A Trustful Infrastructure for the Internet of Things based on EPOSMote

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

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

***Abstract*—This article describes the design, implementation and evaluation of a trustful infrastructure for the Internet of Things (IoT) based on EPOSMote. The infrastructure was built around EPOS' second generation of motes, which features an ARM processor and an IEEE 802.15.4 radio transceiver. It is presented to end users through a trustful communication protocol stack compatible with TCP/IP. Trustfulness was tackled at MAC level by extending C-MAC, EPOS native MAC protocol, with Advanced Encryption Standard (AES) capabilities that were subsequently used to encrypt and authenticate packets containing IP datagrams. Our authentication mechanism encompasses temporal information to protect the network against replay attacks. The infrastructure was designed bearing in mind the severe resource limitation typical of IoT devices. The prototype implementation was assessed for processing, memory, and energy consumption and strongly confirmed our assumptions.**

***Keywords*-Internet of Things; Wireless Sensor Networks; Trustfulness;**

I. Introduction

The idea of an Internet of Things (IoT) is quickly materializing through the adoption of RFID as a replacement for bar code along with the introduction of Near Field Communication (NFC) devices that are able to interface our daily-life objects with the Internet. However, the next steps towards a global network of smart objects will drive us through several large-scale, interdisciplinary efforts. In particular, security and privacy are issues that must be consistently addressed before IoT can make its way into people's lives.

Things in IoT will interact with each other and with human beings through a myriad of communication technologies, often wirelessly, and almost always subject to interference, corruption, eavesdropping and all kinds of cybernetic attacks. Most of the encryption and authentication techniques developed for the original Internet—the Internet of People that we use today—to handle impersonation, tampering, and replay attacks can in theory be applied to the IoT. However, the microcontrollers used in smart objects will seldom be able to put up with their requirements. Furthermore, IoT will be subject to particular conditions not so often faced by today's Internet devices. Some *Things* will send messages that will trigger immediate reactions from the environment. Capturing and reproducing one such valid message, even if it is encrypted and signed, could lead complex of systems such as roadways, factories and even future cities to misbehave. Some *Things* will harvest energy from the environment for hours before they can say something to the world. And when they speak, one will have to decide whether or not to believe in what they say without having a chance to further discuss the subject (at least not for a couple of hours). Conventional solutions such as transaction authentication and channel masking [1] are of little help in this context.

In this paper, we describe the design, implementation and evaluation of a trustful infrastructure for the IoT conceived with these pitfalls in mind. The infrastructure was built around EPOS' second generation of motes, EPOSMoteII, which features an ARM processor and an IEEE 802.15.4 radio transceiver [2]. It is presented to end users through a trustful communication protocol stack compatible with TCP/IP, which per-definition ensures end-to-end reliable and ordered delivery. Trustfulness is tackled at MAC level by extending C-MAC [3], EPOS native MAC protocol, with Advanced Encryption Standard (AES) [4] capabilities that were subsequently used to encrypt and authenticate packets containing IP datagrams. Our authentication mechanism also encompasses temporal information to protect the network against replay attacks. AES was chosen because the ARM-based version of EPOSMoteII features an AES hardware accelerator, without which the implementation of a cryptographic algorithm would be impractical.

Section II presents the related works and in Section III we describe the main components in the EPOSMoteII platform that were used as building blocks for the trustful IoT infrastructure being proposed here. Section IV describes the trustful infrastructure in details. In Section V we present an evaluation of our implementation, followed by our conclusions in Section VI.

