SECURITY INFRASTRUCTURE FOR WIRELESS SENSOR NETWORKS

Laura GHEORGHE[1], Răzvan RUGHINIŞ[2], Nicolae ŢĂPUŞ[3]

Applicaţiile critice cum ar fi monitorizarea militară şi medicală folosesc reţelele de senzori wireless cu scopul detecţiei evenimentelor specifice. De aceea, este important ca aceste reţele să fie protejate împotriva atacurilor. În acest articol, propunem o infrastructură de securitate pentru reţele de senzori wireless, care oferă autentificare, integritate, prevenţia intruziunilor, protecţie împotriva atacurilor de tip replay şi asigurarea fiabilităţii comunicaţiei. Protocolul a fost implementat în kernelul sistemului de operare TinyOS şi testat folosind simulatorul TOSSIM în cadrul mai multor scenarii de atac. Protocolul s-a dovedit capabil de a bloca încercările de injectare de pachete suspecte sau de replay a packetelor.

Critical applications such as military and medical monitoring use Wireless Sensor Networks with the purpose of detecting specific events. Therefore, it is important to protect the network against malicious attacks. In this paper, we propose a Security Infrastructure for Wireless Sensor Networks, which provides authentication, integrity, intrusion prevention, anti-replay protection and reliability. The protocol is implemented in TinyOS and tested with TOSSIM in several attack scenarios. It proves to be able to reject malicious attempts to inject and replay plackets.

Keywords: Wireless Sensor Networks, security, authentication, anti-replay, integrity, reliability, attack, intrusion detection

1. Introduction

Wireless Sensor Networks (WSNs) are composed of small devices, called sensor nodes that have several distinctive characteristics such as low energy consumption, low processing power, limited memory and small radio range. These nodes have the ability to organize themselves into a network and perform sensing and communication operations in order to monitor a certain environment [1].

[1]Assist., Faculty of Automatic Control and Computers, University POLITEHNICA of Bucharest, Romania, e-mail: laura.gheorghe@cs.pub.ro
[2] Conf., Faculty of Automatic Control and Computers, University POLITEHNICA of Bucharest, Romania, e-mail: razvan.rughinis@cs.pub.ro
[3] Prof., Faculty of Automatic Control and Computers, University POLITEHNICA of Bucharest, Romania, e-mail: ntapus@cs.pub.ro

There are at least five features that should be considered when designing WSN solutions: scalability, security, reliability, self-healing and robustness [2]. Depending on the application, these requirements are more or less critical.

In military or medical applications, security is a critical requirement because attackers can intercept and inject malicious packets in the network and this could compromise the whole network [3]. Attackers can steal valuable information or manipulate the network for malicious purposes [4].

A certain amount of resources is required when implementing a security solution, including memory space, processing power and energy [5]. Therefore, traditional security methods cannot be implemented in sensor networks. The solutions designed for Wireless Sensor Networks should take in consideration their specific constrains.

The security requirements of WSNs must also be considered when designing a new security protocol. These requirements are: confidentiality, integrity, authenticity, freshness, reliability, availability and energy-efficiency [6]. In this paper, we present a security infrastructure that meets the following key requirements: authenticity, integrity, freshness, reliability and energy-efficiency.

2. Related Works

Among the most relevant solutions for security issues in WSNs, one can count TinySec (2004), LEAP (2003), and SPINS (2002).

TinySec, designed by Karlof et al. [7] is included in the TinyOS as a link-layer security architecture, addressing essential requirements such as authentication and integrity, semantic security (employing an Initialization Vector) and confidentiality. Anti-replay protection was not included as it was deemed better to address it at higher levels of the communication protocol stack.

LEAP (Localized Encryption and Authentication Protocol) represents a key management protocol for WSNs, designed by Zhu et al.[8]. It has been subsequently implemented in TinyOS as LEAP+ and then used on Berkley Mica2 motes [9]. LEAP relies on four key categories that differentiate among message types in WSNs. There is a Individual Key, which each node shares with the base station. The base station uses Group Keys to communicate securely, by encryption, with nodes. Nodes and their neighbors employ Cluster Keys, while pairs of immediate neighbors use Pairwise Keys.

