An Integrated System for Secure Code Distribution in Wireless Sensor Networks

Nicola Bui
Consorzio Ferrara Ricerche
via Saragat 1
44122 – Ferrara, Italy
Email: buincl@unife.it

Osman Ugus
NEC Europe Ltd.
Kurfürsten-Anlage 36
6911 – Heidelberg, Germany
Email: osman.ugus@nw.neclab.eu

Moreno Dissegna, Michele Rossi and Michele Zorzi
Department of Information Engineering (DEI)
University of Padova
via Gradenigo 6/B 35131 – Padova, Italy
Email: moreno.d@hotmail.it, {rossi,zorzi}@dei.unipd.it

***Abstract*—This paper presents a Secure Code Update (SCU) system for Wireless Sensor Networks (WSNs). This solution achieves different security goals. First, through a dedicated authentication protocol it provides protection against the corruption of code images during their dissemination. Authentication routines exploit a lightweight asymmetric T-time signature algorithm [1] and allow the out-of-order reception of data blocks. Second, confidentiality is provided through the implementation of an optimized symmetric encryption suite, designed to leverage the processing capabilities offered by typical IEEE 802.15.4 radios. Last, our solution offers protection against denial of service attacks. In the first part of the paper we present the integration of this security system with the SYNAPSE++ reprogramming protocol [2], focusing on the description of the security suite as well as on its salient implementation aspects. After this, we present experimental results that demonstrate the effectiveness against security attacks and quantify the loss in performance due to the addition of security components to SYNAPSE++.**

***Keywords*-Wireless Sensor Networks; security; authentication; Denial of Service; Secure Code Update.**

I. Introduction and Related Work

Updating the code running on Wireless Sensor Network (WSN) nodes is a necessary service, which can be used to remove bugs or to add new functionalities after the sensors have been deployed. In open, public, untrusted, or even hostile environments, protecting the code update operation against adversarial interference is an essential requirement. Otherwise, an insecure code update may provide an adversary with a backdoor rendering any security mechanism useless, and may even become a serious risk for the owner. There are mainly three security aspects to be considered in the desing of a Secure Code Update (SCU) mechanism. First, a SCU mechanism shall only allow the load of authentic code images into the nodes' memory. Second, a SCU mechanism must detect the dissemination of a modified or corrupted code image as early as possible. The need is to avoid unnecessary energy consumption due to the propagation of a corrupted image over multiple hops and to the re-transmission of its pages. Finally, a SCU mechanism must keep the secrecy of a code image being disseminated. The need is to prevent eavesdroppers from gaining information on the content of the code image.

Related Work: regarding the first two SCU objectives, *code image authentication* and *Denial of Service (DoS) resilience*, several works [3]–[6] have been published. These solutions are based on the Deluge [7] reprogramming protocol. The basic idea behind these solutions is to bootstrap the code image authentication using a digital signature and to propagate the security of the signature through the code image by means of *hash chains*. The main difference among them is the number of hash chains used, the granularity of the data being hashed and the amount of flexibility in verifying the packets when arriving out-of-order. We observe that out-of-order delivery of packets often occurs in wireless networks due to, e.g., multiple parallel transmissions of the same content by different nodes, collisions, etc. Some SCU protocols require strict in-order delivery, which unavoidably makes the packets received out-of-order useless. Being able to process out-of-order packets speeds up the secure reprogramming phase with savings in terms of time and energy.

Also, the need for implementing a digital signature scheme such as ECDSA or RSA on sensor nodes is the main problem with [3]–[6]. Some efficient ECDSA and RSA implementations have been recently reported for use in sensor nodes [8], [9]. However, using them for SCU protocols would leave too little ROM space for a shared implementation of security routines and code update logic. Ugus et al. [1] proposed a purely symmetric signature scheme called T-TimeSA [1] to overcome this problem.

Other symmetric SCU solutions based on Deluge are presented in [10] and [11]. [10] exploits an approach similar to μTesla [12], exploiting the delayed disclosure of the MAC keys used to authenticate the packets. The main drawback of this scheme is the need for a loose and secure clock synchronization between all sensor nodes and the base station. An approach based on multiple one-way key chains is proposed in [11]; this work does not consider DoS attacks and the possible out-of-order delivery of packets.

