A Trustful Infrastructure for Wireless Sensor Networks

Rodrigo Vieira Steiner, Leonardo Maccari Rufino, and Antônio Augusto Fröhlich

Software/Hardware Integration Lab
Federal University of Santa Catarina
PO Box 476, 88040-900 - Florianópolis, SC, Brazil
{rodrigo,leonardo,guto}@lisha.ufsc.br

***Abstract*—This paper describes the design, implementation and evaluation of a trustful infrastructure for Wireless Sensor Networks (WSNs). This infrastructure is composed by an embedded platform, EPOSMote II, and a communication protocol stack (TCP/IP/C-MAC). We focus on security and reliability while still compromised with the low utilization of resources available in sensor nodes. To achieve confidentiality, integrity, and authentication, we make use of data encryption and authentication. Differently from other approaches our authentication scheme possesses a temporal property which protects the network against replay attacks. Our proposal is evaluated in terms of processing time, memory footprint, and energy consumption presenting very little overhead.**

***Keywords*-security; WSN; authentication;**

I. Introduction

Wireless communication channels are easily eavesdropped. This is a problem for many WSN applications in which security is a critical issue. Anyone in the network range can listen and interfere with the communication traffic, making the network vulnerable to all kinds of attacks, such as impersonation, tampering, and replay attacks.

The use of cryptography solves the problem of confidentiality. Nevertheless, through traffic analysis, someone monitoring the network for a determined period of time can associate certain messages with certain events [1]. Therefore, even if the observer cannot decrypt the message, he can correlate it with an event, and through replay attacks cause a network misbehaviour. Furthermore, even if the message is signed it can still suffer from the same problem. Since the message content is not being altered there is nothing to invalidate it. A solution to this problem is to mask the channel – send dummy traffic when no actual messages are being sent, keeping bandwidth usage constant. Thus avoiding traffic analysis. Nonetheless, this is impractical in WSNs as it would increase packet traffic, consequently increasing energy consumption and substantially reducing the network lifetime.

In this article, we describe the implementation and evaluation of a trustful infrastructure for WSNs. This infrastructure is composed by an embedded platform, EPOSMote II [2], and a stack of communication protocols (TCP/IP/C-MAC). The use of TCP provides end-to-end reliability and ordered delivery. In order to not redo TCP work, e.g. acknowledgement and packet retransmission, we make use of the Configurable Medium Access Control (C-MAC) in the data link layer [3]. To this end we configured C-MAC in a very simplistic form, using only CSMA and backoff periods. To countermeasure the security threats present in WSNs we make use of data encryption and authentication through the network base station. Our authentication scheme has a temporal property that protects the network against replay attacks.

Cryptography algorithms are extremely expensive in terms of execution time because they require many arithmetic and logic operations to be executed, which makes traditional general-purpose processor inefficient for this scenario. Hardware acceleration for cryptographic algorithms not only enhances the performance of the security systems but also leaves the computing resources available to a more useful work [4]. Thus, we make use of the hardware-assisted security mechanism present in EPOSMote II to encrypt and decrypt all necessary data using the Advanced Encryption Standard (AES) [5].

In Section II we describe our infrastructure in details. In Section III we present an evaluation of our implementation, followed by our conclusions in Section IV.

II. Building a Trustful Infrastructure for WSNs

In this Section we describe the EPOSMote II platform, the AES algorithm, our communication protocol stack (TCP/IP/C-MAC), and our security threats countermeasures.

A. EPOSMote

The EPOSMote is an open hardware project [2]. The project main objective is delivering a hardware platform to allow research on energy harvesting, biointegration, and MEMS-based sensors. The EPOSMote II platform focus on modularization, and thus is composed by interchangeable modules for each function. Figure 1 shows the development kit which is slightly larger than a R$1 coin.

Figure 2 shows an overview of the EPOSMote II architecture. Its hardware is designed as a layer architecture composed by a main module, a sensoring module, and a power module. The main module is responsible for processing and communication. It is based on the Freescale MC13224V microcontroller [6], which possess a 32-bit ARM7 core, an

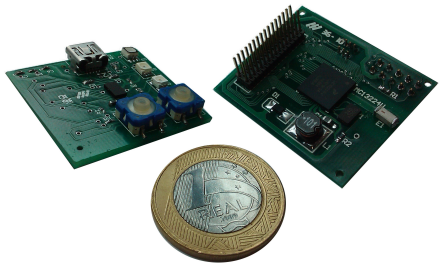

Figure 1: EPOSMote II SDK side-by-side with a R$1 coin. On the left the sensoring module. On the right the main module.