SPINS, designed by Perrig et al. [10] consists of two components: SNEP and μTESLA, implemented to run on TinyOS. SNEP, which has been subsequently replaced by TinySec, provided authentication, integrity, freshness and confidentiality. μTESLA makes authenticated broadcasts possible by a Message Authentication Code (MAC), and it also provides confidentiality by

encryption – emulating asymmetry by a delayed discosure of symmetric keys, and freshness through nonce.

3. Authentication and Anti-replay Security Protocol Design

The Authentication and Anti-replay Security Protocol (AASP) aims at providing authenticity, anti-replay, integrity and intrusion prevention for WSNs. In order to meet these requirements, two methods are designed: an anti-replay method, and an authentication connection [11].

3.1. Anti-replay Method

We designed an anti-replay method that uses Message Authentication Code (MAC) and assumes that the attacker is an outsider that does not know the secret key.

The protocol mechanism consists in including in the current message the MAC computed using the last packet sent between the same source and destination node. This mechanism bounds the packet to its context. The MAC is recomputed and checked at the destination node. If it is correct, the packet is accepted, if not, the packet is dropped.

The MAC is computed from a payload (M), a shared key (K), and a collision-resistant hash function (H), using the HMAC algorithm [12].

In the case of a re-play attack the MAC computed at the destination node would not match the MAC found in the packet.

However, the first packet sent between the same source and destination will be always accepted at the destination if we use only this anti-replay mechanism. For this reason, we introduce the authentication connection that should be created before any data packet is accepted

3.2. Authentication Connection

An authentication connection has to be established before transferring any data packet between a particular source and destination node. In order to obtain an authentication connection, both nodes have to authenticate to each other using an authentication handshake that consists in four steps. The proposed protocol is connection oriented because of the establishment of this authentication connection.

Both handshake and data packets have to be created and verified using the anti-replay method described in the previous subsection, for providing strong authentication and anti-replay protection.

The authentication handshake is initiated by the node that wants to send a message to another node. The message consists in a standard “Authentication

request" message that is sent to the destination node. The destination recognizes the request and generates a random value called Challenge and sends it to the source node. The source node computes the MAC of the Challenge value and sends it to the destination node. The packet also contains the MAC of the Authentication request value as it is build using the anti-replay method. The destination node recomputes the MACs and valides them. If they are correct, the node has to compute the MAC using both the Authentication request and the Challenge value in order to authenticate itself. The MACs are validated by the source node and if they are correct, the authentication connection is established. In this moment, both the source and destination node have authenticated each other and data packets can be exchanged. The Auth field will be 0 until the connection is created and 1 afterwards.

4. Protocol Implementation

TinyOS is an open-source, event-driven and component-base operating system for sensor networks [13]. We implemented our security protocol in the kernel of TinyOS within two layers using three implementation components are three wiring components.

The first layer, called the MAC layer, is placed between the ActiveMessage layer and the AMSender and AMReceiver components. The second layer, called the Authentication layer, is placed between the operating system kernel and the application layer.

The MAC layer consists in two implementation components and two wiring components. The first component is placed under the AMSender component and receives all traffic that is sent by that particular node. This component is used to compute and store the MAC for each packet.

The second component is placed under the AMReceiver and is able to analyze and alter all traffic destined to the current node. It has the role to compute and verify the MAC for each packet. Every time it receives a valid packet, it computes and stores the MAC of that packet. Then, it compares the stored value with the one found in the next received packet. If the value is incorrect, the packet is not sent further to the AMReceiver component, so it is blocked or sent with a modified Auth field towards the Authentication layer where it will be dropped. So, we can either drop packets at the MAC or at the Authentication layer.

The Authentication layer consists in an implementation component and a wiring component. It is used by the application layer for secure data delivery. The component initiates and performs the authentication handshake, not permitting any data traffic from the application layer until the authentication connection has been established. The layer is able to store and validate all handshake messages and to block data packets until the connection is established.

The state of each connection is stored in the Authentication layer of the source and destination node. The connection initiator knows that it has initiated the connection with a particular node and will be able to reject unrequested Challenge messages. Both of them store the challenge value used in the handshake, so they will be able to check the validity of the third and forth handshake messages. The replayed handshake messages will be detected and rejected by the Authentication layer.

The packet has a simple security header consisting in the P_MAC and the Auth fields. The message field represents the packet payload. The network and link layers will add headers of their own, but they are out of the scope for this paper.

We describe the functionality implemented in TinyOS. We represent packets in the format [P_MAC, Auth, Message]. We consider that node X wants to communicate with node Y and initiates the conversation.

