Link-layer Security in TSCH Networks: Effect on Slot Duration

Savio Sciancalepore, Malisa Vucinic, Giuseppe Piro, Gennaro Boggia, Thomas Watteyne

TRANSACTIONS ON EMERGING TELECOMMUNICATIONS TECHNOLOGIES
Trans. Emerging Tel. Tech. 0000; **00**:1–14
DOI: 10.1002/ett

Link-layer Security in TSCH Networks: Effect on Slot Duration

Savio Sciancalepore[1], Mališa Vučinić[2], Giuseppe Piro[1], Gennaro Boggia[1], and Thomas Watteyne[3]

[1] Department of Electrical and Information Engineering, Polytechnic of Bari, Bari (Italy), e-mail: {name.surname}@poliba.it.
[2] Grenoble Alps University, Grenoble Institute of Technology, CNRS Grenoble Informatics Laboratory, 38000 Grenoble, France. e-mail: {name.surname}@imag.fr
[3] Inria, Paris, France. e-mail: {name.surname}@inria.fr

ABSTRACT

The IEEE802.15.4e-2012 standard is widely used in multi-hop wireless Industrial Internet of Things (IIoT) applications. In the Time-Slotted Channel Hopping (TSCH) mode, nodes are synchronized, and time is cut into timeslots. A schedule orchestrates all communications, resulting in high reliability and low power operations. A timeslot must be long enough for a node to send a data frame to its neighbor, and for that neighbor to send back an acknowledgment. Shorter timeslots enable higher bandwidth and lower latency, yet the minimal timeslot duration is limited by how long link-layer security operations take. We evaluate the overhead of link-layer security in TSCH networks in terms of minimal timeslot length, memory footprint, and energy consumption. We implement full link-layer security on a range of hardware platforms, exploring different hardware/software implementation strategies. Through an extensive measurement campaign, we quantify the advantage of hardware accelerations for link-layer security, and show how the minimal duration of a timeslot varies between 9 ms and 88 ms for the most common configuration, depending on hardware support. Furthermore, we also highlighted the impact that the timeslot duration has on both high-level application design and energy consumption. Copyright © 0000 John Wiley & Sons, Ltd.

1. INTRODUCTION

The Internet of Things (IoT) is generally envisioned as a networked system of smart interacting objects (i.e., sensors, machines, vehicles, smart phones, tablets, etc.) [1]. According to recent forecasts of leading companies of the sector [2]-[4], around 50 billions devices are expected by 2020 to be part of the IoT. Furthermore, the introduction of the IoT into the manufacturing environment is leading to the emerging Industrial Internet of Things (IIoT), leveraging a class of low-power wireless networks used in critical applications such as industrial process monitoring and automation. These networks are receiving significant attention from standardization bodies, and a new wave of IIoT products is hitting the market [5].

A cornerstone technology for the IIoT is the Time-Slotted Channel Hopping (TSCH). TSCH networks were introduced in 2006 in the Time Synchronized Mesh Protocol (TSMP) [6]. The core of this proprietary solution was standardized in WirelessHART [7] in 2007 and in the IEEE802.15.4e TSCH amendment [8] in 2012. TSCH is becoming a key enabling technology for the IIoT. Tens of thousands of TSCH networks are operating today [9]. Furthermore, it is receiving lots of attention from the standardization community, for example through the IETF 6TiSCH working group [10], and widespread adoption in happening [4].

In TSCH networks, all nodes are synchronized, and the time is cut into timeslots. During a timeslot, a node sends a frame to its neighbor, and that neighbor sends back a link-layer acknowledgment indicating the successful reception. Specifically, the whole communication is orchestrated by a schedule which indicates, to each node, what to do in each slot: transmit, listen or sleep. In general, it is preferable to have a time slot as small as possible: the shorter the timeslot, the lower the latency and the higher the throughput of a network. However, its value must be minimized by taking into account the computational capabilities actually offered by a real platform, which define the amount of time required to correctly execute all the operations triggered by the aforementioned communication schedule (more details are provided in Section 3).

Prepared using ettauth.cls [Version: 2012/06/19 v2.10]

In the IIoT context, security is paramount. As with all networking solutions, security happens at all layers of the protocol stack. At the link layer, all frames are secured: they can be authenticated and/or encrypted, using the mechanisms standardized in [8]. Securing and unsecuring a frame, as detailed in Section 3, are complex and potentially time-consuming operations. Moreover, different platforms have different hardware capabilities: on some, link-layer security can be accomplished using hardware support; on others, these operations need to be done in software because of the lack of hardware accelerators. Anyway, when security procedures are integrated in the TSCH communication schedule, a further computational overhead is introduced and the (minimum) timeslot duration is expected to increase as well, due to the additional time needed to secure/unsecure frames.

Starting from these premises, the present contribution wants to answer the following question: *How much shorter can a timeslot be?*

At the time of this writing, several contributions already investigated the effects of security operations in IoT systems on system capabilities [11]-[15]. Nevertheless, for the best of authors knowledge, they only focus on the legacy IEEE802.15.4-2011 [16] and no studies on IIoT networks based on TSCH are yet available (TSCH was not present in the legacy IEEE802.15.4-2011 standard, while it was introduced in the IEEE802.15.4e amendment, in 2012, and maintained in the latest version of the standard, that is IEEE802.15.4-2015 [17]).

To bridge this gap, this work aims at experimentally measuring the overhead of link-layer security in TSCH networks, on the minimal duration of a timeslot and on the memory footprint of the implementation. It is important to remark that we do not propose a new link-layer security mechanism. On the contrary, we would deeply investigate the impact that the one already proposed by the IEEE802.15.4e amendment, and maintained in IEEE802.15.4-2015, has in real IIoT systems. The scientific contribution of this articles is four-folded:

- We discuss the security functionalities in TSCH networks, the communication schedule described in the IEEE802.15.4e amendment (and maintained in IEEE802.15.4-2015), and the hardware support for link-layer security operations in today's IIoT platforms.
- We implement full link-layer security on older and state-of-the-art platforms, using different hardware/software strategies. The produced open-source code is released under the BSD license*.
- We conduct an exhaustive experimental study and measure that the minimal value of the timeslot duration is between 9 ms and 88 ms (for the most common security level, described in Section 3.2), across the different platforms and strategies.
- We describe the impact that the minimum timeslot has on both high-level application design and energy consumption.

The remainder of this article is organized as follows. Section 2 lists the most closely related works. Section 3 introduces the link-layer security mechanisms available in IEEE802.15.4. Section 4 discusses our different hardware and software link-layer security implementations, on both older and state-of-the-art platforms. Section 5 presents the experimental results, alongside further considerations on high-level applications design and energy consumption. Finally, Section 6 concludes this article.

2. RELATED WORKS

Although this article is not the first to evaluate the overhead of security in IEEE802.15.4 networks, it is, to the best of our knowledge, the first work that focuses on security in the IEEE802.15.4e TSCH amendment. This section lists the most closely related works.