IEEE 802.15.4-compliant transceiver, 128kB of flash memory, 80kB of ROM memory and 96kB of RAM memory. We have developed a startup sensoring module, which contains some sensors (temperature and accelerometer), leds, switches, and a micro USB (that can also be used as power supply).

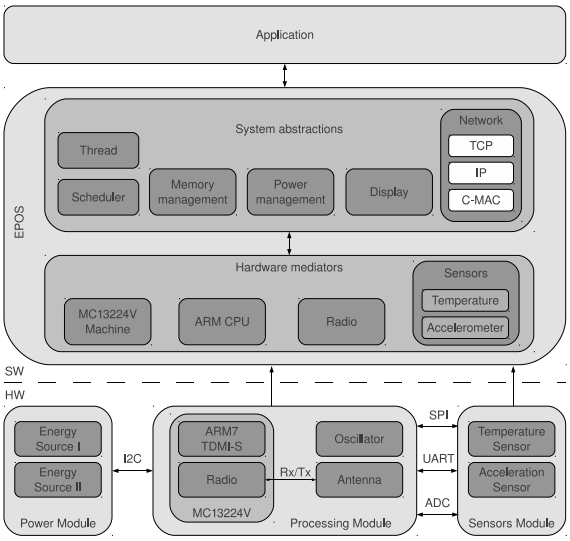

Figure 2: Architectural overview of EPOSMote II.

B. AES

Most symmetric-key algorithms require simple hash, rotation, or scrambling operations, which can be efficiently implemented in hardware or software. On the other hand, asymmetric-key algorithms require exponential operations over a field modulo a large prime number, which are more complex than symmetric-key operations. Therefore, the symmetric-key technology is more viable on resource constrained low-end devices than the asymmetric-key technology. Reason why most of security protocols in WSNs literature are based on symmetric-key technology [7].

The Advanced Encryption Standard is a symmetric-key algorithm considered to be resistant against mathematical attacks. It consists in a block cipher containing a 128-bit block size, with key sizes of 128, 192, and 256 bits. The hardware-assisted security mechanism present in EPOSMote II supports only key sizes of 128 bits. It also supports three encryption modes: Counter (CTR), Cipher Block Chaining-Message Authentication Code (CBC-MAC), and the combination of these two modes Counter with CBC-MAC (CCM). CCM mode provides both confidentiality and authentication, and thus is the mode we use.

C. C-MAC

C-MAC is a highly configurable MAC protocol for WSNs realized as a framework of medium access control strategies that can be combined to produce application-specific protocols [3]. It enables application programmers to configure several communication parameters (e.g. synchronization, contention, error detection, acknowledgment, packing, etc) to adjust the protocol to the specific needs of their applications.

Figures 3, 4, and 5 presents C-MAC architecture. Each activity in theses diagrams is executed by a microcomponent which can have different implementations. These microcomponents alongside with the flow control can be combined to produce application-specific protocols. By using static metaprogramming techniques, microcomponents representing activities that do not make sense for a certain protocol can be completely removed. When an activity is removed, its inputs are forwarded to the activity targeted by its outputs, still maintaining the original flow semantics.

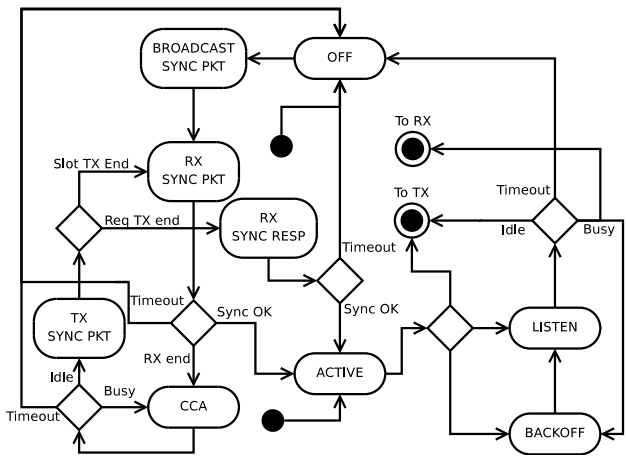

Figure 3: C-MAC Synchronization Activity Diagram.

