Implementation of a Secure Wireless Sensor Network Collection Protocol

Ariel D. Godinho, Bruno T. de Oliveira and Cíntia B. Margi

***Abstract*— Security is an important issue to be addressed in Wireless Sensor Networks (WSNs), since nodes are physically vulnerable and could be compromised. But adding security to WSN increases battery consumption and communication delays, therefore these problems need to be considered when implementing security for existing protocols. In this work, we present our results on the implementation of security for the Collection Tree Protocol, a well-known routing protocol for WSN.**

***Keywords*— Wireless sensor networks, applied cryptography, network layer, overhead.**

I. Introduction

Wireless Sensor Networks (WSNs) have been used to support several different applications, mainly related to monitoring, detection and tracking. Nodes in a WSN are typically battery-powered and resource constrained (i.e. limited amount of memory, processing and communication), and communicate through a multihop ad hoc network [2]. They are comprised of several autonomous sensors that cooperate with each other to perform sensing and data transmission.

WSNs' characteristics make them vulnerable to a large number of attacks, given the utilization of wireless communication and the exposure of the sensor nodes, which can be deployed in unmonitored and unprotected areas. Additionally, a WSN can be heterogeneous and have mobile nodes, enabling a compromised node to harm the whole network. Therefore, it is important to use an end-to-end security solution to provide confidentiality, source authentication, message integrity and replay protection [6].

Security solutions based on cryptography add control data to messages, thereby increasing processing and memory (RAM and ROM) usage. The increase in message length and higher processing demand may also increase network latency and energy consumption [6]. Additionally, the link layer frame size of the standard utilized in the majority of the sensor nodes (IEEE 802.15.4 [5]) has a maximum size of 127 bytes. Hence, an efficient WSN security solution should have: low communication overhead, low processing overhead and low memory footprint.

This paper presents results for the integration of cryptographic mechanisms to the Collection Tree Protocol (CTP) [3], a collection protocol widely used in TinyOS [4] applications. We consider the TelosB [8] as the sensor mote, which has an 8 MHz microcontroller processor with 48 KiB of programmable flash (ROM) and 10 KiB of RAM.

Ariel D. Godinho, Bruno T. de Oliveira and Cíntia B. Margi. Universidade de São Paulo, E-mails: {ariel.godinho, brunotrevizan, cintia}@usp.br.

II. Overview

The security mechanisms added to CTP was implemented following the IEEE 802.15.4 standard security specification. It is based on the Counter with CBC-MAC (CCM*) mode with Cipher Block Chaining Message Authentication Code (CBC-MAC). CCM* is an algorithm that combines Counter Mode (CTR) as encryption mode used to provide confidentiality. CBC-MAC provides source authentication and message integrity, and its security level depends on the message authentication code (MAC) length.

CCM* in the IEEE 802.15.4 standard includes the following operation modes: 0 - No Security; 1 - Authentication (32-bit MAC); 2 - Authentication (64-bit MAC); 3 - Authentication (128-bit MAC); 4 - Encryption only; 5 Authenticated Encryption (32-bit MAC); 6 - Authenticated Encryption (64-bit MAC); 7 - Authenticated Encryption (128-bit MAC).

Since CTP is hardware-independent, we used the Advanced Encryption Standard (AES) with 128-bit key size, implemented by Simplício [9], as underlying block cipher. We also had to adapt the initialization vectors to the application layer, since some inputs specified by the standard are originally generated by hardware.

III. Implementation

This implementation consists of a modified CTP protocol that uses a CCM implementation in nesC and C and provides an additional CTPSecurity interface used to set the encryption parameters. The inputs are as follows:

- A message *m* and its length, *l(m)*.
- An authenticating string *a* and its length, *l(a)*, as defined in Table I.
- A 13 bytes nonce *N*, as defined in Table II.
- A 128 bits key *Key*.
- The Operation Mode, defining the MAC length *M*.

TABLE I
CCM* Authentication String defined by software

Bytes:	2	2	1	1	4
Content:	senderID	0xFFFF	opMode	collect_id	seqNumber

TABLE II
CCM* Nonce defined by software

Bytes:	2	2	4	1	4
Content:	senderID	0xFFFF	0x00	collect_id	seqNumber

The CTP header had to be modified to include a *SecurityOptions* flag and a larger Sequence Number, changing the

header size from 8 bytes to 12 bytes. The MAC has to be sent with the message, decreasing the maximum payload size depending on the security mode.

In order to avoid all nodes sharing the same key, and to stop the root node from having a table with all keys, this implementation uses an auxiliary java class to send the *MasterKey* to a node. This key is used by the CTP Forwarding Engine in conjunction with the *NodeID* to generate the *DerivedKey* through CBC-MAC. The *MasterKey* is then erased and the *DerivedKey* is stored in Flash Memory, persisting through reboot. The root can easily derive the node-specific key to each received message through the same algorithm, using the *NodeID* in its header and the *MasterKey*.