The Advanced Encryption Standard (AES)-CBC-MAC Mode (CCM) scheme can be used at different layers of the protocol stack. For instance, in [18] it is used for application-layer security. [19], [20] and [21] further discuss its suitability for the TLS/DTLS protocols at the transport layer. In [22] and [23] it is used at the network layer of an IoT-compliant protocol stack. Finally, in [24, 25] it is used at the link layer.

In these last works, the authors show, by experimentation, that the hardware support provided by different platforms can be used to significantly speed up the execution of cryptographic primitives.

[13] studies the performance of the AES-CCM at the link layer, and compares it to other well-known cryptography schemes. The authors show that, even though AES is not the most efficient block cipher in terms of code-size and time performance, its well-known security properties, as well as the possibility to perform it in hardware, makes it the most suitable solution. The work presented in [26] evaluates the impact that cryptographic algorithms have on IoT resources (i.e. memory, processing, energy). It demonstrates that the Rijndael algorithm, which is the AES scheme, is one of efficient ciphers for IoT devices (as other block ciphers), offering a good compromise between code-size and cycle-count.

* http://telematics.poliba.it/openwsn_ieee802154_security

Several works discuss the advantages of hardware acceleration for security operations. The first contribution that highlighted the performance gain offered by hardware accelerators in CC2420-based platform (i.e. Crossbow MICAz and MoteIVs TmoteSKY) was [27]. In [28], the authors describe a compact and energy-efficient hardware implementation of IEEE802.15.4 security, and show its advantage in terms of execution speed and energy consumption. In [29], the authors design an energy-efficient hardware architecture for AES-CCM for IEEE802.15.4 networks. This work is completely integrable in the current article, as it provides an efficient way to perform AES-CCM. [30] evaluates hardware, software, and hybrid implementations of the AES encryption engine on IoT micro-controllers, along with a discussion about the tradeoffs between energy, throughput of the hardware module, memory footprint, and battery lifetime. Nevertheless, authors do not discuss the integration of AES cryptography algorithm with IEEE 802.15.4(e) security processing, as well as the impact derived by the adoption of such solutions on the overall network operation. All of these works are perfectly integrable with the current article, and can be effectively used to further improve global network performances.

Similar to the current article, [11, 12, 13, 14, 15] analyze the impact of IEEE802.15.4 security processing on network performance. [11] analyzes the impact of the inclusion of link-layer encryption and authentication services on energy and memory consumption, on Tmote-Sky motes. However, only the hardware implementation is presented, timing issues are not considered, nor is the integration with TSCH. [12] analyzes the impact of link-layer security processing on IEEE802.15.4 network performance (at the application layer) and on other protocol parameters. The authors focus on time and energy consumption. By simulations, they show that encryption and authentication affect the end-to-end application delay, but they do not include a discussion about the impact of link-layer security on the performance of the MAC protocol. [13] presents an implementation and performance evaluation of security functionalities at the link layer of IEEE802.15.4-compliant IoT devices. The authors present software and hardware implementations of security functionalities over the AVR XMEGA platform, evaluating memory and energy consumption produced by each one. Authors in [31] propose a novel mathematical tool for analyzing the IEEE 802.15.4 network performance, considering a fixed slot duration. The included model is perfectly integrable in our work, and can be used to derive different MAC delays when security is enabled. Authors in [14] evaluate the energetic cost of IEEE802.15.4 security in the context of energy harvesting devices and beacon-enabled mode. However, the impact of security processing over the TSCH mode of IEEE802.15.4e-2012 is not considered. An analytical evaluation of security overhead in the IEEE802.15.4 networks only is also presented in [15].

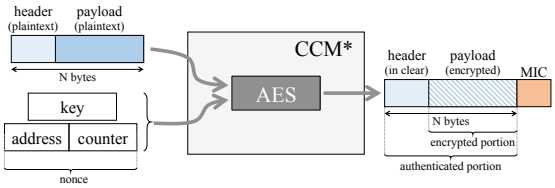


Figure 1. CCM* allows every frame to be encrypted and/or authenticated.

Several works focus on the new IEEE802.15.4e-2012 TSCH standard [9, 32, 33]. Yet, to the best of our knowledge, no work focuses on the performance of link-layer security.

3. LINK-LAYER SECURITY MECHANISMS IN IEEE802.15.4

Link-layer security has been part of IEEE802.15.4 since its first revision in 2003. This section provides the necessary background by describing the underlying mechanisms AES and CCM* (Section 3.1), and how these are used in IEEE802.15.4-2011 (Section 3.2), IEEE802.15.4e-2012 (Section 3.3), and IEEE802.15.4-2015 (Section 3.4).

3.1. AES-128 and CCM*

The Advanced Encryption Standard (AES) is a block cipher which encrypts a sequence of plaintext bytes in blocks of 16 bytes, using a 128-bit key in its AES-128 variant. It uses a series of permutations and substitutions, and therefore executes fast when implemented in both hardware and software.

CCM* is a "wrapper" cryptographic primitive around AES-128 that uses CTR mode for encryption and CBC-MAC for authentication. It encrypts and/or authenticates an arbitrarily long sequence of plaintext bytes (note that a theoretic upper bound on the message length for CCM scheme exists, but for practical purposes it can be considered arbitrarily long for IIoT technology) . When applied to a link-layer frame, this means that CCM* can encrypt the Medium Access Control (MAC) payload while keeping the MAC header intact. When used to authenticate a frame, CCM* calculates a Message Integrity Code (MIC) over the complete frame. This MIC is truncated to the desired length (4, 8 or 16 bytes), and appended at the end of the frame.

Each frame secured with a given key must use a different nonce. Encrypting two frames with the same nonce has severe consequences on security: plaintext of both frames may be easily recovered. By constructing the nonce with a monotonic counter, it is possible to ensure replay protection for two communicating nodes. We depict this in Fig. 1.

3.2. Link-layer Security in IEEE802.15.4-2011

Table I depicts the security levels of IEEE802.15.4. Levels differ in MIC length and whether encryption is applied on the payload or not. A higher security level induces a larger frame due to the longer MIC, but computational overhead stays the same. Each MAC frame can use a different security level. Local policies dictate if the security level of the received frame should be accepted or not. These conformance checks are a pre-requisite for the CCM* verification to be invoked.

Each secured frame carries an Auxiliary Security Header (ASH) with signaling information related to the key and nonce. In IEEE802.15.4-2011, the nonce is created from the address of the sender and the local frame counter that increments for each transmitted frame. The 4-byte frame counter must be signaled to the recipient, and is therefore included in the ASH. Because the recipient keeps track of the last frame counter it received from a given neighbor, frames are protected against replay attacks. Key signaling overhead varies with the key management scheme used, and can range from 0 bytes for implicit keying to 9 bytes. The total ASH overhead (not including the MIC) ranges from 5 to 14 bytes.

The upper layer sets the specific security level and the key to be used on beacon, command or data frames. However, Acknowledgment (ACK) frames in IEEE802.15.4-2011 do not support security and are always sent in clear.

3.3. Changes introduced with link-layer security in IEEE802.15.4e-2012

IEEE802.15.4e introduces Information Elements (IE) to exchange information between neighbor nodes in a TSCH network. Nodes maintain synchronization by indicating the time correction as part of an IE in ACK frames. An attacker could perform Denial of Service (DoS) attacks by altering this time correction; for this reason IEEE802.15.4e-2012 added support for secured ACK frames.