The main C-MAC configuration points include:

Physical layer configuration: These are the configuration points defined by the underlying transceiver (e.g. frequency, transmit power, date rate).

Synchronization and organization: Provides mechanisms to send or receive synchronization data to organize the network and synchronize the nodes duty cycle.

Collision-avoidance mechanism: Defines the contention mechanisms used to avoid collisions. May be comprised of a carrier sense algorithm (e.g. CSMA-CA), the exchange of contention packets (*Request to Send* and *Clear to Send*), or a combination of both.

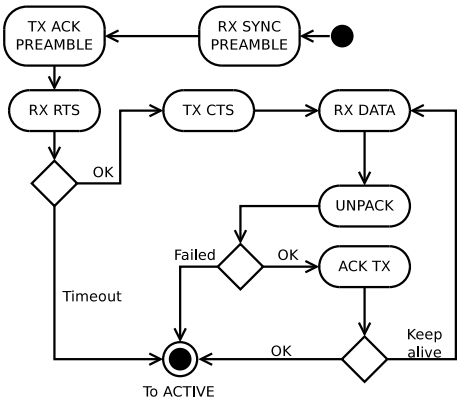

Figure 4: C-MAC Reception Activity Diagram.

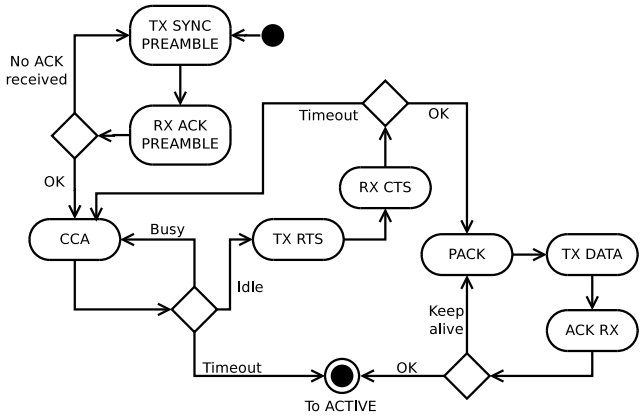

Figure 5: C-MAC Transmission Activity Diagram.

Acknowledgment mechanism: The exchange of *ack* packets to determine if the transmission was successful, including preamble acknowledgements.

Error handling and security: Determine which mechanisms will be used to ensure the consistency of data (e.g. CRC check) and the data security.

C-MAC was evaluated with figures comparable to other MAC protocols for WSNs. This is due to the use of static metaprogramming techniques (e.g. templates, inline functions, and inline assembly), which ensures that configurability does not come at expense of performance or code size.

Through C-MAC configurability we were able to adequate the MAC functionalities to our needs, instead of using a general non-optimal solution. Since TCP provides end-to-end reliability and ordered delivery we configured C-MAC in a very simplistic form, disabling acknowledgements and complex synchronization mechanisms. We decided to use just CSMA and backoff periods to avoid colisions.

D. TCP/IP

Reliable – in a sense of packet delivery – transport protocols such as TCP have been tuned for traditional networks made up of wired links and stationary hosts. TCP performs very well on such networks by adapting to end-to-end delays and packet losses caused by congestion, providing reliability by retransmitting any packet whose acknowledgment is not received within a predefined time. Due to the relatively low bit-error rates over wired networks, all packet losses are correctly assumed to be because of congestion. In the presence of the high error rates and intermittent connectivity characteristic of wireless links, TCP reacts to packet losses as it would in the wired environment. These measures result in an unnecessary reduction in the link's bandwidth utilization, thereby causing a significant degradation in performance in the form of poor throughput and very high interactive delays [8].