Tan et al. [13] proposed a Deluge-based SCU mechanism. This protocol, which supports code image encryption, is an extension of [11] with a weak authenticator for a digital signature. The packets of a code image are encrypted by using the hash of a packet as the encryption key for the same packet. The adversary can decrypt the code image and inject

malicious packets to perform DoS attacks by compromising a sensor node. Therefore, we believe that the level of security provided by this solution for code image confidentiality is no better than simply using the link layer encryption that we adopt in our solution.

Bohli et al. [14] presented a security mechanism for code update protocols based on rateless LT Codes. Their scheme does not consider code image confidentiality and does not support the out-of-order delivery of pages.

Our Contribution: the SCU system proposed in this paper has been designed to securely reprogram, in a completely decentralized fashion, dense Wireless Sensor Networks (WSNs) spanning over multiple hops. The system disseminates the new software using SYNAPSE++ [2] as the underlying protocol for data distribution. SYNAPSE++ is a reprogramming protocol featuring routing functions and an error recovery engine based on computationally efficient fountain codes. It builds routing trees on-the-fly, so as to adapt to any (connected) topology. A dedicated "program-start" message is sent by the Base Station (BS) prior to the dissemination to wake up sleeping sensor nodes and to prepare the network for the subsequent programming phase. After that, the software is spread to the network nodes following an epidemic approach similar in style to that of Deluge [7], i.e., the nodes with the new software send advertisements (ADVs) to their neighbours. The neighbours, in turn, send back request (REQ) messages in case they still have to receive the software. This procedure is iterated until the entire network is programmed. SYNAPSE++'s dissemination protocol is detailed in [2], [15].

Our security system integrates into SYNAPSE++ the following security functionalities: confidentiality, protection against the dissemination of corrupted images and against Denial of Service (DoS) attacks. Each security component has been optimized accounting for the memory requirements of the selected sensor platform (TelosB nodes [16]) and the interactions with SYNAPSE++'s dissemination protocol.

This paper is structured as follows. Section II presents the SCU system, discussing our design criteria and showing how we integrated SCU routines into SYNAPSE++. Section III discusses the security level of each security component. Section IV shows experimental results on the effectiveness of our SCU system against security attacks and on the loss in performance due to the addition of our security components to a non-secure system. Section V concludes the paper.

II. System Description

In a WSN it is particularly important to secure the code update process with regard to the following security requirements:

1) **Code image confidentiality:** the secrecy of the disseminated code images must be protected against unauthorized entities. The updated code images need to be kept secret to prevent eavesdroppers from gaining information on new images being disseminated to the sensor nodes.
2) **Bogus code image protection:** the code image needs to be authenticated in order to prevent the WSN from running corrupted software, which could make the WSN useless or even involve a serious risk for the owner. Hence, no sensor node shall write an unauthenticated code image into its memory. This amounts to ensuring *authenticity* and *integrity* of the data.
3) **Denial of Service protection:** the reprogramming protocol shall prevent the nodes from performing energy exhausting operations when an adversary interferes with the wireless medium by sending modified packets; this type of attack is referred to as Denial of Service (DoS). Note that modified or corrupted data shall be detected as early as possible during the code update in order to avoid unnecessary energy consumption due to the propagation of malicious code over multiple hops and to keep the parts of the code image that need retransmission at a minimum.

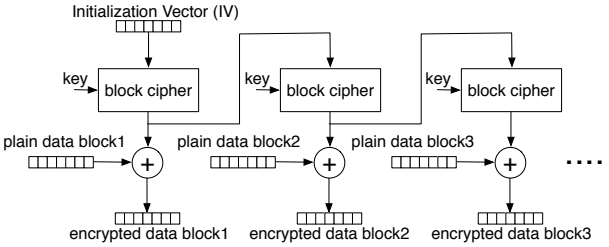

Figure 1. Encryption diagram for the OFB operation mode.

In the following, we discuss how these requirements are achieved by our SCU system.

A. Code Image Confidentiality

Confidentiality is achieved through a suitable encryption algorithm. The first issue to face when designing an encryption/decryption suite is the selection of a block cipher. In this respect, conflicting results have been obtained in the literature, depending on the optimization goals, such as memory requirements, performance or energy consumption (see [12], [17]–[23]). The primary goal in our case is to keep a small ROM footprint, because a considerable amount of space is already taken on the nodes by the reprogramming protocol. Keeping memory occupancy and execution time into account, we opted for the exploitation of the hardware implementation of AES-128 (i.e., Rijndael [24]) provided by the CC2420 radio chip of the TelosB platform.