Fig. 2 depicts the operations that happen in a timeslot when `A` sends a data frame to `B`:

- `A` secures the data frame. We call this operation sec_1.
- At precisely `TsTxOffset` into the timeslot, `A` sends this (secured) frame to `B`. The transmission of the data frame takes at most `TsMaxTx`.
- `B` unsecures the data frame, which can involve decrypting and/or authenticating it (operation sec_2).
- If the unsecuring operation is successful, `B` secures an ACK frame (operation sec_3).
- Exactly `TsTxAckDelay` after receiving the end of the data frame, `B` sends the (secured) ACK frame. The transmission of the ACK frame takes at most `TsMaxAck`.
- `A` unsecures the ACK frame (operation sec_4). If successful, it removes the data frame from its transmission queue.

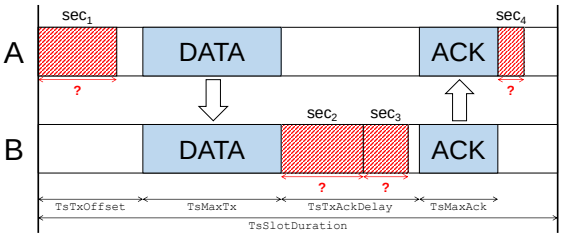


Figure 2. The minimal duration of a timeslot in an IEEE802.15.4e TSCH network depends on how long link-layer security operations take.

Eq. (1) indicates the timing constraints that the duration of those security operations put on the different TSCH timings. We denote $dur(sec_1)$ the duration of sec_1.

$$\begin{cases} \texttt{TsTxOffset} \geq & dur(sec_1) \\ \texttt{TsTxAckDelay} \geq & dur(sec_2) + dur(sec_3) \\ \texttt{TsSlotDuration} \geq & \texttt{TsTxOffset} + \texttt{TsMaxTx} + \\ & \texttt{TsTxAckDelay} + \texttt{TsMaxAck} + \\ & dur(sec_4) \end{cases} \tag{1}$$

Time synchronization in the network means that all nodes share the Absolute Slot Number (ASN): the number of slots which have passed since the network has started. The ASN is forever incrementing[†]. A common notion of time simplifies replay protection as a node does not need to maintain a frame counter for each of its neighbors. Instead, TSCH uses the ASN as a frame counter and omits its inclusion in the ASH, reducing the overhead by 4 bytes.

The use of ASN in the nonce implies that the sec_1 operation can only be done once the ASN of the slot is known (see Fig. 2). In practice, this means that pre-calculating the sec_1 operation is not possible. Operation sec_1 includes the key lookup and

[†]The ASN is encoded on 5 bytes. With a 10 ms timeslot duration, the ASN value rolls over every 350 years.

Table I. The security levels in IEEE802.15.4.

mode	encrypted payload?	MIC length
`MIC-32`	NO	4 bytes
`MIC-64`	NO	8 bytes
`MIC-128`	NO	16 bytes
`ENC`	YES	*no MIC*
`ENC-MIC-32`	YES	4 bytes
`ENC-MIC-64`	YES	8 bytes
`ENC-MIC-128`	YES	16 bytes

CCM* encryption on a potentially maximum length frame (127 bytes). Before the receiving node can transmit an ACK, it must verify the conformance of the frame against local security policies and decrypt/authenticate it (sec_2). Finally, the node prepares and secures the ACK frame (sec_3), which includes the time correction indication. Before the time correction can be applied on the transmission side (node `A` in Fig. 2), the ACK frame must pass the CCM* check and conformance verifications (sec_4). The duration of the sec_1, sec_2, sec_3 and sec_4 operations on different hardware platforms directly influences the minimum slot duration, which we evaluate experimentally in Section 5.

3.4. Further amendments introduced with IEEE802.15.4-2015

The security sub-layer standardized in IEEE802.15.4-2015 [17] generally integrates security services defined in both IEEE802.15.4-2011 and IEEE802.15.4e-2012. However, some few amendments have been introduced.

First of all, the ENC security level is not supported anymore because it is not robust against trivial Man-In-The-Middle attacks. Secured and unsecured communications can coexist in the same network (for instance, in the case a device has not enough resources to manage security, it can use a *Capability IE* to inform its neighbors about this issue, thus asking them to switch to unsecured communications). Moreover, IEs and specific frames introduced by previous specifications (such as command frames defined in IEEE802.15.4e-2012) can be protected.

A different management of the frame counter is defined for the two possible network operation modes. In the legacy mode, it is an integer value associated to the device and to the key used to protect the communication. In the TSCH mode, instead, the frame counter is elided from the packet.

Few updates of both ongoing and incoming security functionalities have been added in order to take care of aforementioned amendments. Nevertheless, these changes do not modify the computational overhead introduced by security functionalities defined in IEEE802.15.4e-2012, which is instead mainly due to AES and CCM* operations.

4. LINK-LAYER SECURITY IMPLEMENTATIONS

This section describes the strategies we pursued to implement link-layer security and CCM* in IEEE802.15.4e-2012 (and IEEE802.15.4-2015) TSCH networks. In Section 4.1, we introduce the hardware platforms used for our evaluations that are also supported in the OpenWSN stack. Three different implementations strategies are presented: software (Section 4.2), hardware (Section 4.3) and hybrid (Section 4.4). They differ in whether hardware support for AES and/or CCM* is available and exploited (see Table II).

All source code developed is available open-source, under a BSD license.

4.1. Platforms Used

Our goal is to quantify the overhead of link-layer security on a range of hardware platforms, and using different software strategies. Therefore, we choose to implement full link-layer security on platforms separated by a decade of hardware development: TelosB (in many ways obsolete but still very popular) and the OpenMote-CC2538 (a state-of-the-art platform).

TelosB. The TelosB mote [34] (designed in 2004) features an MSP430 microcontroller (16-bit architecture, 8 MHz maximum CPU speed, 48 kB flash, 10 kB RAM), and a CC2420 radio chip. The CC2420 radio [35] chip provides hardware acceleration for both AES and CCM*.

To execute an AES operation in hardware on the CC2420, one loads 16 bytes of cleartext into a buffer and a 16-byte key in a dedicated register, and issues a `SAES` command. This "atomic" AES operation takes $14\mu s$, a number which does *not* account for the time to load the buffer and key from the micro-controller into the radio chip. Also, one has to execute this operation for each 16-byte block of a long frame.

To execute a CCM* operation in hardware on the CC2420, one sets the `Security Control 0` and `Security Control 1` registers to configure the security mode, the length of the authentication tag, the length of data to be authenticated but not encrypted, and the exact position where authenticated data starts. The nonce has to be loaded in the `TXNONCE` register and the frame into the transmission queue. Issuing a `STXENC` executes a CCM operation in the buffer; issuing `STXON` additionally transmits the frame. Considering a message of 119 bytes, a CCM operation takes $222\mu s$.

The TelosB is a two chip-solution: the micro-controller and radio chips are connected by an SPI digital interface. Given the speed of this bus, it takes approximately 1 ms to transfer a full-sized frame between the chips. This adds a extra delay to the minimal timeslot duration.