Focusing on sensor battery's useful life, *Braun and Dunkels* [9] introduces an approach to support energy efficient TCP operation in sensor networks. The concept called TCP Support for Sensor nodes (TSS) allows intermediate sensor nodes to cache TCP data segments and to perform local retransmissions when they assume that a cached segment has not been received by the successor node towards the destination, by not receiving an acknowledgement packet. TSS does not require any changes to TCP implementations at end points, and simulations show that it reduces the number of TCP data segment and acknowledgement transmissions in a wireless network. *Ganesh* [10] also introduces a similar mechanism which improves TCP performance, called TCP Segment Caching.

Elrahim et al. [11] also proposes an energy-efficient way to implement TCP protocol in scenarios with high losses. They present a modified Congestion Control Algorithm for WSN. Since a TCP sender constantly tracks the Round Trip Time (RTT) for its packets, and uses a timeout mechanism to trigger retransmissions in case an ACK is not received before the timer expires. By increasing the retransmission timeout value, they reduce the number of TCP segment transmissions that are needed to transfer a certain amount of data across a wireless sensor network with relatively high bit/packet error rates.

The size of TCP implementation also is very important when developing for resource-constrained sensors. NanoTCP [12] is a protocol stack for WSNs with reduced overhead. The low memory consumption of the protocol show its suitability to resource constrained devices. But nanoTCP is a simplified version of TCP protocol, not respecting its fields and not being compatible.

E. Security Threats Countermeasures

For key management we opted for a centralized key distribution scheme, as shown in Figure 6. Each sensor shares a unique key with the base station, and no one else. We assume that most of communication occurs between a node and the base station (data request and reply). In this case there is no overhead besides the encryption/decryption process. Two nodes can communicate with each other consulting the base station for authentication. But, since node to node communication is sporadic, it does not result in large communication overhead.

To countermeasure replay attacks we introduce a field in our packet which contains time information. This time information can be provided by a GPS device. Another alternative is to

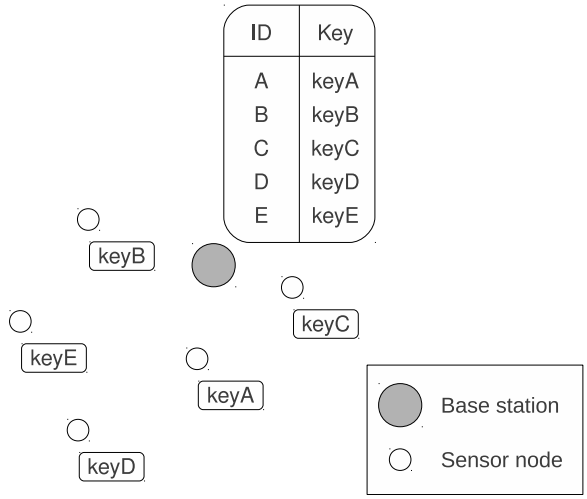

Figure 6: Network architecture.

use one of many clock synchronization protocols for WSNs present in the literature [13] [14] [15]. Using time, the network can protect itself from replay attacks discarding a message that was eavesdropped by an observer and used later in an attack. Figure 7 shows our packet format. Each packet includes our communication protocols headers, the application data, the current time, and the message authentication code.

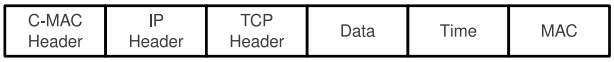

Figure 7: Packet format.

III. Experimental Results

We evaluated our proposal in three aspects: memory consumption, encryption/decryption time, and energy consumption. For all experiments we used: GCC 4.4.4 to compile the application; the MC13224V clock set to 24 MHz; payload of 32 bytes when using encryption, and 7 (request) and 6 (reply) bytes without encryption; and 4.5 dBm of TX power.

Figure 8 presents our test scenario. We used two EPOSMote II. One node is the base station and works as a gateway between the WSN and the user's network, and the other is a sensor node. The base station sends an encrypted temperature request every 10 seconds to the sensor node. The sensor node must decrypt the request, collect the environment temperature, and answer with an encrypted packet.

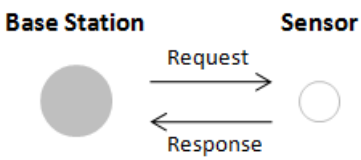

Figure 8: Evaluation test scenario.