Node X Application layer generates a data packet. When receiving the data packet, the Authentication layer stores it and generates an Authentication Request (AR) packet [0, 0, AR] destined to node Y.

When node Y receives the AR, if it has no authentication connection with node X, it generates a random Challenge (C) and sends it to the source node [0, 0, C]. Node Y stores C for further computations.

Node X receives the Challenge and if it does not have an authentication connection with node Y, it generates the third handshake message by computing the two MACs. The packet [MAC(AR, K), 0, MAC(C, K)] is sent to node Y. The Challenge is stored by node X for the verification of the forth handshake packet.

When node Y receives the packet, it computes the MACs and verifies them. If they are valid, the node generates the forth handshake message [MAC(C, K), 0, MAC(AR, C, K)]. Node X receives the packet and verifies it. If the forth packet is valid, the authentication connection is established. In the next packets, the auth field will be equal to 1.

When node X and Y have an authentication connection and one of them receives a handshake packet from the other, it rejects it with the alert: “Node already authenticated”. If they do not have an authentication connection and one of them sends a data packet, the packet is rejected with the alert: “Node not authenticated”.

Whenever the P_MAC field is found incorrect, the packet is dropped with the alert: “Incorect MAC”. The detection is made at the MAC layer.

5. Experimental Results

TOSSIM is tool that is able to simulate TinyOS applications in a precise manner [14]. We are able to run some attack simulations using this particular

simulator. In this paper we present some of the attack scenarios that were analyzed and simulated.

We assume that the attacker is an outsider, an external device that does not have the cryptographic keys of the network. However, the protocol format cannot be obfuscated from the attacker, so we assume that he is able to interpret and build coherent protocol messages.

In the first scenario, the attacker sends an authentication request to a particular node and receives a Challenge. The attacker is not able to compute a valid MAC, so the third handshake message should be rejected at the destination with the alert: "Not a valid handshake packet", as presented in Fig. 1. The packet is recognized as invalid at both MAC and Authentication layers because both P_MAC and the payload contain incorrect MACs.

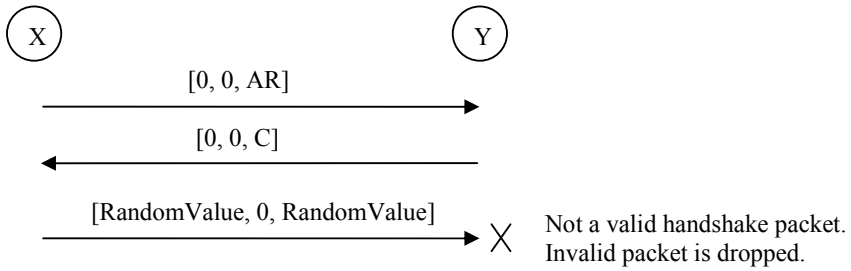

Fig. 1. Not a valid handshake packet

In the second scenario, a malicious node poses as a node that already has an authentication connection with the target node (source address spoofing) and tries to send an invalid data packet. The P_MAC field does not contain the MAC of the last message because the attacker cannot compute a correct value, so the packet is dropped at the destination with the alert: "Incorrect MAC", as observed in Fig. 2.

Fig. 2. Not a valid handshake packet

In the third scenario, the attacker tries to replay a packet from a valid conversation. The P_MAC field is not matching the MAC of the previous packet in the target conversation, so the packet is dropped with the alert: "Incorrect MAC", similar to Fig. 2.

In the forth scenario, the attacker tries to replay a handshake message. The Authentication layer is able to recognize this packet and can generate a more

specific alert: “Replayed handshake message”. However, both layers are able to recognize this packet as invalid.

In the fifth scenario, the attacker poses as an authenticated node and tries to send an “Authentication request” message to its connection partner. The target rejects the message with the alert: “Node already authenticated”. If we would permit this kind of re-authentication, it would easily be used by attackers to tear down connections. This problem should be solved using connection timeout. If an authentication connection is desynchronized or one of the nodes looses connection data, re-authentication would be possible after the connection times out.

The AASP protocol has proved itself resistant to many types of injection and replay attacks. However, the de-synchronization problem appears when packets are lost. We describe the problem and its solution in the subsequent sections.