II. Related Work

TinySec [5] defines a link-layer security architecture for Wireless Sensor Networks (WSNs), providing encryption and authentication. TinySec supports two different security options: authenticated encryption (TinySec-AE), and authentication only (TinySec-Auth). In authenticated encryption mode, TinySec encrypts the data payload according to the Skipjack block cipher [6] and authenticates the packet with a Message

Authenticity Code (MAC). The MAC is computed over the encrypted data and the packet header. In authentication only mode, TinySec authenticates the entire packet with a MAC, but the data payload is not encrypted. The inclusion of a MAC to ensure authenticity and integrity has a cost on radio usage and, consequently, in energy consumption. This is because the hash values commonly represent a long sequence of bits—the length of a MAC determines the security strength of a MAC function [7]. TinySec achieves low energy consumption by reducing the MAC size, hence decreasing the level of security provided. TinySec also does not attempt to protect against replay attacks, and does not discuss how to establish link-layer keys. TinySec was implemented in TinyOS and runs on Mica, Mica2, and Mica2Dot platforms, each one using Atmel processors. TinySec has 3000 lines of nesC code [8] and the implementation requires 728 bytes of RAM and 7146 bytes of program space.

MiniSec [9] is a secure network layer protocol for WSNs which attempts to solve the known problems of TinySec. MiniSec accomplishes this by combining three techniques. First, it employs a block cipher mode of operation that provides both privacy and authenticity in only one pass over the message data. Second, MiniSec sends only a few bits of the Initialization Vector (IV)—a block of bits used by some operating modes to randomize the encryption, producing distinct ciphertexts from the same plaintext over time—while retaining the security of a full-length IV per packet. In order to protect against replay attacks and reduce the radio's energy consumption, it uses synchronized counters, but only sending the last bits of the counter along with each packet. However *Jinwala et al.* showed that such scheme requires costly resynchronization routines to be executed when the counters shared are desynchronized (packets delivery out-of-order) [10].

III. Proposed Infrastructure Building Blocks

In this Section we describe the main components in the EPOSMoteII platform that served as building blocks for the infrastructure being proposed here, including mote itself, EPOS Configurable Medium Access Control Protocol (C-MAC), EPOS TCP/IP protocol stack and the AES hardware accelerator.

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 EPOSMoteII 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 EPOSMoteII 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, storage, and communication. The model used in this research features a 32-bit ARM7 processor, 128kB of flash, 96kB of RAM,

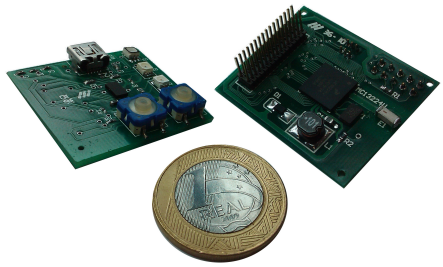

Figure 1: EPOSMoteII SDK side-by-side with a R$1 coin. On the left the sensoring module. On the right the main module.

and an IEEE 802.15.4-compliant radio transceiver. We have developed a startup sensoring module, which contains some sensors (e.g. temperature and accelerometer), leds, switches, and a micro USB (that can also be used as power supply).

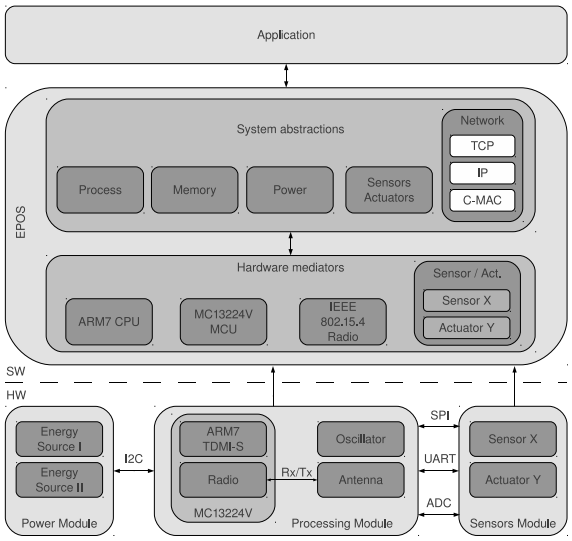

Figure 2: Architectural overview of EPOSMoteII.

B. 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, 5, and 4 depict 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.