Also, when the length of the data being encrypted is larger than the block size of the employed encryption routine, block ciphers must be combined with a so called "operation mode". The CC2420 radio chip only implements the encryption function of AES-128 [25]. Thus, OFB, CFB and CTR are the only possible operation modes and, among

them, we selected OFB due to its good performance [17]. Fig. 1 shows how OFB is used to encrypt data files of arbitrary length. First, the plain code image is subdivided into a suitable number of "plain data blocks" of smaller size. The key is a shared key uploaded into the sensor nodes during their deployment and the Initialization Vector (IV) is a randomly chosen bit-string. Each plain data block is encrypted using the block cipher and, for $i \geq 2$, the result from block $i-1$ is used as the initialization vector for block i. The scheme of the decryption routine is identical with "plain data block" and "encrypted data block" swapped.

We note that code images are encrypted only once at the BS while sensor nodes only need to decrypt a received code image before writing it into the program memory. Hence, only the decryption routines need to be stored into the nodes. In addition, since we use the AES-128 implementation of the CC2420 radio chip, there are issues related to the concurrent use of the radio chip for decryption and communication (e.g., transmitting and receiving packets). We carefully synchronized these activities to keep this into account.

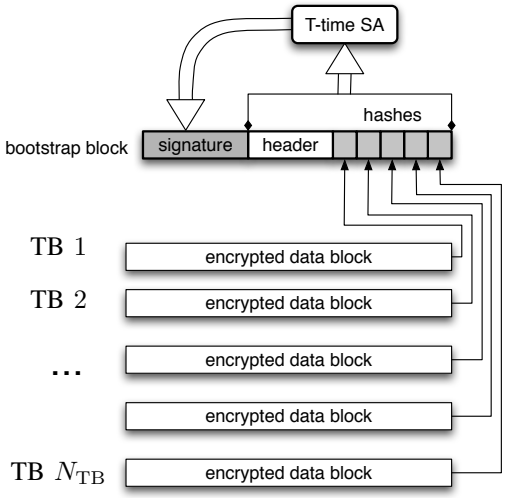

Figure 2. Authentication of the code image at the Base Station.

B. Bogus Code Image Protection

Security against bogus code image insertion attacks requires protecting authenticity and integrity of the code image being disseminated. Authenticity and integrity of the code image can be protected by signing the entire code image with a digital signature scheme. Since sensor nodes accept a new code image only if its signature is valid, performing a bogus code image attack requires a forgery on the employed signature scheme. The success probability of such a forgery is negligible, when a proper digital signature scheme is used. Even though this approach allows detecting any corruption occurring due to channel errors or malicious activities, it is not very efficient due to the following reasons. First, an invalid signature would imply a corruption on the code image, but it would not indicate exactly which part of the image is corrupted. In such a case, retransmission of the entire code image would be required. This would not be very efficient and could be exploited by adversaries to perform resource exhausting DoS attacks. Second, due to its size, a code image usually cannot be stored entirely into the sensor nodes' RAM. Therefore, the encrypted code image is split into a number of Transport Blocks (TBs) whose size complies with the memory requirements of the nodes. Thus, the reprogramming protocol disseminates the program image one block at a time. This implies that the verification of a received code image must be performed over the individual blocks when they are received rather than over the entire code image.

To ensure this property, a code image is authenticated at the BS before its dissemination, as illustrated in Fig. 2. First, the code image is encrypted as described in Section II-A. Thus, it is subdivided into a number N_{TB} of TBs. The hash value for each transport block is computed using a hash function (i.e., SHA1). Finally, these hash values together with some header information, i.e., the software version number and the Initialization Vector (IV) used to encrypt the N_{TB} plain data blocks, are signed with a digital signature mechanism (which will be discussed shortly). Hence, we introduce a further block, called "bootstrap block", containing the hash values of the N_{TB} TBs, the signature message, the initialization vector, and some header information.

Sensor nodes exploit the bootstrap block to verify the authenticity of a received code image as follows. Upon reception of the bootstrap block, they first verify the signature message; this will validate all the data in the bootstrap block, including the hash values of the following N_{TB} TBs of data. At this point, these N_{TB} hashes are stored into the nodes memory and will be used to verify the remaining data blocks when they are received. Note that this allows the verification of the following N_{TB} blocks in an *out-of-order fashion* as the N_{TB} hashes are calculated independently for each block. This is a desirable feature for a dissemination protocol as transport blocks are often received out-of-order in practical cases.