OpenMote-CC2538. The OpenMote-CC2538 mote [36] (designed in 2014) is a state-of-the-art platform at the time to writing, which features a single-chip CC2538 solution. This chip contains both an ARM Cortex-M3 micro-controller (32-bit architecture, 32 MHz maximum CPU speed, 512 kB flash, 32 kB RAM) and a IEEE802.15.4-compliant radio. One immediate advantage of such a single-chip solution is that the micro-controller and radio chip share the same RAM memory; no time needs to be spent transferring a packet to/from the radio chip.

The CC2538 uses a single `AES Control` register to configure/trigger AES and CCM* operations in hardware. The AES module cuts the input bytes into 16-byte blocks and triggers the AES operation autonomously. When the operation finishes, the master controller generates a `data_done` interrupt so the software can retrieve the encrypted data. The software sets the nonce, key and other CCM* configurations in the `AES Control` register.

A Direct Memory Access (DMA) modules takes care of feeding the AES and CCM* modules, without intervention from the software. This significantly speeds up the operation when compared to the CC2420. Encrypted/authenticated data is available directly in RAM memory, no (time consuming) SPI transfers are needed. Finally, the external 32 MHz crystal oscillator is used to clock the AES and CCM* hardware modules. This is possible because the AES cryptoprocessor can be clocked independently from the processor.

The AES algorithm executes at 18 Mbps[‡]. CCM executes at 12 Mbps, including the final operation of create the MIC. Thus, considering a single block of 128 bits, security operations are executed in $7\mu s$ for the AES algorithm and $85\mu s$ for the CCM scheme. The CC2538 completes AES and CCM tasks faster than the CC2420.

Table II. Implementation Strategies.

strategy		AES	CCM*
"software"	(Sec. 4.2)	software	software
"hybrid"	(Sec. 4.4)	hardware	software
"hardware"	(Sec. 4.3)	hardware	hardware

OpenWSN. We use the OpenWSN implementation on both platforms. OpenWSN is an open-source implementation of a standards-based protocol stack for the IoT. Its protocol stack includes IEEE802.15.4e-2012 TSCH, 6LoWPAN, RPL and CoAP [37].

We augment OpenWSN with link-layer security, implemented using three strategies: in software (Section 4.2), in hardware (Section 4.3), and a hybrid solution (Section 4.4). As shown in Table II, these strategies differ in whether they exploit hardware acceleration for AES and/or CCM*. The three strategies are detailed in the next sections.

4.2. Software Implementation

The "software" implementation strategy consists in implementing both AES and CCM* in software. We use a software AES implementation from Texas Instruments[§], unmodified. This implementation is optimized for embedded devices, and features an execution time approximately 10 times shorter than a baseline AES implementation such as the one presented in [38]. We implement CCM* from scratch. As indicated above, all software is available open-source.

In the software implementation, we purposely do *not* use the hardware acceleration for AES/CCM* offered by both platforms in order to be able to (1) quantify how much shorter the timeslot can be when using hardware acceleration, and (2) have an idea of the performance of a platform which does not have hardware acceleration.

4.3. Hardware Implementation

The "hardware" implementation strategy consists in exploiting hardware acceleration for both AES and CCM*. Both platforms offer AES and CCM* hardware support, and we use the techniques described in Section 4.1 in the implementation.

Even though the hardware executes the core of the security operation, software instructions are necessary to store security-related parameters in the

[‡] http://www.ti.com.cn/cn/lit/ug/swru319c/swru319c.pdf

[§] http://www.ti.com/tool/AES-128

correct location, disable or enable hardware interrupts, give the "go" signal for encryption/decryption operations, and fetch the output of the operation.

4.4. Hybrid Implementation

Many hardware platforms offer AES hardware acceleration, but not CCM*. To measure the overhead of link-layer security on those platforms, we adopt a "hybrid" implementation strategy in which we rely on the hardware for AES, but implement CCM* is software.

The hybrid implementation hence uses hardware-accelerated AES block cipher. The rest of the CCM* algorithm, which includes CTR and CBC-MAC modes of operation, creation of plaintext and ciphertext is handled through software instructions.

5. EXPERIMENTAL RESULTS

This section presents the experimental results obtained on the implementations presented in Section 4. Section 5.1 details the experimental setup we use and the methodology we adopt to tune the different durations in the timeslot. Section 5.2 indicates the memory footprint of the implementations, for both flash and RAM memory. Section 5.3 presents the different durations measured on both platforms, across the three implementation strategies, and for all eight security levels. We also discuss the implication for IEEE802.15.4e TSCH networks. Section 5.4 discusses the energy consumption related to the three implementation strategies, and for all eight security levels. Finally, Section 5.5 provides further comments on the impact of timeslot duration on high-level application design.

5.1. Goals and Methodology

The goal of this experimental evaluation is to implement the three implementation strategies on both platforms, tune the timeslot duration to the minimal value obtainable given the duration of the link-layer security operations, and measure the different timings. Specifically, we want to answer the following question: *How much shorter can a timeslot be if hardware acceleration is used?* We also measure and report the memory footprint (the amount of RAM and flash memory required) and the energy consumption of the different implementation strategies.

The experimental setup consists of two nodes forming one network. One of the nodes is the root of the network, the other is a leaf node that attaches to the root. In each case, after loading the appropriate software on the nodes, we boot the root node and wait for the leaf node to synchronize to it. The root node is attached to a computer; from that computer we use the `ping` program to verify connectivity to the leaf node. `ping` allows us to choose the size of the payload sent in the ICMPv6 echo request/response packets; we choose it so that the resulting link-layer frame is always 127 bytes long (the maximum length for IEEE802.15.4-compliant nodes).

In OpenWSN, IEEE802.15.4e TSCH is implemented as a finite state machine. Different timings, illustrated in Fig. 2, cause the state machine to advance. `TsTxOffset` is an example timing: when it expires, the state machine kicks off the transmission of the frame. This means, at that point in the timeslot, all the operations for preparing the packet (including sec_1) need to be complete. The OpenWSN implementation uses a 32-kHz counter to measure the duration of different phases of the IEEE 802.15.4e finite state machine. This allows us to measure the duration of the sec_1, sec_2, sec_3 and sec_4 operations, and verify the `TsTxOffset`, `TsTxAckDelay` and `TsSlotDuration` durations.

For each evaluated case, we "tune" the different durations in order to obtain the shortest possible timeslot. We start by setting `TsTxOffset`, `TsTxAckDelay` and `TsSlotDuration` to very high values, then reduce them. In all cases, these values must satisfy the constraints in Eq. (1). The different steps of the tuning procedure are illustrated in Table III. We first measure the duration of sec_1, and reduce the value of `TsTxOffset` accordingly. According to the standard [8], `TsMaxTx` and `TsMaxAck` are set to 4.256 and 2.400 ms respectively. We then measure the durations of sec_2 and sec_3, and set the value of `TsTxAckDelay` accordingly. Similarly, we measure the duration of sec_4 and tune `TsSlotDuration`.