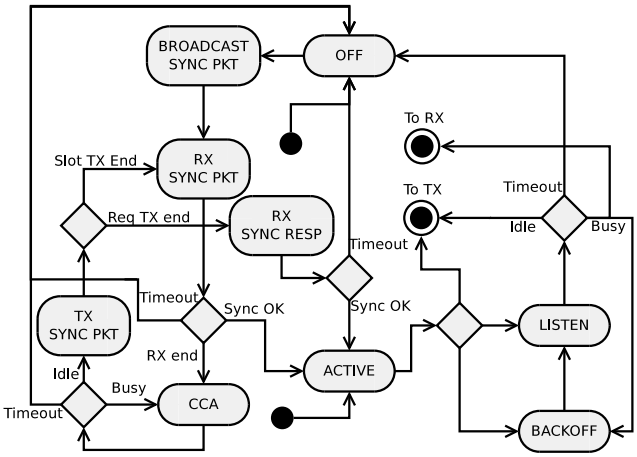

Figure 3: C-MAC Synchronization Activity Diagram.

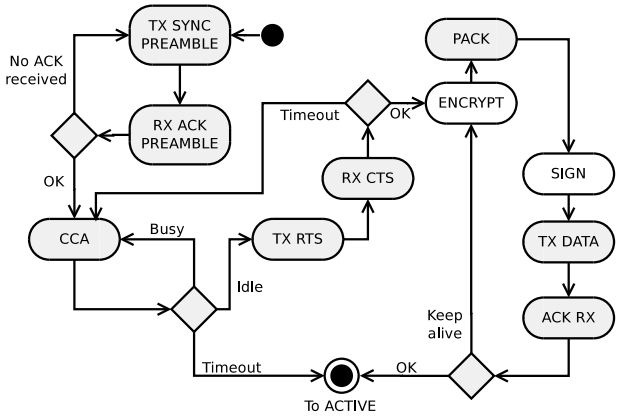

Figure 4: C-MAC Transmission Activity Diagram.

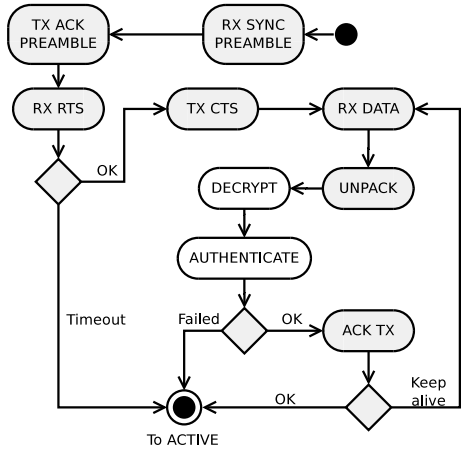

Figure 5: C-MAC Reception Activity Diagram.

The microcomponents responsible for trustfulness are outlined in Figures 4, and 5. `ENCRYPT` is responsible for encrypting the payload. `SIGN` attaches the time-stamp, which also goes encrypted, and the message authentication code to the packet. `DECRYPT` decrypts the payload. And `AUTHENTICATE` verifies if both the time-stamp and authentication code are valid.

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.

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.

When configured to mimic preexisting MAC protocols, like B-MAC for instance, C-MAC delivers comparable performance. 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 [3]. In this way, C-MAC's instances are fully customized at compile-time and yield extremely lean run-time MACs.

C-MAC high configurability was essential to the research being presented here, for it enabled us to customize the MAC protocol to closely match the requirements of upper level protocols (instead of using a general, non-optimized MAC). Since TCP provides end-to-end reliability and ordered delivery, we configured C-MAC in a rather simplistic way, disabling acknowledgments and complex synchronization mechanisms. We use CSMA and back-off periods to avoid collisions.