We note that the bootstrap block can be signed with a conventional digital signature scheme such as RSA or ECDSA. However, these conventional digital signature schemes are very expensive in terms of computational and memory requirements. Moreover, even though some efficient implementations have been recently proposed for use in sensor nodes [8], [9], using them for securing our reprogramming protocol would leave too little ROM space for the dissemination protocol. To overcome this drawback, a signature scheme called T-TimeSA was developed in [1] that merely uses a symmetric primitive i.e., a hash function. This allows a small ROM footprint while, at the same time, providing a

sufficient level of security.

C. DoS Protection

Denial of Service is a subtle type of attack in WSNs. An adversary sends forged or replayed messages that once received at one or multiple nodes will force them to go through energy expensive activities: the overall aim is that of depleting the nodes' resources. As an example, in a routing protocol, these messages may contain false route requests or route failures, whose reception will force the nodes to check the validity of the current routing path or, even worse, to start a new route discovery procedure (which is an expensive operation for the node itself and for the entire network). During a secure reprogramming event, fake messages may contain invalid data that will be detected by the authentication check but that will nevertheless force the receiver to perform long and energy expensive operations.

Immunizing a node from these attacks requires methods to distinguish between messages sent by trusted nodes and forged or replayed messages sent by malicious ones. These methods combine packet-level authentication with an appropriate set of counters called "nonces" to guarantee the freshness of received messages. The 802.15.4/ZigBee standard [18] includes a security suite specification to provide access control, data encryption, frame integrity and sequential freshness (a counter is incremented at each transmission; a check is performed at the receiver on this counter to make sure that incoming packets are not replayed by an adversary). Thus, the CC2420 radio chip provides these functionalities and we exploit them in our solution to provide protection against DoS attacks.

As a security service we used the CBC-MAC hardware implementation of the CC2420 radio chip. Given a message and a key, the sender creates a message authentication code (MAC) tag by using the CBC-MAC module. The MAC tag is appended to the packet being transmitted. The receiver, which shares the key with the sender, recalculates the MAC tag and compares it against the received one to guarantee the message is authentic. The key is a shared MAC key uploaded in the nodes during their deployment.

We note that the CBC-MAC authentication does not guarantee data freshness but only provides a means to check that a received message originates from a node possessing the shared secret key. In order to provide packet freshness, we had to analyze the reprogramming protocol and add the appropriate set of nonces to each type of message. How we did so is explained in the next subsection.

D. Integrating DoS Protection with SYNAPSE++

SYNAPSE++ is described in [2], but we will shortly recall here the basics of its protocol. The messages sent by SYNAPSE++ during the dissemination are: advertisement (ADV), request (REQ), data (DATA) and command (CMD).

The CMD message is the simplest one. It is a control message that is broadcast over the network by the BS, telling the nodes to perform some operations, such as FLASH formatting, rebooting, or loading stored images. The ADV message advertises the image to be disseminated, informing the receiver(s) about the blocks that are available at the sender. Once a node receives some ADVs from its neighbours, it decides which block(s) should be requested; the request is performed through REQ messages. Once a node receives some REQ messages, it decides which of the requesting nodes should be satisfied and starts sending DATA messages related to one of the requested TBs. The selection of which TB should be requested (upon the reception of an ADV by a neighbour) and which should be subsequently satisfied (i.e., disseminated by a node that previously sent ADVs) is performed taking priorities into account, see [2]. Furthermore, data transmission periods for nodes within a neighborhood are loosely synchronized.

Design of the nonces: nonces must be designed to ensure freshness of authentic received packets. In our case, a first measure is to maintain a counter of the number of reprogramming sessions, and include it in each transmitted packet. Although this guarantees that the message is not a replay of a message sent in a previous dissemination, it does not suffice to ensure freshness because an attacker could successfully replay a message sent within the current dissemination session.

We propose to use four nonces, one for each type of packet, i.e., ADV, REQ, DATA and CMD. Each of them contains the session counter, stored in each node's permanent memory, which is incremented by the BS at each dissemination session and updated by each node when a new session is detected.