6. Reliability Mechanisms

The main problem regarding the basic functionality of AASP is related to packet loss. The wireless medium is not a perfect one; therefore it is not possible to avoid packet loss. When packets are lost, the anti-replay mechanism is de-synchronized.

In Fig. 3, we present the effect of packet loss upon the authentication connection. Message i contains the MAC of message i-1, and node Y compares this MAC with the one computed from the previous valid packet. If they are equal, the packet is considered correct and the MAC of message i is computed and stored. Message i+1 contains the MAC of message i, but it is lost on its way to destination. Message i+2 contains the MAC of message i+1 and node Y compares the stored MAC of message i with the MAC found in the packet, of message i+1. The values are not equal, so the packet is considered invalid and dropped. All subsequent messages will be dropped by node Y and the connection is considered de-synchronized.

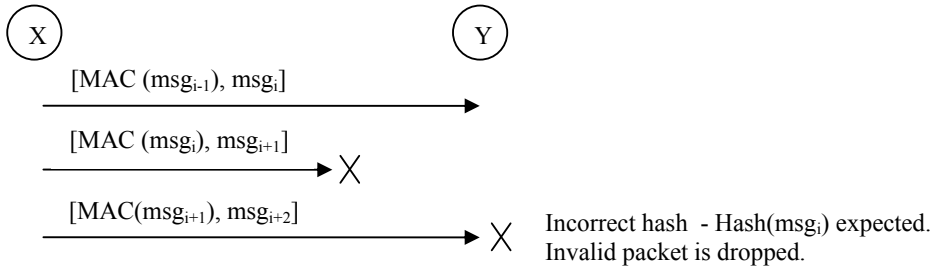

Fig. 3. Anti-replay mechanism de-synchronization

SPINS deals with the same problem. They use, for the anti-replay mechanism, a counter that is incremented at each packet. The anti-replay mechanism is de-synchronized when packets are lost, because the destination expects one counter and the received packet is based on another counter.

In order to recover from de-synchronization, the lost packets have to be re-sent by the source node. Therefore, a retransmission mechanism has to be implemented. The packet retransmission can be accomplished by either positive or negative acknowledgments. The positive acknowledgements are more reliable but they consume more energy than the negative ones.

In the case of positive acknowledgements, the destination node has to send an acknowledgement for each valid packet that it has received, as in Fig. 4.

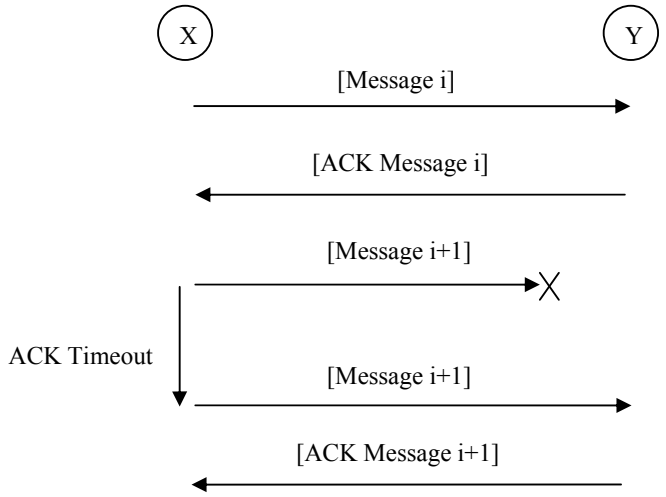

Fig. 4. Positive ACK

The source node waits a pre-defined period of time to receive the acknowledgement and when it times out, it resends the packet. Only when it receives the acknowledgement for the current packet, it will send the next packet. The acknowledgement has to contain the sequence number of the acknowledged packet.

In the case when acknowledgements are lost, the packet is resent anyway and it will be considered re-played message. Therefore, losing data packets is less harmful then losing data packets.

In the case of negative acknowledgements, the destination node detects lost packets only when it receives out-of-order packets, which are packets with a sequence number greater than the expected one. When the destination detects packet loss, it sends a negative acknowledgement to the source node.The source node will re-send all packets with the sequence number equal and greater than the expected one, and less then the received one. The destination node will store the

out-of-order packet until the lost ones are recovered and they are all checked by the anti-replay mechanism. The mechanism is described in Fig. 5.