C. TCP/IP

TCP is a key protocol for the trustful IoT platform being proposed here, for it ensures ordered delivery of packets. However, ordinary TCP implementations have been tuned for decades to traditional networks, made up of wired links and stationary hosts. TCP now performs very well on such networks. Its acknowledgement and flow control mechanisms have been optimized to efficiently handle congestion, presumably the unique significant cause for packet losses on such low-error rate networks. In the presence of higher error rates and intermittent connectivity typical of IoT wireless links, traditional TCP implementations would continue to react to packet losses in the same way, causing a significant degradation of

performance observed by peers as poor throughput and high latency [11].

EPOS TCP/IP stack was conceived in this context, considering also the limited resource availability typical of IoT devices and bearing in mind that many *things* will operate on batteries. The current IPv4 implementation uses TCP's window-based flow control mechanism to implement a *rendezvous* protocol and thus virtually eliminates buffer management on IoT nodes. Peers announce buffer availability for a single message at a time by adjusting the window length in acknowledgements messages accordingly. Several optimizations have also been conducted to keep IP datagrams in pace with IEEE 802.15.4 127-byte MTU, a challenging goal for the upcoming IPv6 version.

Energy efficiency is sought in EPOS TCP/IP stack by incorporating the pheromone concept behind the Ant-based Dynamic Hop Optimization Routing Protocol (ADHOP) [12] as the IP routing metric. ADHOP is a self-configuring reactive routing protocol for WSNs that uses ant colony optimization to discover and maintain routes, specially on mobile sceneries. ADHOP pheromone concentration and evaporation rates are dynamically adjusted considering global information collected and disseminated by ants, such as hop counting and round trip time to destination, but also local information that encompasses resource availability, such as residual battery and buffer memory. A node forwarding too many packets because it is on an strategic location will adjust pheromone to favor other routes as soon as it realizes its resources are being drained too quickly.

D. AES Accelerator

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. As any computationally secure cryptographic algorithm, AES is extremely expensive in terms of execution time because it requires many arithmetic and logic operations to be executed. The need to perform such operations makes traditional general-purpose processors inefficient for this scenario.

The use of hardware acceleration for cryptographic algorithms not only enhances the performance of security systems but also leaves the computing resources available to a more useful work [13]. Thus, we make use of the hardware-assisted security mechanism present in EPOSMoteII to encrypt and decrypt all necessary data. The mechanism uses a key size of 128 bits, and 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). The last provides both confidentiality and authentication and therefore was the choice for our infrastructure.

IV. Trustfulness for the Proposed Infrastructure

IoT devices will often communicate through a wirelessly technology, however a wireless channel is open to everyone allowing any radio interface configured at the same frequency band to monitor or participate in communications. This provides a convenient way for attackers [14].

In order to avoid undesired attacks a secure infrastructure must provide:

- **Confidentiality** prevents unauthorized access to information. It can be accomplished by encrypting critical parts of a packet before its transmission, then only the allowed receivers can access the data, decrypting the packet.
- **Authenticity** confirms the origin of a message, It can be accomplished by signing the message.
- **Integrity** ensures that the message was not modified for some reason, such as by an attacker or simply unwanted errors. It can be accomplished with a checksum.

The rest of this Section describes the strategy that was adopted to enrich EPOSMoteII with a level of trustfulness, including the incorporation of AES by C-MAC, and countermeasures to selected security threats.

A. 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.

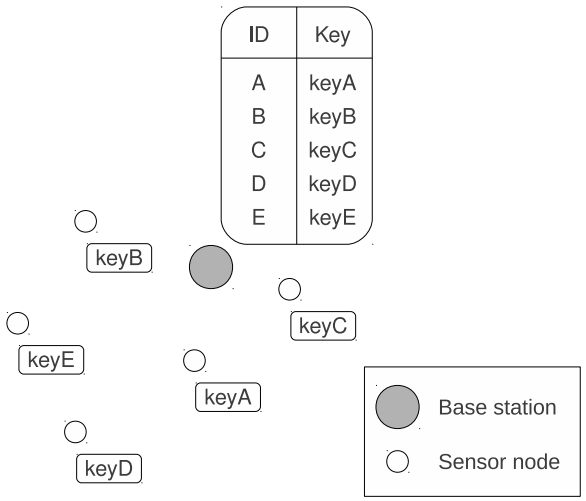

Figure 6: Network architecture.