- *Nonces for ADVs:* in addition to the session counter, ADVs also contain a cycle counter. The cycle counter counts the number of ADV-REQ-DATA cycles performed in a nodes neighbourhood in the current dissemination session. Session and cycle counters are the nonce for ADV messages. The cycle counter is updated at the reception of new ADV packets, and incremented based on a timer when no ADVs are received. The latter procedure is implemented to counteract the following attack. Imagine the following transmission setup: a node A is transmitting ADVs to a node B; a third malicious node C sends a jamming signal so that node B will fail to receive some of the ADVs sent by A. After this, node C may reuse these lost ADVs that may be deemed as valid by node B. However, this attack fails if the cycle counter is updated at node B even when it does not receive packets. In fact, node C cannot forge a valid authentication code for a new counter value, i.e., for a counter value that has not yet been used.
- *Nonces for REQs:* REQs contain a cycle counter to ensure freshness, which works as the one contained in

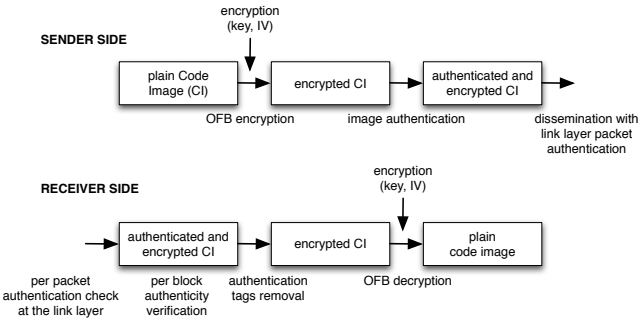

Figure 3. Security flow.

the ADVs.

- *Nonces for DATA messages:* as ADVs and REQs, DATA packets contain the session id. In addition, they also contain a block id and a further identifier that locates the packet within the current TB. These three ids are used to infer whether a packet is a duplicate. Duplicate packets are discarded.
- *Nonces for CMDs:* CMD messages are broadcast by the BS and must be received by all nodes. Hence, to ensure freshness it is sufficient to maintain a counter, incremented by the BS and updated by each node each time a CMD is received.

E. Security Flow

In Fig. 3 we show the overall security flow for encryption, decryption and authentication. At the sender side (i.e., the BS), the plain code image is encrypted with AES-128 using the OFB operation mode as detailed in Section II-A. This takes as input the plain code image, an Initialization Vector (IV), an encryption key, and returns the encrypted image. After this, the encrypted image is authenticated using the procedure illustrated in Section II-B. We note that the initialization vector is chosen randomly at the BS and included in the bootstrap block. Instead, the encryption key is a shared key and is uploaded in the nodes during the deployment phase.

At the receiver side (i.e., sensor nodes), the plain code image is retrieved reversing the order of the previous security operations, i.e., applying the authentication check first and finally decrypting the encrypted image using the CC2420 hardware implementation of AES-128. Checking first the authenticity has several advantages. In fact, a data packet upon its reception is decrypted only after being authenticated. This allows detection of data corruption due to channel errors and/or adversarial interference without having to decrypt the data in the packet. This has advantages in terms of resources such as energy and processing time.

Using the same key for authentication and encryption is not secure. Therefore, our secure code update mechanism requires sharing two secret keys which are common to all nodes (and which may be pre-loaded into the nodes during their deployment). The first key is an encryption key needed for encrypting/decrypting the code images as described above. The second key is a message authentication key needed to authenticate the communication packets at the link layer as described in Section II-C.

III. Security Analysis

Code image confidentiality protection: there is no known attack which breaks the security of AES-128 (128-bit AES) faster than the exhaustive search (i.e., brute force attack) [26]. Thus, using the AES-128 encryption of the CC2420 radio chip practically provides unbreakable security in protecting confidentiality of the code images. However, this does not hold if an adversary physically captures a node and reads the secret key from its internal memory; using the secret key the adversary can decrypt the data. This security threat may be overcome through the execution of specialized key distribution schemes. However, these are often cumbersome as they require the exchange of a large number of packets and imply a considerable amount of additional code, which would not fit the memory requirements of most sensor platforms. Nevertheless, our current implementation for code image confidentiality is selected as a suitable compromise among offered security level, energy efficiency and memory occupancy. In addition, compromising the secret key does not impair the security level against the bogus code image protection which is our main security goal.