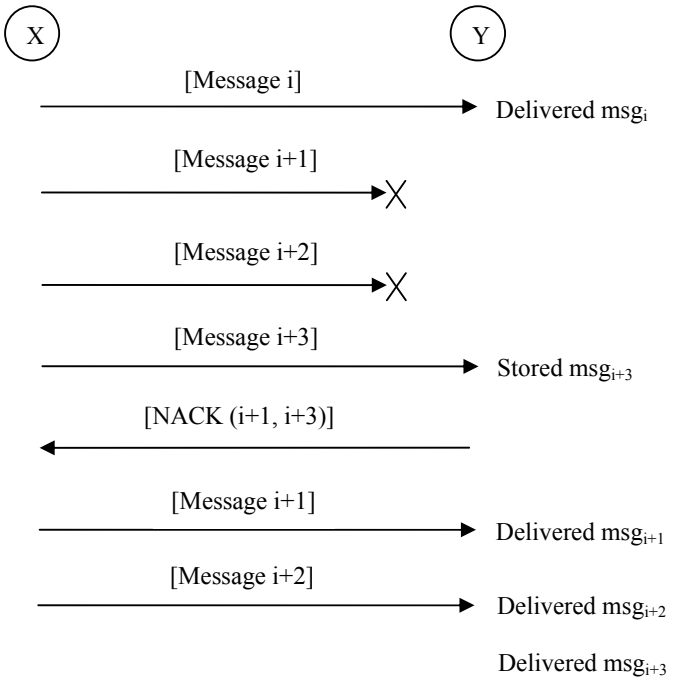

Fig. 5. Negative ACK

Different sensor network applications require different reliability level. Critical application require all messages to be received correct and without delay, while habitat monitoring can tolerate packet loss and delay.

When comparing the two packet recovery methods in the context of sensor network applications, both of them present advantages and disadvantages. We compare them in terms of energy consumption, reliability and delay in the case of packet loss, as presented in Table 1.

Table 1

Comparison between positive and negative acknowledgements

ACK type	Positive	Negative
Consumed energy	Greater	Less
Reliability	Greater	Less
Delay – low packet loss	Greater	Less
Delay- high packet loss	Less	Greater

In terms of energy consumption, the negative ACKs are more efficient than the positive ones, because they are sent only when an out of order packet is received by the destination.

However, the positive ACKs provide a greater level of reliability because they are tracking the state of delivery for each packet and are able to detect packet loss faster than negative ACKs. The negative ACKs method permits the loss of several packets before detection and recovery. Positive ACKs method performs recovery from the first lost packet.

The delay depends on the amount of lost packets. In the case of low packet loss, the positive ACKs introduce greater delay then negative ones because the source has to wait for every packet to be acknowledged.

In the case of high packet loss, the negative ACKs introduce greater delay than positive ones, because a large amount of packets can be lost before the destination identifies the problem and sends a negative ACK.

8. Implementing Acknowledgements

For implementing this optimization we introduce a new field in the protocol header, called ACK/NACK flag that is represented on one bit. The packet is acknowledgement when the ACK flag is set.

In the case of the handshake packets, acknowledgments can be piggybacked in the actual messages. However, be cannot use the same procedure for data packets because in sensor networks the traffic is mostly unidirectional, from the sensor nodes to the base station. Therefore, we need separate acknowledgement packets for data messages.

The positive ACK packets contain the sequence number of the acknowledged packet in order to specify which packet has reached destination. The negative ACK packets contain two sequence numbers: the sequence number of the expected packet, which is the first lost packet, and the sequence number of the out-of-order packet that was received at the destination. Based on this information, the source node has to re-send all lost packets.

When using positive ACKs, the source node has to store only the last packet sent until the ACK is received. In the case of negative ACKs, the node has to store a pre-determined number of packets that can be requested by the destination node.

For the positive ACKs we need a single timer to implement the ACK timeout and resend the packet. If the ACK is received before timeout, the next packet is sent and the timer is reset. If the ACK is not received before timeout, the packet is re-sent and the timer is reset.

For the negative ACKs we need a timer at the destination. This timer is set when the negative ACK is sent to the source node. If the lost packets are not received until timeout, the negative ACK is resent. If they are received, the timer is stopped.

We implemented this optimization at the Authentication layer in two different variants: one for positive and one for negative ACKs. The implementation uses a TinyOS timer and structures that store packets that can be recovered.