We repeat this tuning operation for both platforms, using the three implementation approaches, and for each of the 8 security levels. There are hence 48 cases, discussed in the following sections.

Note that sec_1, sec_2, sec_3, and sec_4 have been measured by using the 32KHz counter available in the OpenWSN protocol stack. Now, considering that the variability of the amount of time needed to execute security operations is very lower than the resolution of this counter, obtained results (reported in the following subsections) appear as deterministic.

Finally, results related to the energy consumption are generated by considering the amount of energy consumed by TelosB and OpenMote-CC2538 platforms when they perform different operations in a timeslot (reference values are taken from their datasheets).

Table III. The different steps in the tuning procedure.

$dur(sec_1)$	measure
`TsTxOffset`	tune to $dur(sec_1)$
`TsMaxTx`	set to 4.256 ms
$dur(sec_2)$	measure
$dur(sec_3)$	measure
`TsTxAckDelay`	tune to $dur(sec_2)$+$dur(sec_3)$
`TsMaxAck`	set to 2.400 ms
$dur(sec_4)$	measure
`TsSlotDuration`	tune to `TsTxOffset` + `TsMaxTx` + `TsTxAckDelay` + `TsMaxAck` + $dur(sec_4)$

5.2. Memory Footprint

The memory footprint is the amount of flash [¶] and RAM memory that needs to be reserved for link-layer security. Fig. 3 and 4 summarize the results. The security level does not impact the memory footprint, and is hence not presented. Since the TelosB and OpenMote-CC2538 have both different processor architectures and compilers, and since different toolchains are used to build their software, comparing the absolute value of the memory footprint is not appropriate. Focusing on a single platform, instead, the analysis of both ROM and RAM footprints is useful to compare different link-layer security implementations.

On the TelosB mote, link-layer security occupies between 7% and 15% of the available flash memory, and 30-40% of RAM memory. On the OpenMote-CC2538, it occupies 1.5-2.5% of flash and 8% of RAM. As can be expected, the footprint of the software (resp. hardware) implementation is the largest (resp. lowest).

In general, it is possible to observe that using hardware acceleration for link-layer security does not result in a drastically lower memory footprint. This is because (i) a software implementation of AES and CCM is relatively simple and (ii) even when hardware acceleration is used, software is still needed to drive the hardware.

On the TelosB mote, the number of instructions needed to control AES and CCM algorithms in hardware is the same. Therefore, the hardware implementation registers a reduction of the ROM footprint with respect to the hybrid implementation because part of the software related to CCM operations (e.g., instructions for CCM* vectors handling) is deleted.

On the OpenMote-CC2538, instead, the driver needed to control CCM operations in hardware has a ROM footprint that is quite similar to the amount of software used in the hybrid implementation. Therefore, this platform seems to report the same ROM footprint when both hybrid and hardware implementations are used.

[¶] We use flash and ROM (Read-Only Memory) interchangeably.

5.3. Minimal Slot Length

Figs. 5 and 6 present the resulting `TsTxOffset`, `TsTxAckDelay` and `TsSlotDuration` durations after tuning procedure, for the TelosB and OpenMote-CC2538 platforms, respectively.

When the "software" implementation is used, TelosB and OpenMote-CC2538 executes security operations at different speeds of the CPU. For instance, TelosB runs at 8 MHz; OpenMote-CC2538 can run at a maximum speed of 32MHz. As a result, the software-based implementation of link-layer security in OpenMote-CC2538 will result in a timeslot duration reduced by a factor of $2-3\times$.

The hardware implementation strategy results in a shorter slotframe than the software implementation. Depending on the security level, a hardware-based implementation of link-layer security will result in a timeslot duration reduced by $3-4\times$.

The difference in timeslot duration between "software" and "hybrid" implementation strategies reflects the advantage of having AES execute in hardware. Similarly, the comparison between "hybrid" and "hardware" implementations reflects the advantage of a hardware-based CCM*.

On slower platforms such as the TelosB (Fig. 5), the biggest gains are made by running AES in hardware.

The most common security level in TSCH networks (including WirelessHART and 6TiSCH) is `ENC-MIC-32`, i.e. frames are encrypted and a 4-byte MIC is used for authentication. We highlight that security level in Figs. 5 and 6. A full software implementation on an older platform such as the TelosB results in a minimal timeslot duration of 88 ms. Using hardware acceleration reduces the time duration by a factor of 4, down to 25 ms. Switching to a state-of-the-art platforms, which features both faster hardware implementation of AES and CCM*, and a single-chip architecture, allows the timeslot

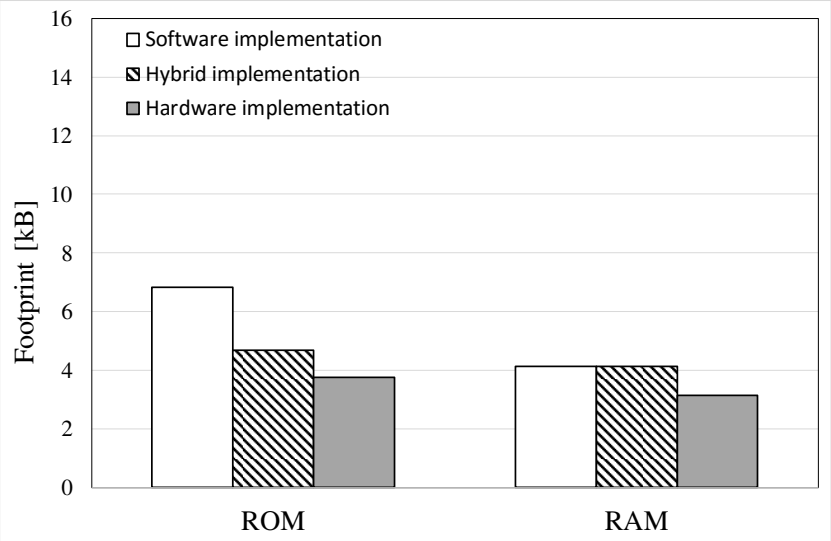


Figure 3. ROM and RAM Footprint of the implementations for TelosB mote.

