and Discussions

This section presents the implementation of a key management solution to be integrated in the TinyOS operating system. TinyOS is written in NesC [11], a C-based language that provides support for the TinyOS component and concurrency model. Each component can correspond to a hardware element (led, timer, ADC, etc.) and can be reused in different applications. An application is composed by a set of components linked together to achieve a fixed goal. The implementation of a component is done by defining a set of commands, events, and tasks. Figure 5 shows the architecture of a TinyOS component. TinySec, de facto security architecture for wireless sensor networks, provides security properties such as integrity and confidentiality. It is composed of a set of components linked together. RC5 and Skipjack are the two cipher components already implemented in TinySec module. These Cryptographic algorithms are based on block cipher and use a single shared key. The main object of this work is to implement an efficient key management scheme which avoids the reliance of TinySec on a single unsecured key.

Figure 6 depicts the components and the interfaces of TinyOS used by our proposed scheme. Main is a major component that is executed first in any TinyOS application. StdControl is the common interface used to initialize and start TinyOS components. In the boot sequence, Main will call Stdcontrol init() and start() functions. In addition, our application uses a linear-feedback shift register (RandomLFSR) component to generate pseudo-random numbers needed by D matrices. As block cipher, we have chosen the Skipjack component to provide both encryption and authentication. The choice of Skipjack as a symmetric key cipher for this system is due to the fact that it has already been implemented and tested for the Mica2 platform in TinySec. Also, it is more efficient than RC5.

Basically, our implementation uses several Timer interfaces (provided by a Timer component) for handling message transmission during key establishment phases. To enable sending and receiving messages our scheme uses the genericComm component that provides interfaces sendMSg and receiveMsg. This code reuse in TinyOS saves code space in ROM as well as data space in RAM because less variables and cipher contexts have to be defined.

However, one of the major constraints on implementing any scheme on a sensor platform is the small available payload size of packets. Under TinyOS specifically, packet payload is limited to 29 bytes. Each of the symmetric keys deployed in a node are 8 bytes, which matches the key size used for the TinySec. In the agreement phase, after finding a common space, any two nodes that would like to communicate must exchange their corresponding rows of matrix A. A row of matrix A contains λ values of 8 bytes. So it is difficult to send all the amount of data using a 29 bytes packet, especially when λ is high. The solution that we have chosen is to generate 4 keys of 2 bytes instead of computing a unique key of 8 bytes. To compute a single key of 2 bytes, the rows of A and G matrices need only to have 8 bit values. If we choose λ<30, sending a single message is sufficient, else, we must send multiple messages. Because a continuous transmission of these packets fails in TOSSIM, we employed timers that fire every 100 ms in our simulations because this provides sufficient spacing for TOSSIM while still allowing for maximum channel utilization. Another constraint of sensor nodes is the small available memory space. In order to cope with the severe hardware constraints of sensor nodes, TinyOS only allows for static memory allocation. This makes it very

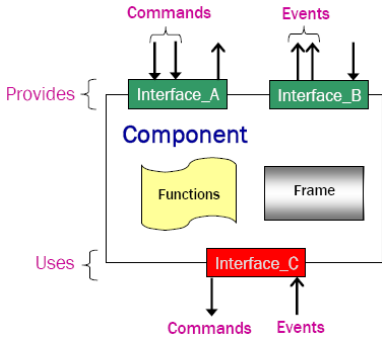

Fig. 5. Architecture of a TinyOS component.

space and time efficient because there is no need for maintaining an additional data structure to manage the dynamic heap. This allows using the entire RAM for storing information. But the downside is that all variables and their sizes have to be known at compile time which makes working with dynamic data structures, such as linked lists or hash maps, almost impossible. Another problem is the amount of available RAM because some applications may need more memory, probably just temporarily, than the node offers. This is where the EEPROM (also called flash) might come in handy. The flash is mostly larger than the RAM but reading from it and writing to it are operations needing a lot of time and energy. These memory constraints lead us to optimize the use of variables, especially global variables that take a lot of memory space. When it is necessary to use them, we make pointers to them to reduce this space.

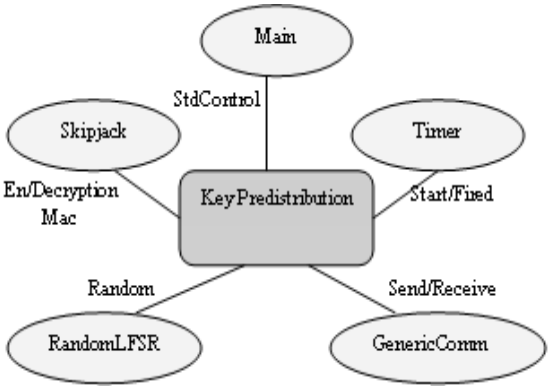

Fig. 6. Components and interfaces used by our proposed solution.