A. Results

In order to analyse the memory footprint we used the arm-size tool, from GNU Binutils version 2.20. The results are shown in Table I. The *AES mediator* column represents the code size implemented in software to accomplish the encryption and decryption. Due to the hardware acceleration present in EPOSMote II, the code has a relatively small size. *App using AES* and *App without AES* columns presents the code size of the application using encryption or plain text, respectively. It is possible to notice that there is a difference between the value of *App using AES* and the sum of *App without AES* and *AES mediator*. This is due to the fact that not all methods from the *AES mediator* are executed in *App using AES*. The methods not executed are not present in the *App using AES* due to compiler optimizations.

Table I: Memory footprint.

Section	AES mediator	App using AES	App without AES
.text	1336 bytes	47184 bytes	45916 bytes
.data	0 bytes	217 bytes	217 bytes
.bss	10 bytes	5268 bytes	5268 bytes
TOTAL	1346 bytes	52669 bytes	51401 bytes

To evaluate the encryption/decryption and message authentication code check time we used an oscilloscope. We set a General Purpose Input/Output (GPIO) pin of the MC13224V microcontroller to 1 before the encryption process and to 0 after it, measuring the time between. We run the experiment for one minute. Table II shows the average time of each operation. These values also have an impact in the node's power consumption.

Table II: Encryption/decryption and MAC check processing time.

	Encryption	Decryption	MAC Check
Time	17 μs	15 μs	12 μs

Figure 9 shows the energy consumed by both applications, with and without AES, over time. As expected, the graph shows that the energy consumption behaves the same way in both applications, with *App using AES* consuming slightly more than *App without AES*. After 7 minutes executing, the applications have consumed 37.2 J and 36.8 J, respectively, a difference of 1%.

B. Discussion

For the sake of rigorous limits in WSN devices, *Huai et al.* [16] proposes to cut down duty cycles and decrease the energy consumption of executing the AES algorithm, by running both CTR and CBC-MAC in parallel. Similar to our scheme, their proposed design employs a full-hardware implementation to offload CPU. It uses an 8-bit data path, and a shared key expansion module with both AES cores, encryption and authentication, to guarantee rigorous requirements of area

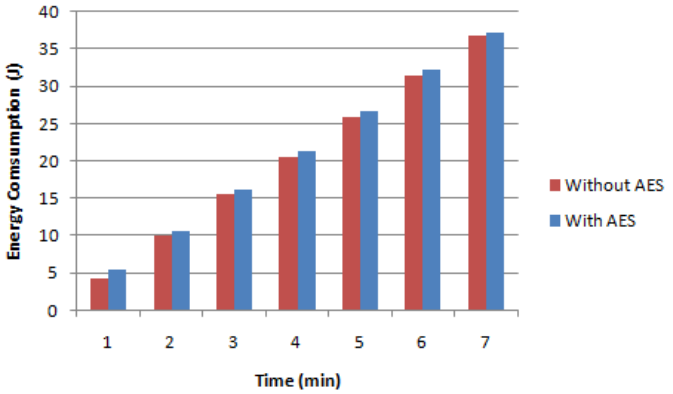

Figure 9: Energy consumption.

and energy. They achieved an encryption time of 71.6 ns for a payload of 17 bytes. Their parallel hardware acceleration provides better results when compared with the MC13224V hardware acceleration which works in a sequential execution mode.

Lee et al. [17] measures the encryption and decryption performance of AES-128 CBC algorithm on an 8-bit microcontroller. As a result, the time and CPU cycle grows proportional to the data size. They achieved an encryption/decryption time of 898 and 912 ms when using 32 bytes of data size.

IV. Conclusions

This paper presented a trustful infrastructure for WSNs within the EPOSMote project. By trustful we mean reliable and secure. This infrastructure is composed by the EPOSMote II platform, and a stack of communication protocols (TCP/IP/C-MAC). We used the hardware-assisted security mechanism present in EPOSMote II, which supports AES in CCM mode. Our proposal uses a centralized key distribution scheme, where each sensor shares a unique key with the base station. By introducing time information into our packets we countermeasure replay attacks.

We experimentally evaluated our proposal in terms of memory consumption, encryption/decryption time, and energy consumption. The results corroborate our proposal can provide confidentiality, authentication, integrity, and reliability without introducing a considerable overhead to the network.