9. Testing Acknowledgements

For testing ACKs we took a section of a bigger topology, more exactly: node 1 and 3. Node 3 wants to communicate with node 1, which is the base station, and sends an Authentication Request. It receives from node 1 a Challenge and sends the proper response. Node 1 also authenticates itself by sending the correct handshake response. In Fig. 6 we can observe the handshake packets and the piggybacked ACKs. The connection is successfully established.

```
(3): AuthLayer: Packet sent [payload=124 MAC=0 auth=0 ack=0 (3->1)]
(1): AuthLayer: Packet received [payload=124 MAC=0 auth=0 ack=0 (3->1)]
(1): AuthLayer: Packet sent [payload=198 MAC=0 auth=0 (1->3)]
(3): AuthLayer: Packet received [payload=198 MAC=0 auth=0 ack=1 (1->3)]
(3): AuthLayer: Packet sent [payload=64671 MAC=65040 auth=0 (3->1)]
(1): AuthLayer: Packet received [payload=64671 MAC=65040 auth=0 ack=1 (3->1)]
(1): AuthLayer: Packet sent [payload=47653 MAC=64671 auth=0 (1->3)]
(1): AuthLayer: Managed to authenticate myself to node 3
(3): AuthLayer: Packet received [payload=47653MAC=64671 auth=0 ack=1 (1->3)]
(3): AuthLayer: Managed to authenticate myself to node 1
```

Fig.6. Authentication handshake

After the authentication connection has been established, data can be exchanged between the two nodes. Node 3 starts sending sensed data towards node 1. As we can observe in Fig. 7, the auth field is 1 and the ack field is 0 for the data packets.

```
(3): AuthLayer: Packet sent [seq=1 auth=1 ack=0 (3->1)]
(1): AuthLayer: Packet received [seq=1 auth=1 ack=0 (3->1)]
(1): AuthLayer: Packet sent [payload=1 auth=1 ack=1 (1->3)]
(3): AuthLayer: Packet received [payload=1 auth=1 ack=1 (1->3)]
```

Fig.7. Normal flow of packets

Node 1 sends an ACK for each valid data packet. The ACK packets have the auth and ack fields set to 1 and the payload contains the sequence number of the acknowledged packet. For the data messages, the payload is not displayed in the output.

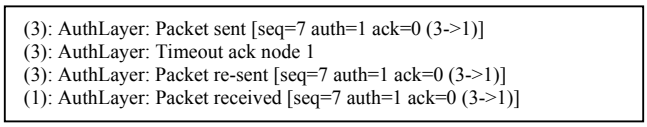

Fig.8. Packet recovery

In the case of packet loss, the timer is fired after the pre-defined period of time and the packet is resent, as presented in Fig. 8.

10. Discussion

The acknowledgement timeout period should be determined experimentally when deploying the application, because it depends on the network dimension, density, environment conditions, and other factors.

The timeout period should be large enough to permit the data packets and the acknowledgements to travel the longest path from a sensor node to the base station. Therefore, the timeout should be greater than twice the period it takes a packet to travel the longest path.

The application reliability, energy consumption and delay requirements should be taken into consideration when choosing between the two packet recovery methods and when adjusting the timeout period. A tradeoff between energy consumption and reliability should be considered depending on the application requirements.

A similar attack to SYN Flood attack can be reproduced using ARs. The malicious node sends AR packets with spoofed source addresses to the same target node in order to determine the generation of a large number of Challenge messages and to create half-opened connections. This attack is very dangerous because it prevents the destination sensor to sleep. We developed a mechanism that detects flood attacks called the Storm Control Mechanism [15].

Our security protocol is able to fight against external attackers that do not possess the secret keys. However, it is not able to protect against compromised nodes. The solution is to use a trust and reputation model and we developed a lightweight solution for sensor networks [16].

11. Conclusion

This paper presents an innovative Security Infrastructure for Wireless Sensor Networks whose main features are authentication, anti-replay, integrity, intrusion detection, and reliability.

Strong authentication is provided by the use of a MAC and by establishing an authentication connection between the communicating nodes. The MAC is computed using a shared key between source and destination nodes; therefore,

external attackers cannot compute a valid MAC because they do not possess that key. An end-to-end authentication connection between two nodes is required in order to exchange data packets between two nodes. The connection is built by a four-step handshake which requires both nodes to authenticate to each other.