Bogus code image protection: the code image is protected by the signature message transmitted in the bootstrap block. The private key used for generating the signature is only known to the trusted Base Station and cannot be accessed by the adversary. For the T-TimeSA [1] with n as the security parameter we have that the signature size is on average $ln/2$ bits, where $l = n + \lfloor \log(n) \rfloor + 1$. In this case, it can be shown that [1] a polynomial run-time restricted adversary only has a probability smaller than or equal to $P_n = 2^{-n}$ to produce a valid forgery under a known message attack. The signed bootstrap block contains the hash values of the following N_{TB} TBs, which propagate the security of the signature over the entire code image. To keep the same security level of the signature for the transport blocks, their hash values are truncated to n bits. The probability of forging a valid signature for a second pre-image attack against a truncated hash value is still in the range of P_n [25]. Finding a second pre-image is needed to break the authentication with a known message attack.

DoS protection: in [27] it is formally proven that CBC-MAC is a secure message authentication code if the underlying block cipher is secure. Since the implementation of CBC-MAC on the CC2420 radio chip uses AES-128, when properly used, a tag created with it provides an n-bit security with an n-bit MAC key. Attacks against this security level are impractical for polynomial run-time adversaries. We propose to truncate the MAC tags to trade security

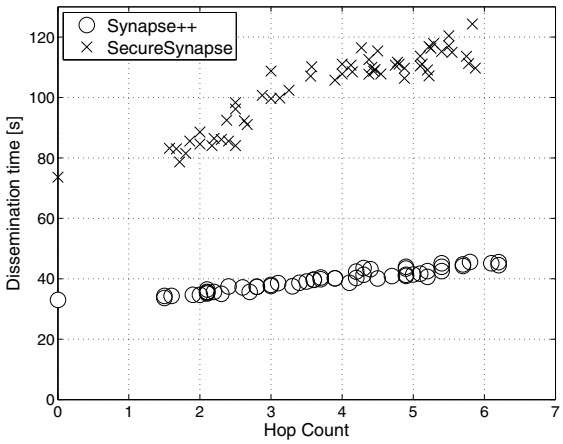

Figure 4. Dissemination time, no attacks.

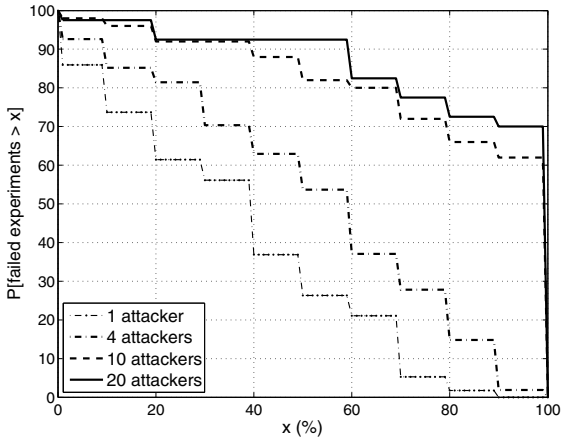

Figure 5. Bogus code image insertion attack.

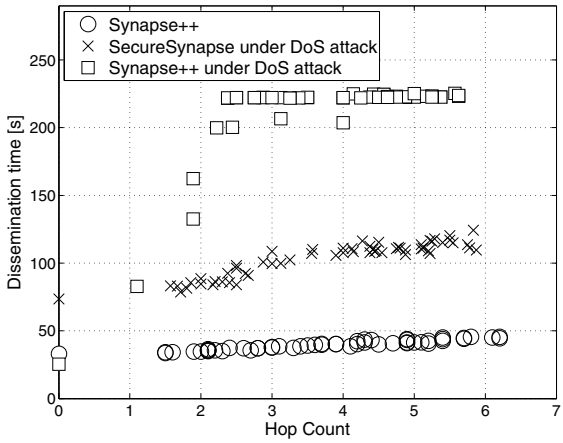

Figure 6. Dissemination time, DoS attack.

with performance as described in the following section. We recall that compromising the MAC-key would fully break the packet-level authentication security. However, it would not impair the security level against the bogus code image insertion which is our main security goal.

A. Choice of Security Parameters

We propose concrete security parameters for bogus code image protection, code image confidentiality protection, and DoS protection. A security mechanism is assumed to provide long-term security against small organizations if the fastest attack breaking it requires more than 2^{80} steps [28]. Therefore, in order to have sufficient security against bogus code image insertion attacks, we select the signature parameter for 80-bit security and the hash values of the transport blocks are truncated to 80 bits. The protection against DoS attacks by packet modification can be weaker, since it is an online attack that has to be executed on-the-fly. A security level of 2^{32} is reasonable [28]. Therefore, we suggest truncating the MAC tags to 32 bits to achieve a reasonable security against DoS attacks. Breaking the security of AES-128 encryption requires 2^{128} steps which offers long-term confidentiality protection [28].