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 use one of many clock synchronization protocols for WSNs present in the literature [15] [16] [17]. 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.

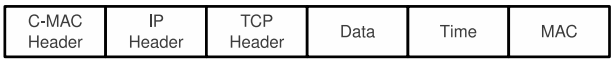

Figure 7: Packet format.

V. Proposed Infrastructure Evaluation

We gauged the implementation of the trustful IoT infrastructure proposed in this paper in respect to three aspects: memory consumption, encryption/decryption time, and energy consumption. For all experiments, we used GCC 4.4.4 to compile the application and the run-time support system (i.e. EPOS). EPOSMoteII ARM processor clock was set to 24 MHz. Messages were adjusted to carry a payload of 16 bytes when encryption was activated and 7 (request) and 6 (reply) bytes otherwise. EPOSMoteII radio transceiver was adjusted to transmit at 4.5 dBm.

Figure 8 illustrates our test scenario. We used two EPOSMoteII. One node acts as a base station for the local IoT, interfacing its nodes to the ordinary Internet[1], while the other one is a sensor node. The base station broadcasts encrypted temperature requests every 10 seconds. The sensor node decrypts the request, collects the required data, and sends back a signed and encrypted reply.

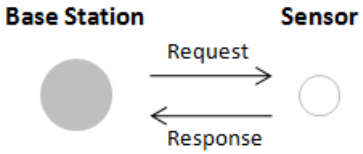

Figure 8: Trustful IoT infrastructure evaluation scenario.

In order to obtain the memory footprint of our implementation, we used the `arm-size` tool that is part of GNU Binutils. Results are shown in Table I. The *AES mediator* column represents the code needed to interact with the AES hardware accelerator available on EPOSMoteII in order to accomplish encryption, decryption, and authentication. *App using AES* column presents the code size of the application using the proposed trusted infrastructure and the *App without AES* column the size when using the original, plain text, TCP/IP stack. 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 used in *App using AES*. Mediator methods that are not effectively invoked by the client program are eliminated during compilation.

We used an oscilloscope to measure the time needed to encrypt, decrypt and authenticate messages in our infrastructure. A General Purpose Input/Output (GPIO) pin in EPOSMoteII is connected to the oscilloscope and set to high before the intended procedure is executed and reset to low right after. We run the experiments for one minute and calculated the averages shown in Table II. Obtained values, besides confirming the efficiency of the implementation in terms of execution time, also have positive impact in the node's battery lifetime.

[1]For a larger scale experiment, the gateway would rather be configured to provide some sort of NAT service between both realms, thus alleviating the address limitation of IPv4.

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

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 the time. The small increase in energy consumption for *App using AES* arises from the efficient usage of the hardware accelerator available in EPOSMoteII. After 10 minutes executing, the difference is minimal (53.2 J with AES and 52.6 J without), and after 1 hour, the applications have consumed 319.5 J and 315.5 J, respectively, a difference of 1.25%.

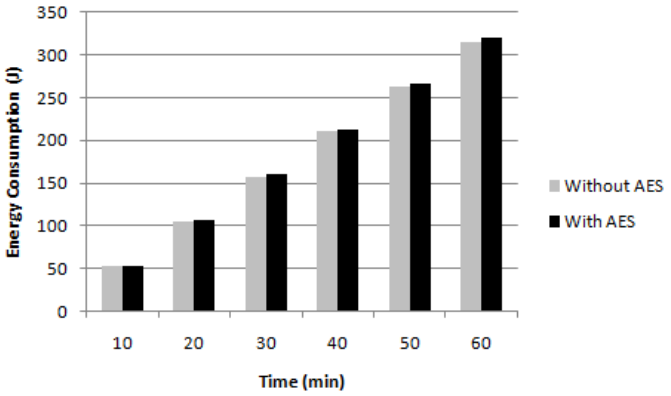

Figure 9: Energy consumption.