Anti-replay protection is assured by a mechanism in which the MAC is computed on the basis of packet context information, specifically the last packet sent between the source and destination nodes and the sequence number. Integrity protection is granted by computing the MAC as a function of the contents of the current packet.

Reliability is provided by using positive or negative acknowledgements according to application requirements, taking into account that positive acknowledgements provide greater reliability with higher energy consumption and delay than negative ones.

The Security Infrastructure has been implemented in the kernel of TinyOS and several attack scenarios have been simulated using TOSSIM. Evaluation tests indicate that the protocol is resistant to injection and replay attacks. Further research is required to estimate the performance of this solution regarding resource consumption, on variables such as energy, bandwidth, memory, and processor power.

R E F E R E N C ES

[1] *J. Zheng, A. Jamalipour*, "Wireless Sensor Networks: A Networking Perspective", Wiley-IEEE Press, 2009

[2] *D. Westhoff, J. Girao, A. Sarma*, "Security Solutions for Wireless Sensor Networks", NEC Technical Journal, **vol. 1**, 2006, pp. 2-6

[3] *T. Kavitha and D. Sridharan*, "Security Vulnerabilities In Wireless Sensor Networks: A Survey", Journal of Information Assurance and Security, **vol. 5**, 2010, pp. 31-44

[4] *T. Zia, A. Zomaya*, "Security issues in wireless sensor networks", International Journal of Communications, **vol. 2**, 2006, pp. 106-115

[5] *J.P. Walters, Z. Liang, W. Shi, V. Chaudhary*, "Wireless Sensor Network Security: A Survey", Security in Distributed, Grid, Mobile, and Pervasive Computing, CRC Press, 2007

[6] *Y. Qian, K. Lu, D. Tipper*, "Towards Survivable and Secure Wireless Sensor Networks", 2007 IEEE International Performance, Computing, and Communications Conference, Apr. 2007, pp. 442-448

[7] *C. Karlof, N. Sastry, D. Wagner*, "TinySec: a link layer security architecture for wireless sensor networks", Proceedings of the 2nd international conference on Embedded networked sensor systems, ACM, 2004, pp. 162-175

[8] *S. Zhu, S. Setia, S. Jajodia*, "LEAP: efficient security mechanisms for large-scale distributed sensor networks", Proceedings of the 10th ACM conference on Computer and communication security - CCS '03, New York, New York, USA: ACM Press, 2003, pp. 62-72

[9] *S. Zhu, S. Setia, S. Jajodia*, "LEAP+: Efficient security mechanisms for large-scale distributed sensor networks", ACM Transactions on Sensor Networks, **vol. 2**, Nov. 2006, pp. 500-528

[10] *A. Perrig, R. Szewczyk, J. Tygar, V. Wen, D.E. Culler*, "SPINS: Security protocols for sensor networks", Wireless networks, **vol. 8**, 2002, pp. 521-534

[11] *L. Gheorghe, R. Rughinis, R. Deaconescu, N. Tăpus*, "Authentication and Anti-replay Security Protocol for Wireless Sensor Networks", The Fifth International Conference on Systems and Networks Communications, ICSNC 2010, 2010, pp. 7-13

[12] *B. Arazi*, "Message authentication in computationally constrained environments", IEEE Transactions on Mobile Computing, **vol. 8**, 2009, p. 968–974

[13] *P Levis, S. Madden, J. Polastre, R. Szewczyk, K. Whitehouse, A. Woo, D. Gay, J. Hill, M. Welsh, E. Brewer, D. Culler*, "TinyOS: An Operating System for Sensor Networks", Ambient Intelligence, 2005, pp. 115-148

[14] *P. Levis, N. Lee, M. Welsh, D. Culler*, "TOSSIM: Accurate and scalable simulation of entire TinyOS applications", Proceedings of the 1st international conference on Embedded networked sensor systems, ACM, 2003, pp. 126-137

[15] *L. Gheorghe, R. Rughinis, R. Deaconescu, N. Tăpus*, "Reliable Authentication and Anti-replay Security Protocol for Wireless Sensor Networks", The Second International Conferences on Advanced Service Computing, SERVICE COMPUTATION 2010, 2010, pp. 208-214

[16] *R. Rughinis, L. Gheorghe*, "Storm Control Mechanism in Wireless Sensor Networks", 9th RoEduNet IEEE International Conference, IEEE, 2010, pp. 430-435.