IV. Experimental Results

We performed an experimental comparison between our SCU protocol and SYNAPSE++ [2] for a program image of 10 kbytes in three different cases: 1) no attacks, 2) bogus code image insertion attack and 3) DoS attack. The tests have been performed in a real WSN deployment at the Department of Information Engineering of the University of Padova, considering 55 Telosb nodes and one base station, see [29]. Each point in the following graphs represents the performance of a single node; its abscissa is the maximum hop count distance from the base station during a run of the dissemination experiment, averaged over 25 runs. For the remaining configuration parameters see [2].

The dissemination time under normal operating conditions, see Fig. 4, increases as a consequence of the addition of security features, becoming almost twice that of the insecure SYNAPSE++. In fact, the presence of the signature makes the image 465 bytes larger. Also, the sensor nodes spend about 410 milliseconds to validate the signature that is received within the first TB, while the remaining TBs can only be disseminated when this operation is complete. Finally, the transmission time for each TB is 30% longer for the SCU system to accommodate the additional time required by link layer security and TB authentication.

Fig. 5 shows the effect of the bogus image insertion attack, quantifying the probability, for any given node in the WSN, that this node will fail to retrieve the original image in more than $x\%$ of the experiments, where x is the value in the abscissa (different curves are plotted for varying number of attackers). This attack does not compromise the reliability of the dissemination for SCU, which is therefore not shown.

Fig. 6 quantifies the dissemination time in the presence of DoS attacks. Only one curve is shown for SCU as DoS

has little impact on its performance. Instead, SYNAPSE++ without DoS protection is highly impacted and its dissemination time becomes markedly longer than that of SCU as the number of malicious nodes increases.

V. Conclusions

In this paper we presented and validated a Secure Code Update system featuring a full range of security mechanisms for WSNs. Our main result was the integration of diverse security procedures within an efficient dissemination protocol obtaining a small impact on the performance, while adhering to the hardware limitations of sensor nodes.

Acknowledgment

This work has been supported by the FP7 EU project "SENSEI, Integrating the Physical with the Digital World of the Network of the Future," Grant Agreement Number 215923, http://www.ict-sensei.org.

References

[1] O. Ugus, D. Westhoff, and J.-M. Bohli, "A ROM-friendly Secure Code Update Mechanism for WSNs Using a Stateful verifier T-time Signature Scheme," in *ACM WiSec*, Zurich, Switzerland, Mar. 2009.

[2] M. Rossi, N. Bui, G. Zanca, L. Stabellini, R. Crepaldi, and M. Zorzi, "SYNAPSE++: Code Dissemination in Wireless Sensor Networks using Fountain Codes," *Transactions on Mobile Computing*, 2010, accepted for publication.

[3] P. K. Dutta, J. W. Hui, D. C. Chu, and D. E. Culler, "Securing the deluge Network programming system," in *IEEE IPSN*, Nashville, TN, USA, Apr. 2006.

[4] P. E. Lanigan, R. Gandhi, and P. Narasimhan, "Sluice: Secure Dissemination of Code Updates in Sensor Networks," in *IEEE ICDCS*, Reading, UK, May 2006.

[5] J. Deng, R. Han, and S. Mishra, "Secure code distribution in dynamically programmable wireless sensor networks," in *IEEE IPSN*, Nashville, TN, USA, Apr. 2006.

[6] S. Hyun, P. Ning, A. Liu, and W. Du, "Seluge: Secure and DoS-Resistant Code Dissemination in Wireless Sensor Networks," in *IEEE IPSN*, St. Louis, MO, USA, Apr. 2008.

[7] J. W. Hui and D. Culler, "The dynamic behavior of a data dissemination protocol for network programming at scale," in *ACM SenSys*, Baltimore, MD, USA, Nov. 2004.

[8] B. Driessen, A. Poschmann, and C. Paar, "Comparison of innovative signature algorithms for WSNs," in *ACM WiSec*, Alexandria, VA, USA, Apr. 2008.

[9] A. Liu and P. Ning, "TinyECC: A Configurable Library for Elliptic Curve Cryptography in Wireless Sensor Network," in *IEEE IPSN*, St. Louis, MO, USA, Apr. 2008.