V. EXPERIMENTATION AND PERFORMANCES EVALUATION

In this section, we present experimental results of our proposed solution in terms of memory overhead and key computation time. Experiments were conducted with the TOSSIM simulator [18]. TOSSIM is actually a discrete event emulator designed specifically for TinyOS applications. It allows verification of basic properties of applications before they are loaded into motes for operation in the field. We also evaluate the average node energy consumption overhead needed to compute a pairwise key. To achieve this goal, the PowerTOSSIM simulator [19] is used. It is based on TinyOS and TOSSIM. PowerTOSSIM makes use of the TinyOS and TOSSIM component model to instrument hardware state transitions for the purpose of tracking power consumption. Simulated hardware components (radio, sensors, LEDs, etc.) make calls to the PowerState module, which emits power state transition messages for each component. These messages can be combined with a power model to generate detailed power consumption data or visualizations.

A. Parameter setting

In our simulations, we fix the total number of nodes to 100. The security parameter λ=20. The values of ω and τ are chosen in a manner that the network connectivity is very high, with $p_{actual} > 0{,}9$. To improve the accuracy of our results, we repeated the simulations at least 50 times.

B. Time to compute a pairwise key

To measure the average time for computing a pairwise key, we use SysTime, a TinyOS component that provides a 32-bit system time based on the available hardware clock. The average time to compute a pairwise key was found equal to 6 seconds. This time is very low compared to the time spent by ECDLP protocol (see Table 1).

C. Memory overhead

In terms of memory space, the implementation of this scheme on Mica2 motes occupied approximately 14 KB of ROM, representing 10% of the available ROM and 521 bytes of RAM which represent only 13% of the RAM.

D. Energy Overhead

The commercially available platforms such as the Mica2 are limited to 128 KB of program memory and 4 KB of RAM [20]. They are alimented with double AA batteries that offer energy of 2850 mAh alimented by a 3v power. According to Equation (2), a sensor node provides a total energy of 30780 joule.

$$\text{Watt} = \text{Joules} / \text{sec} = \text{Volt} * \text{Ampere} \quad (2)$$

The energy overhead introduced by our application is 417 mj which is negligible regarding the total energy of a mote (< 1%). Therefore, the proposed key management solution is very efficient and suits well the severe constraints of sensor nodes. Such efficiency is necessary for any security solution in wireless sensor devices.

Table 1 shows some comparative results with two well known key distributions protocols implemented specially for sensor networks, namely Diffie-Hellman based on the Elliptic Curve Discrete Logarithm Problem (ECDLP) [3] and the Localized Encryption and Authentication Protocol (LEAP) [7]. ECDLP is a public key algorithm used in asymmetric cryptography. Indeed, elliptic curves are believed to offer security computationally equivalent to that of Diffie-Hellman based on DLP with remarkably smaller key sizes. LEAP is a key establishment protocol designed for symmetric ciphers. LEAP supposes that no single keying mechanism is appropriate for all the secure communications that are needed in sensor networks. As such, LEAP supports the establishment of four types of keys for each sensor node – an individual key shared with the base station, a pairwise key shared with another sensor node, a cluster key shared with multiple neighbouring nodes, and a group key that is shared by all the nodes in the network. The experimental values shown for the LEAP protocol in Table 1 consider only the necessary memory for establishing pairwise and cluster key between 2 nodes. Since each scheme has different parameters for measurements, this is a broad comparison.

TABLE I
A COMPARISON SUMMARY OF KEY MANAGEMENT SCHEMES IMPLEMENTED FOR TINYOS.

	ECDLP	LEAP	Proposed scheme
ROM (KB)	34.1	17.9	14
RAM (Bytes)	1000	600	521
Time (second)	34	-	6
Energy (joules)	0.9	-	0,417

Table 1 shows clearly that the proposed solution outperforms ECDLP and LEAP with respect to all criteria. Our proposed scheme is very memory efficient regarding the fact that memory space in sensor nodes is very limited and must be used with care. In terms of key generation time and energy consumption overhead, Table 1 compares only between ECDLP protocol and our proposed scheme. The LEAP protocol has not been evaluated for these metrics. We can conclude that our proposed scheme is also time efficient: the time (6 seconds) spent by our scheme to generate a pairwise key is lower than the time spent by ECDLP (34 seconds) to generate a pair (public/private key). Energy is the scarcest resource of all and each milliamp consumed is one milliamp closer to death. As a result, every protocol designed for sensor nodes must be evaluated in terms of power consumption to be validated. The evaluation that we have done demonstrates that our scheme consumes 417 mj which represent nearly half the energy needed by the ECDLP protocol.

VI. CONCLUSION

In WSN, the level of security versus the consumption of energy, computation, and memory resources constitute a major design trade-off. This paper presents an efficient solution for the key management problem under the TinyOS distribution. Performance comparisons with two well known key establishment protocols show that our proposed solution exhibits better results in terms of memory, time of computation, and energy overhead. Although the proposed solution solves the problem of key establishment facing these networks, there are still other points to be solved such as the support of node addition and revocation. Also, an open problem is to design an energy aware intrusion detection mechanism to detect compromised nodes in the network.