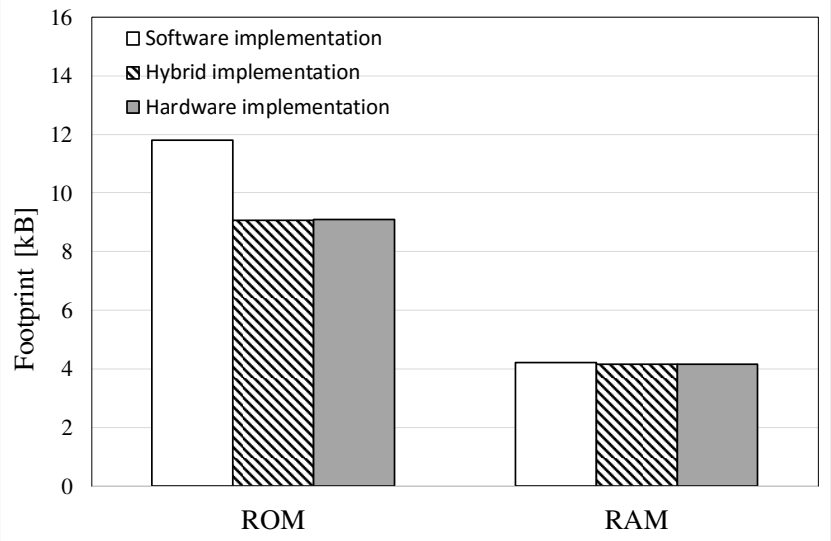


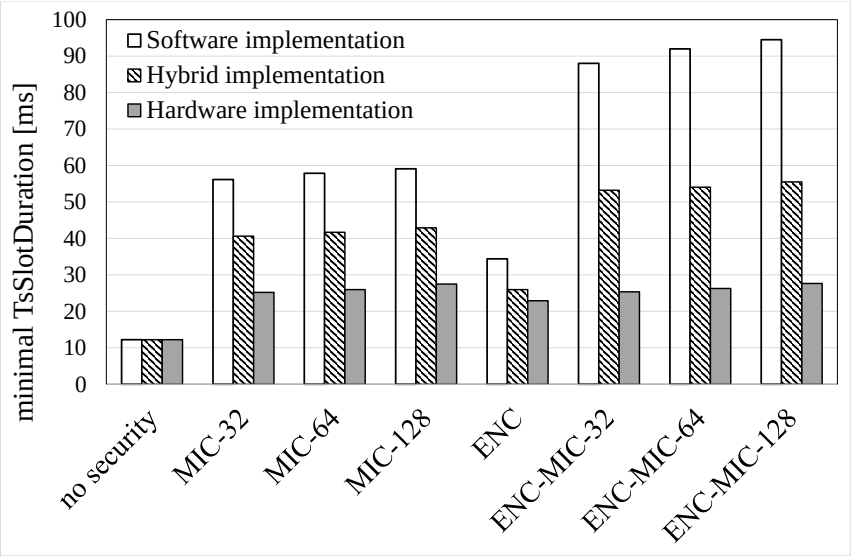
Figure 4. ROM and RAM Footprint of the implementations for OpenMote-CC2538 mote.

duration to be further reduce by a factor of more than 2, down to 9 ms.

By using hardware accelerations in OpenMote-CC2538, it is possible to satisfy the target value of the timeslot duration reported by the standard, i.e., 10 ms [8]. In the other cases, this goal is not reached. Anyway, it is important to note that conducted tests considered the worst case, where the maximum size of the ASH is used (e.g. equal to 14 bytes). Now, considering that the higher the size of the ASH, the higher the computational overhead introduced by additional (security) lookup procedures,, all the results reported in Section 5 refer to the most complex network configuration requiring higher timeslot durations. Other configurations, instead, may require lower timeslot durations.

5.4. Energy consumption

Figs. 7 and 8 reports the amount of energy consumed by TelosB and OpenMote-CC2538, respectively, for the three implementation strategies, and for all eight security levels. These results have been estimated (not experimentally measured) by taking into account the energy consumption related to each phase of the timeslot (TX, RX and CPU idle), as reported by reference datasheets.



Security Level	Software Implementation			Hybrid Implementation			Hardware Implementation		
	TsTx Offset [ms]	TsTx AckDelay [ms]	*minimal* TsSlot Duration [ms]	TsTx Offset [ms]	TsTx AckDelay [ms]	*minimal* TsSlot Duration [ms]	TsTx Offset [ms]	TsTx AckDelay [ms]	*minimal* TsSlot Duration [ms]
no security	3.63	4.46	12.24	3.63	4.46	12.24	3.63	4.46	12.24
MIC-32	15.71	35.61	56.15	9.90	27.59	40.62	7.75	11.87	25.21
MIC-64	16.32	35.89	57.86	10.11	27.96	41.69	7.81	12.18	25.97
MIC-128	16.72	36.5	59.09	10.42	28.66	42.91	7.90	12.79	27.50
ENC	9.93	20.01	34.38	8.60	11.56	25.96	7.66	10.01	22.92
ENC-MIC-32	26.48	44.22	88.02	14.92	33.24	53.22	7.78	12.48	25.36
ENC-MIC-64	28.25	44.52	91.98	14.98	33.61	54.05	7.83	12.54	26.28
ENC-MIC-128	30.53	45.04	94.49	15.29	34.15	55.51	7.97	12.94	27.65

Figure 5. Minimum timeslot duration for the TelosB mote.

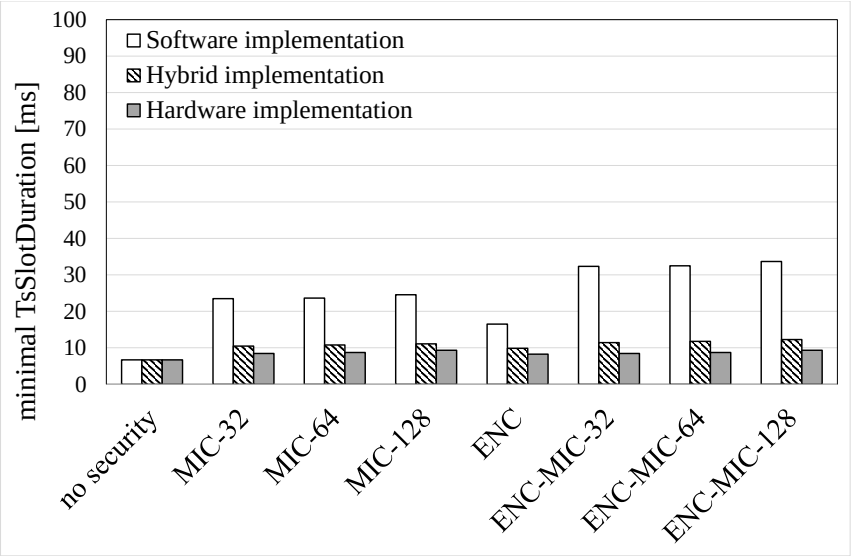
Link-layer security brings to an increment of the per-slot energy consumption because: (i) additional operations are executed for protecting the transmitted packet and decrypting the corresponding ACK and (ii) the radio chip is in charge of transmitting more data due to the ASH and the MIC.

On the TelosB mote, hardware accelerators reduce energy consumption up to 27% (for the hybrid implementation) and 47% (for the hardware implementation). On the OpenMote-CC2538, instead, hardware accelerators leads to a gain of up to 58 % (for the hybrid implementation) and 66% (for the hardware implementation). Moreover, OpenMote-CC2538 always emerges as the more energy-consuming platform.

5.5. Final considerations on high-level application design

To conclude, based on the results described in the previous Section, additional comments on high-level application design and energy consumption can be formulated.