[10] D. H. Kim, R. Gandhi, and P. Narasimhan, "Exploring Symmetric Cryptography for Secure Network Reprogramming," in *IEEE ICDCS*, Toronto, Ontario, Canada, Jun. 2007.

[11] H. Tan, S. Jha, D. Ostry, J. Zic, and V. Sivaraman, "Secure multi-hop network programming with multiple one-way key chains," in *ACM WiSec*, Alexandria, VA, USA, Apr. 2008.

[12] A. Perrig, R. Szewczyk, J. D. Tygar, V. Wen, and D. E. Culler, "SPINS: security protocols for sensor networks," *Wireless Networks*, vol. 8, no. 5, pp. 521–534, Sep. 2002.

[13] H. Tan, S. Jha, D. Ostry, J. Zic, and V. Sivaraman, "Secure multi-hop network programming with multiple one-way key chains," in *ACM WiSec*, Alexandria, VA, USA, Apr. 2008.

[14] J.-M. Bohli, A. Hessler, O. Ugus, and D. Westhoff, "Security enhanced multi-hop over the air reprogramming with Fountain Codes," in *IEEE LCN*, Zurich, Switzerland, Oct. 2009.

[15] M. Rossi, G. Zanca, L. Stabellini, R. Crepaldi, A. F. Harris III, and M. Zorzi, "SYNAPSE: A Network Reprogramming Protocol for Wireless Sensor Networks using Fountain Codes," in *IEEE SECON*, San Francisco, CA, US, Jun. 2008.

[16] Crossbow, "TelosB Mote Platform Datasheet." [Online]. Available: http://www.xbow.com/

[17] J. Großschädl, S. Tillich, C. Rechberger, M. Hofmann, and M. Medwed, "Energy evaluation of software implementations of block ciphers under memory constraints," in *DATE*, Nice, France, Apr. 2007.

[18] Y. W. Law, J. Doumen, and P. Hartel, "Survey and benchmark of block ciphers for wireless sensor networks," *ACM Transactions on Sensor Networks*, vol. 2, no. 1, pp. 65–93, Feb. 2006.

[19] D. Jinwala, D. Patel, and K. Dasgupta, "Optimizing the Block Cipher and Modes of Operations Overhead at the Link Layer Security Framework in the Wireless Sensor Networks," in *ICISS*, Hyderabad, India, Dec. 2008.

[20] J. Deng, R. Han, and S. Mishra, "A Performance Evaluation of Intrusion-Tolerant Routing in Wireless Sensor Networks," in *IEEE IPSN*, Palo Alto, CA, USA, Apr. 2003.

[21] G. Guimaraes, E. Souto, D. Sadok, and J. Kelner, "Evaluation of security mechanisms in wireless sensor networks," in *SENET*, Montreal, Canada, Aug. 2005.

[22] C. Karlof, N. Sastry, and D. Wagner, "TinySec: a link layer security architecture for wireless sensor networks," in *ACM SenSys*, Baltimore, MD, USA, Nov. 2004.

[23] M. Luk, G. Mezzour, A. Perrig, and V. D. Gligor, "MiniSec: A Secure Sensor Network Communication Architecture," in *IEEE IPSN*, Cambridge, MA, USA, Apr. 2007.

[24] J. Daemen and V. Rijmen, "AES Proposal: Rijndael," AES algorithm original submission, Sep. 1999.

[25] A. Menezes, P. van Oorschot, and S. Vanstone, *Handbook of Applied Cryptography*, 1st ed. CRC Press, 1996.

[26] A. Biryukov, O. Dunkelman, N. Keller, D. Khovratovich, and A. Shamir, "Key Recovery Attacks of Practical Complexity on AES Variants With Up To 10 Rounds," Cryptology ePrint Archive, Report 2009/374, 2009.

[27] M. Bellare, J. Kilian, and P. Rogaway, "The security of the cipher block chaining message authentication code," *Journal of Computer and System Sciences*, vol. 61, no. 3, pp. 362–399, Dec. 2000.

[28] ECRYPT, "Yearly Report on Algorithms and Keysizes," D.SPA.7 Rev. 1.0, ICT-2007-216676 ECRYPT II, 2009.

[29] P. Casari, A. P. Castellani, A. Cenedese, C. Lora, M. Rossi, L. Schenato, and M. Zorzi, "The Wireless Sensor Networks for City-Wide Ambient Intelligence (WISE-WAI) Project," *Sensors*, vol. 9, no. 6, May 2009.