The feature that enables storing and loading the key in Flash is optional, making necessary to declare *USE_FLASH_KEY* and pass the *MasterKey* as a parameter when executing the java class, as exemplified in the *installtelosb* script.

IV. Results

A. Computation Time

We measured the encryption time of the AES software implementation [9] and the CC2420 hardware implementation [1], being 7400 μs and 1600 μs respectively. Therefore, the software implementation is almost 5 times slower.

Next we measured the average execution time of the CTR and CBC-MAC. As shown in Figure 1, the execution time of these algorithms are directly related to the amount of block ciphers used, both taking 7.7 ms for each input block. The CCM* implementation uses a 10 bytes *a* string and both the CTR and CBC-MAC algorithms. Thus the total execution time is basically their sum, plus the time needed to compute the associated data, as shown in Figure 1.

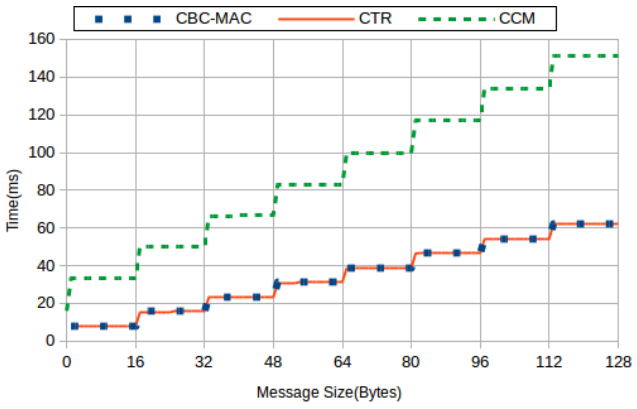

Fig. 1. Encryption algorithms execution times for different message sizes.

In the worst case scenario, a 94-byte message with security mode set to 6 sent directly to the root is expected to take at least 620 ms [7] to be transmitted, resulting in a maximum increase of 18% in the time needed to send and compute the message. If a 64-byte message, which is a common size, is used with security mode set to 6 and needs 5 hops to reach the root, the overhead of using security can be as low as 1%.

B. Memory

We tested the default MViz application[1] and a modified version that was used to test the security in CTP, comparing the used ROM and RAM as shown by the TelosB compiler. Results are shown in Table III.

TABLE III
MViz memory usage comparison

Implementation	ROM (bytes)	RAM (bytes)
MViz - 114 bytes	30128	3474
Secure MViz	30442	4056
Secure MViz - Key stored in Flash	34636	4232

The message length was set to 114 bytes (the maximum alowed by CTP) to ensure an accurate comparison. Comparing default MViz with the Secure MViz implementation with the key stored in Flash memory, the increase is of 4428 bytes in ROM and 756 bytes in RAM, representing an overhead of 14,7% and 21,7% respectively.

V. Final Considerations

WSN security is an important matter in real circumstances and must be taken into consideration when developing a network with multiple nodes. This work presents an implementation that tackles this issue. As shown by our experiments, security can be added to the CTP layer with little overhead, making this implementation viable in most situations.

Acknowledgments

This work is partially funded by São Paulo Research Foundation (FAPESP) under grants #2015/10976-0, #2013/15417-4 and #2014/06479-9. Cíntia Borges Margi is supported by CNPq research fellowship #307304/2015-9.

References

[1] Chipcon. CC2420 Datasheet. `http://focus.ti.com/lit/ds/symlink/cc2420.pdf`, 2007.

[2] D. Culler, D. Estrin, and M. Srivastava. Overview of sensor networks. *Computer Magazine*, 37(8):41–49, 2004.

[3] O. Gnawali, R. Fonseca, K. Jamieson, D. Moss, and P. Levis. Collection tree protocol. In *Proceedings of the 7th ACM Conference on Embedded Networked Sensor Systems*, SenSys '09, pages 1–14, New York, NY, USA, 2009. ACM.

[4] J. Hill, R. Szewczyk, A. Woo, S. Hollar, D. Culler, and K. Pister. System architecture directions for networked sensors. *SIGPLAN Notices*, 35(11):93–104, 2000.

[5] IEEE Standard. IEEE 802.15.4: Wireless medium access control (MAC) and physical layer (PHY) specifications for low-rate wireless personal area networks (WPANs), 2006.

[6] 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*, pages 162–175, New York, NY, USA, 2004. ACM.

[7] C. B. Margi, B. T. de Oliveira, G. T. de Sousa, M. A. S. Jr, P. S. L. M. Barreto, T. C. M. B. Carvalho, M. Naslund, and R. Gold. Impact of operating systems on wireless sensor networks (security) applications and testbeds. pages 1 –6, aug. 2010.

[8] MEMSIC. Telosb datasheet. `http://www.memsic.com/userfiles/files/DataSheets/WSN/telosb_datasheet.pdf`, 2011.

[9] M. A. Simplício. Aes: C version for 8-bit platforms (128-bit keys, encryption-only). `http://www.larc.usp.br/~mjunior/en/downloads/index.html`, 2014.

[1] `https://github.com/tinyos/tinyos-main/tree/master/apps/MViz`