A. Discussion

Huai 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 [18]. Similarly to our scheme, their design employs a hardware accelerator to offload CPU. It uses an 8-bit data path and a shared key expansion module with both AES cores, encryption and authentication. 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 sequential AES hardware accelerator in the FreeScale MC13224V present in EPOSMoteII. On the other hand, Lee evaluated the performance of AES-128 CBC on an 8-bit microcontroller [19]. Time and CPU cycle grew proportionally to payload size, reaching 449 ms to encrypt and 456 ms to decrypt 16 bytes of data.

VI. Conclusions

This paper presented a trustful infrastructure for the IoT developed within the realm of project EPOS. By trustful we mean *reliable* and *secure*. Aspects such as people privacy in respect to traffic pattern analysis and data dependability have not been considered in this paper. The proposed infrastructure is implemented around the EPOSMoteII platform and delivered to end users through a trustful communication protocol stack compatible with TCP/IP. Trustfulness for the infrastructure was achieved through a combination of mechanisms. From TCP/IP, we inherited reliable and ordered end-to-end packet delivery. C-MAC was enriched with AES-based encryption and authentication. It now also time-stamps messages to prevent replay attacks. Authentication is performed using a centralized key distribution scheme in which each sensor shares a unique key with the local base station. We experimentally evaluated our proposal in terms of memory consumption, encryption/authentication time, and energy consumption. The results confirm that the proposed infrastructure is able to provide confidentiality, authentication, integrity, and reliability without introducing excessive overhead to a network of things, a key step in making the Internet of Things a daily reality.

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

[5] C. Karlof, N. Sastry, and D. Wagner, "Tinysec: a link layer security architecture for wireless sensor networks," in *Proceedings of the 2nd international conference on Embedded networked sensor systems*, ser. SenSys '04. New York, NY, USA: ACM, 2004, pp. 162–175. [Online]. Available: http://doi.acm.org/10.1145/1031495.1031515

[6] (1998, may) National security agency, skipjack and kea algorithm specifications. [Online]. Available: http://cryptome.org/jya/skipjack-spec.htm

[7] H.-M. Sun, S.-Y. Chang, A. Tello, and Y.-H. Chen, "An authentication scheme balancing authenticity and transmission for wireless sensor networks," in *Computer Symposium (ICS), 2010 International*, dec. 2010, pp. 222 –227.

[8] D. Gay, P. Levis, R. von Behren, M. Welsh, E. Brewer, and D. Culler, "The nesc language: A holistic approach to networked embedded systems," in *Proceedings of the ACM SIGPLAN 2003 conference on Programming language design and implementation*, ser. PLDI '03. New York, NY, USA: ACM, 2003, pp. 1–11. [Online]. Available: http://doi.acm.org/10.1145/781131.781133

[9] M. Luk, G. Mezzour, A. Perrig, and V. Gligor, "Minisec: A secure sensor network communication architecture," in *Information Processing in Sensor Networks, 2007. IPSN 2007. 6th International Symposium on*, april 2007, pp. 479 –488.

[10] D. Jinwala, D. Patel, S. Patel, and K. Dasgupta, "Replay protection at the link layer security in wireless sensor networks," in *Computer Science and Information Engineering, 2009 WRI World Congress on*, vol. 1, 31 2009-april 2 2009, pp. 160 –165.

[11] 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.

[12] A. Okazaki and A. Frohlich, "Ad-zrp: Ant-based routing algorithm for dynamic wireless sensor networks," pp. 15–20, may. 2011.

[13] 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.

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

[15] 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.

[16] 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.

[17] 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.

[18] 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.

[19] 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.