REFERENCES

[1] I.F.Akyildiz, W. Su, Y. Sankarasubramaniam, and E. Cayirci, "A survey on sensor networks", IEEE Communication Magazine, vol. 40, no. 8, Aug. 2002, pp. 102–116.

[2] W. Du, J. Deng, Y. S. Han, and P. K. Varshney, "A pairwise key pre-distribution scheme for wireless sensor networks", in Proceedings of the 10th ACM Conference on Computer and Communications Security (CCS), Washington DC, USA, October 27-31 2003, pp. 42–51.

[3] D.J Malan., M. Welsh and M. D. Smith, "A public-key infrastructure for key distribution in tinyos based on elliptic curve cryptography", in 2nd IEEE International Conference on Sensor and Ad Hoc Communications and Networks (SECON 2004), 2004, pp. 71–80.

[4] R. J. Watro, D. Kong, S. f. Cuti, C. Gardiner, C. Lynn, and P. Kruus, "Tinypk: securing sensor networks with public key technology", In 2nd ACM Workshop on Security of ad hoc and Sensor Networks (SASN'04), Washington, DC, October 2004, pp. 59–64.

[5] A. Perrig, R. Szewczyk, V. Wen, D. Culler, and J. D. Tygar, "SPINS: Security protocols for sensor networks", In Seventh Annual ACM International Conference on Mobile Computing and Networks (MobiCom 2001), July 2001.

[6] J. Kohl and B. Neuman, "The Kerberos Network Authentication" Service (V5). RFC 1510, Sep. 1993

[7] S. Zhu, S. Setia, and S. Jajodia, "LEAP: Efficient security mechanisms for large-scale distributed sensor networks", ACM Conference on Computer and Communications Security (CCS '03), October, 2003, pp. 62–72.

[8] C. Karlof, N. Sastry, and D. Wagner, "Tinysec: a link layer security architecture for wireless sensor networks", In SenSys '04: Proceedings of the 2nd international conference on Embedded networked sensor systems, ACM Press, 2004, pp. 162-175.

[9] R. Blom, "An optimal class of symmetric key generation systems," Advances in Cryptology: Proceedings of EUROCRYPT 84, Lecture Notes in Computer Science, Springer-Verlag, 1985, 209:335–338.

[10] C. Blundo, A. D. Santis, A. Herzberg, S. Kutten, U. Vaccaro, and M. Yung, "Perfectly-secure key distribution for dynamic conferences," Lecture Notes in Computer Science, 1993, 740:471–486,.

[11] D. Gay, P. Levis, R. V. Behren, M. Welsh, E. Brewer, and D. Culler, "The nesC Language: A Holistic Approach to Networked Embedded Systems", In Proceedings of Programming Language Design and Implementation (PLDI) 2003, June 2003.

[12] L. Eschenauer and V. D. Gligor, "A key-management scheme for distributed sensor networks", In Proceedings of the 9th ACM conference on Computer and communications security, November 2002.

[13] H. Chan., A. Perrig, and D. Song, "Random key predistribution schemes for sensor networks" In IEEE Symposium on Research in Security and Privacy, 2003.

[14] W. Du, J. Deng, Y. Han, S. Chen, and P. Varshney, "A key management scheme for wireless sensor networks using deployment knowledge", In IEEE Infocom'04, 2004.

[15] D. Liu and P. Ning, "Location-Based Pairwise Key Establishments for Relatively Static Sensor Networks," in 2003 ACM Workshop on Security of Ad Hoc and Sensor Networks (SASN'03), October 31, 2003.

[16] S. Hussain, F. Kausar, A. Masood, "An efficient key distribution scheme for heterogeneous sensor networks", In proceeding of the International Conference On Communications And Mobile Computing, 2007.

[17] P. Traynor, R. Kumar, H. Bin Saad, G. Cao and T. La Porta, "LIGER: Implementing Efficient Hybrid Security Mechanisms for Heterogeneous Sensor Networks", in proceeding of ACM MobiSys'06, June 19–22, 2006.

[18] P. Levis, N. Lee, M. Welsh, and D. Culler, "TOSSIM: Accurate and scalable simulation of entire TinyOS applications", in Proceedings of the First ACM Conference on Embedded Networked Sensor Systems (SenSys) 2003, Nov. 2003.

[19] V. Shnayder, M. Hempstead, B. Chen, G. Werner-Allen, and M. Welsh. "Simulating the Power Consumption of Large-Scale Sensor Network Applications", In Proceedings of the Second ACM Conference on Embedded Networked Sensor Systems (SenSys'04), 2004.

[20] http://webs.cs.berkeley.edu/tos.

[21] Crossbow technology inc. URL: http://www..xbow.com