As we have already anticipated in the Introduction, the shorter the timeslot, the higher the network throughput. Therefore, the provisioning of link-layer security inevitably brings to a throughput reduction. In addition, the message overhead produced by



Security Level	Software Implementation			Hybrid Implementation			Hardware Implementation		
	TsTx Offset [ms]	TsTx AckDelay [ms]	*minimal* TsSlot Duration [ms]	TsTx Offset [ms]	TsTx AckDelay [ms]	*minimal* TsSlot Duration [ms]	TsTx Offset [ms]	TsTx AckDelay [ms]	*minimal* TsSlot Duration [ms]
no security	1.15	0.67	6.71	1.15	0.67	6.71	1.15	0.67	6.71
MIC-32	2.48	10.07	23.50	1.98	2.24	10.47	1.28	1.12	8.45
MIC-64	2.51	10.19	23.65	2.14	2.36	10.77	1.31	1.19	8.72
MIC-128	2.54	10.31	24.57	2.29	2.45	11.08	1.34	1.25	9.34
ENC	2.31	3.20	16.51	1.77	1.51	9.86	1.28	0.98	8.24
ENC-MIC-32	6.44	14.80	32.35	2.38	2.82	11.44	1.28	1.12	8.45
ENC-MIC-64	6.57	14.86	32.50	2.41	2.94	11.78	1.31	1.19	8.72
ENC-MIC-128	6.60	15.26	33.66	2.44	3.24	12.27	1.34	1.25	9.34

Figure 6. Minimum timeslot duration for the OpenMote-CC2538 mote.

each security level (i.e., Auxiliary Security Header plus MIC, if present) differently impacts on the effective amount of bytes that can be inserted, in each packet, by the application layer. Practically, the data generation rate, R, that a high-level application running on the i-th device could use should satisfy the following equation:

$$R \leq \frac{N_i \ A_j}{N_{ts} \ \texttt{TsSlotDuration}}, \qquad (2)$$

where N_i, A_j, and N_{ts} are the number of timeslots assigned to the i-th device for transmitting data, the maximum size of an application message due to the j-th security level[‖], and the total number of timeslots available in a TSCH slot frame (i.e., a predefined number of timeslots that repeats over the time). For simplicity, Eq. (2) has been obtained by assuming the absence of packets fragmentation at the application layer. Maximum performance are reached if $R = (N_i \ A_j)/(N_{ts} \ \texttt{TsSlotDuration})$ and no packet losses are registered on the wireless communication channel. Higher values of R are not

[‖] For instance, $A_{no\ security} = 568$ bits, $A_{MIC-32} = 520$ bits, $A_{MIC-64} = 488$ bits, $A_{MIC-128}$ 424 = bits, $A_{ENC} = 424$ bits, $A_{ENC-MIC-32} = 520$ bits, $A_{ENC-MIC-64} = 488$ bits, and $A_{ENC-MIC-128} = 424$ bits

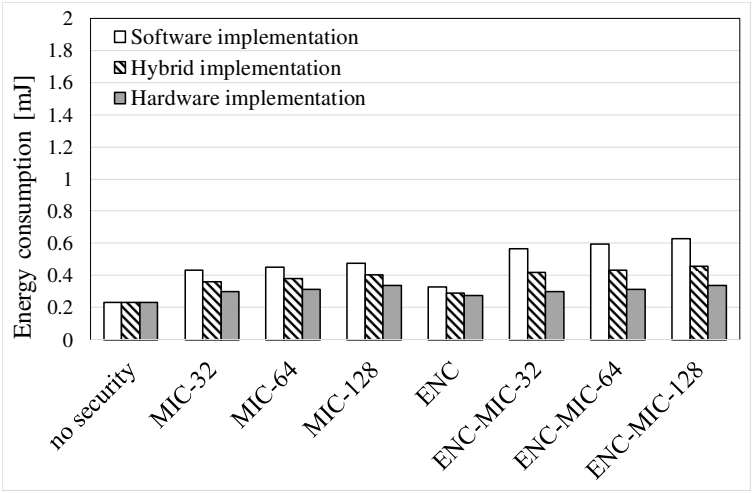


Figure 7. Energy consumed in a single timeslot for the TelosB mote.

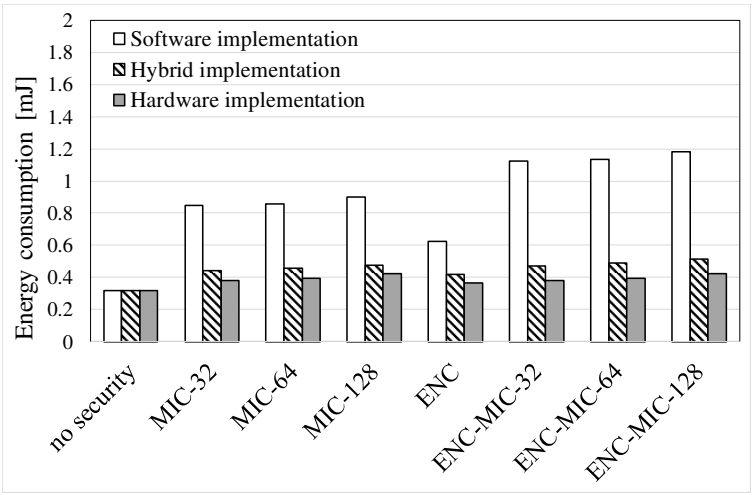


Figure 8. Energy consumed in a single timeslot for the OpenMote-CC2538 mote.

supported because they produce an uncontrolled increment of packet queue, thus bringing to a growth of packet losses (e.g., packets are lost at the transmitter side because there is not enough space in the queue).

6. CONCLUSION

This works evaluates the overhead of using link-layer security in IEEE802.15.4e Time Synchronized Channel Hopping (TSCH) networks. The challenge is that four link-layer security operations need to happen in a single timeslot: securing/unsecuring the data frame, and securing/unsecuring the acknowledgment. The duration of the link-layer securing/unsecuring operations hence directly impacts on the minimum duration of the timeslot. To increase the overall throughput of the network, and decrease the end-to-end latency, this duration should be as low as possible.

We conduct a thorough experimental study, and measure the overhead of link-layer security across different generations of hardware, different hardware/ software implementation strategies, and different security levels in the IEEE 802.15.4 protocol. We use two popular off-the-shelf platforms, separated by a decade of hardware development: the older but still popular TelosB and the state-of-the-art OpenMote-CC2538. All implementations are done on the OpenWSN protocol stack, and available open-source under a BSD license.

We show the benefits of hardware acceleration for two "atomic" link-layer security mechanisms:

AES – a block cipher – and CCM* – a "wrapper" cryptographic primitive around AES. Overall, using hardware acceleration offers a $3-4\times$ reduction in timeslot duration. Using the latest generation hardware and full hardware acceleration allows a timeslot duration reduction of roughly $10\times$, from 88 ms on older hardware using a fully software implementation, down to 9 ms.

ACKNOWLEDGMENTS

This work is supported by the PON project RES NOVAE funded by the Italian MIUR and by the European Union (European Social Fund), by the project BONVOYAGE, which receives funding from the European Union's Horizon 2020 research and innovation programme under grant agreement 635867, and by the project symbIoTe, which receives funding from the European Union's Horizon 2020 research and innovation programme under grant agreement 688156. The authors would like to acknowledge Luigi Alfredo Grieco, Kris Pister, Bernard Tourancheau, and all the people actively working in the IETF 6TiSCH working group for their guidance and input.