References

[1] X. Fu, B. Graham, R. Bettati, and W. Zhao, "Active traffic analysis attacks and countermeasures," in *Proceedings of the 2003 International Conference on Computer Networks and Mobile Computing*, ser. ICCNMC '03. Washington, DC, USA: IEEE Computer Society, 2003, pp. 31–. [Online]. Available: http://portal.acm.org/citation.cfm?id=950787.950964

[2] EPOS Project. [Online]. Available: http://epos.lisha.ufsc.br

[3] R. V. Steiner, T. R. Mück, and A. A. Fröhlich, "C-MAC: a Configurable Medium Access Control Protocol for Sensor Networks," in *Proceedings of the IEEE Sensors Conference*, 2010.

[4] J.-T. Chang, S. Liu, J. Gaudiot, and C. Liu, "Hardware-assisted security mechanism: The acceleration of cryptographic operations with low hardware cost," in *Performance Computing and Communications Conference (IPCCC), 2010 IEEE 29th International*, dec. 2010, pp. 327 –328.

[5] NIST, "Advanced Encryption Standard (AES)," Federal Information Processing Standards Publication 197, November 2001.

[6] F. Semiconductor, "MC1322x Advanced ZigBee-Compliant Platform-in-Package (PiP) for the 2.4 GHz IEEE 802.15.4 Standard," Tech. Rep., 2010.

[7] Y. Zhou, Y. Fang, and Y. Zhang, "Securing wireless sensor networks: A survey," *IEEE Communications Surveys & Tutorials*, vol. 10, pp. 6–28, 2008.

[8] E. A. Hari Balakrishnan, Srinivasan Seshan and R. H. Katz, "Improving tcp/ip performance over wireless networks," in *Proceedings of the 1st annual international conference on Mobile computing and networking*. New York, NY, USA: ACM, 1995, pp. 2–11.

[9] T. V. Torsten Braun and A. Dunkels, "TCP support for sensor networks," in *Fourth Annual Conference on Wireless on Demand Network Systems and Services*, January 2007, pp. 162–169.

[10] R. A. S. Ganesh, "Energy efficient transport protocol for wireless sensor networks," in *2nd IEEE International Conference on Computer Science and Information Technology*, August 2009, pp. 464–468.

[11] S. E. R. Adel Gaafar A. Elrahim, Hussein A. Elsayed and I. Magdy M, "Improving TCP congestion control for wireless sensor networks," in *4th Annual Caneus Fly by Wireless Workshop*, Montreal, QC, Canada, June 2011, pp. 1–6.

[12] C. Jardak, E. Meshkova, J. Riihijarvi, K. Rerkrai, and P. Mahonen, "Implementation and Performance Evaluation of nanoIP Protocols: Simplified Versions of TCP, UDP, HTTP and SLP for Wireless Sensor Networks," in *IEEE Wireless Communications and Networking Conference*, March 2008, pp. 2474–2479.

[13] D. Fontanelli and D. Petri, "An algorithm for wsn clock synchronization: Uncertainty and convergence rate trade off," in *Advanced Methods for Uncertainty Estimation in Measurement, 2009. AMUEM 2009. IEEE International Workshop on*, july 2009, pp. 74 –79.

[14] D. Fontanelli and D. Macii, "Towards master-less wsn clock synchronization with a light communication protocol," in *Instrumentation and Measurement Technology Conference (I2MTC), 2010 IEEE*, may 2010, pp. 105 –110.

[15] A. Swain and R. Hansdah, "An energy efficient and fault-tolerant clock synchronization protocol for wireless sensor networks," in *Communication Systems and Networks (COMSNETS), 2010 Second International Conference on*, jan. 2010, pp. 1 –10.

[16] L. Huai, X. Zou, Z. Liu, and Y. Han, "An energy-efficient aes-ccm implementation for ieee802.15.4 wireless sensor networks," *Networks Security, Wireless Communications and Trusted Computing, International Conference on*, vol. 2, pp. 394–397, 2009.

[17] H. Lee, K. Lee, and Y. Shin, "Implementation and Performance Analysis of AES-128 CBC algorithm in WSNs," in *The 12th International Conference on Advanced Communication Technology*, 2010, pp. 243–248.