REFERENCES

1. O. Hersent, D. Boswarthick, and O. Elloumi, *The Internet of Things: Key Applications and Protocols*, 2nd ed. Wiley, 2012.
2. D. Evans, "The Internet of Things, How the Next Evolution of the Internet Is Changing Everything," Cisco Internet Business Solutions Group (IBSG). White paper, Apr. 2011.
3. Ericsson, "More than 50 billion connected devices," Ericsson White Paper, Feb. 2011.
4. B. Emmerson, "M2M: the Internet of 50 Billion Devices," *M2M Magazine*, vol. 2010, no. 1, pp. 19–22, January 2010.
5. L. D. Xu, W. He, and S. Li, "Internet of Things in Industries: A Survey," *IEEE Trans. on Ind. Inf.*, vol. 10, no. 4, pp. 2233–2243, Nov. 2014.
6. K. S. J. Pister and L. Doherty, "TSMP: Time Synchronized Mesh Protocol," in *Int. Symp. on Distributed Sensor Networks (DSN)*. Orlando, FL, USA: IASTED, 16-18 November 2008, pp. 391–398.
7. *WirelessHART Specification 75: TDMA Data-Link Layer*, HART Communication Foundation Std., Rev. 1.1, 2008, hCF_SPEC-75.
8. *802.15.4e-2012: IEEE Standard for Local and metropolitan area networks–Part 15.4: Low-Rate Wireless Personal Area Networks (LR-WPANs) Amendment 1: MAC sublayer*, IEEE Std., 16 April 2012.
9. T. Watteyne, L. Doherty, J. Simon, and K. Pister, "Technical Overview of SmartMesh IP," in *Int. Worksh. on Ext. Seamlessly to the Internet of Things (esIoT)*, Taiwan, 3-5 July 2013.
10. D. Dujovne, T. Watteyne, X. Vilajosana, and P. Thubert, "6TiSCH: Deterministic IP-enabled Industrial Internet (of Things)," *IEEE Comm. Magazine*, vol. 52, no. 12, pp. 36–41, Dec. 2014.
11. R. Daidone, G. Dini, and M. Tiloca, "On Experimentally Evaluating the Impact of Security on IEEE 802.15.4 networks," in *Int. Conf. on Distr. Comput. in Sensor Systems and Workshops (DCOSS)*, Jun. 2011, pp. 1–6.
12. F. Chen, X. Yin, R. German, and F. Dressler, "Performance Impact of and Protocol Interdependencies of IEEE 802.15.4 Security Mechanisms," in *Int. Conf. on Mobile Adhoc and Sensor Syst. (MASS)*. IEEE, Oct. 2009.
13. D. Altolini, V. Lakkundi, N. Bui, C. Tapparello, and M. Rossi, "Low Power Link Layer Security for IoT: Implementation and Performance Analysis," in *Int. Wirel. Commun. and Mobile Comput. Conf. (IWCMC)*, Jul. 2013, pp. 919–925.
14. M. Vucinic, B. Tourancheau, F. Rousseau, A. Duda, L. Damon, and R. Guizzetti, "Energy cost of security in an energy-harvested IEEE 802.15.4 Wireless Sensor Network," in *Med. Conf. on Embedded Comp. (MECO)*, Jun. 2014, pp. 198–201.
15. Y. Xiao, H.-H. Chen, B. Sun, R. Wang, and S. Sethi, "MAC Security and Security Overhead Analysis in the IEEE 802.15.4 Wireless Sensor Networks," *Eurasip Journal on Wirel. Comm. and Netw.*, pp. 1–12, May 2006.
16. *802.15.4-2011: IEEE Standard for Local and metropolitan area networks–Part 15.4: Low-Rate Wireless Personal Area Networks (LR-WPANs)*, IEEE Std., 16 June 2011.
17. "IEEE Standard for Low-Rate Wireless Personal Area Networks (WPANs)," *IEEE Std 802.15.4-2015 (Revision of IEEE Std 802.15.4-2011)*, pp. 1–709, Apr. 2016.
18. M. Vucinic, B. Tourancheau, F. Rousseau, A. Duda, L. Damon, and R. Guizzetti, "OSCAR: Object Security Architecture for the Internet of Things," in *Int. Symp. on a World of Wirel., Mobile and Multim. Netw. (WoWMoM)*. IEEE, Jun. 2014, pp. 1–10.
19. S. L. Keoh, S. Kumar, and H. Tschofenig, "Securing the Internet of Things: A Standardization Perspective," *IEEE Internet of Things Journal*, vol. 1, no. 3, pp. 265–275, Jun. 2014.
20. *AES-CCM Cipher Suites for Transport Layer Security (TLS)*, Internet Engineering Task

Force Std. RFC6655, 2012.
21. *AES-CCM Elliptic Curve Cryptography (ECC) Cipher Suites for TLS*, Internet Engineering Task Force Std. RFC7251, 2014.
22. K. Rantos, A. Papanikolaou, and C. Manifavas, "IPsec over IEEE 802.15.4 for Low Power and Lossy Networks," in *Int. Symp. on Mobility Manag. and Wirel. Acc. (MobiWac)*. New York, NY, USA: ACM, 2013, pp. 59–64.
23. *Using Advanced Encryption Standard (AES) CCM Mode with IPsec Encapsulating Security Payload (ESP)*, Internet Engineering Task Force Std. RFC4309, 2005.
24. S. Raza, S. Duquennoy, J. Höglund, U. Roedig, and T. Voigt, "Secure Communication for the Internet of Things - A Comparison of Link-Layer Security and IPsec for 6LoWPAN," *Security and Commun. Netw., Wiley*, vol. 7, no. 12, pp. 2654–2668, Dec. 2014.
25. S. Raza, S. Duquennoy, T. Chung, D. Yazar, T. Voigt, and U. Roedig, "Securing Communication in 6LoWPAN with Compressed IPsec," in *Distributed Computing in Sensor Systems and Workshops (DCOSS)*, June 2011, pp. 1–8.
26. Y. W. Law, J. Doumen, and P. Hartel, "Survey and Benchmark of Block Ciphers for Wireless Sensor Networks," *ACM Trans. Sen. Netw.*, vol. 2, no. 1, pp. 65–93, Feb. 2006.
27. M. Healy, T. Newe, and E. Lewis, *Smart Sensors and Sensing Technology*. Springer Berlin Heidelberg, 2008, ch. Analysis of Hardware Encryption Versus Software Encryption on Wireless Sensor Network Motes, pp. 3–14.
28. P. Hamalainen, M. Hannikainen, and T. Hamalainen, "Efficient Hardware Implementation of Security Processing for IEEE 802.15.4 Wireless Networks," in *Midwest Symposium on Circuits and Systems*, vol. 1, Aug. 2005, pp. 484–487.
29. L. Huai, X. Zou, Z. Liu, and Y. Han, "An Energy-Efficient AES-CCM Implementation for IEEE802.15.4 Wireless Sensor Networks," in *Int. Conf. on Netw. Security, Wirel. Commun. and Trusted Comput. (NSWCTC)*, vol. 2, Apr. 2009, pp. 394–397.
30. C. T. O. Otero, J. Tse, and R. Manohar, "AES Hardware-Software Co-design in WSN," in *Int. Symp. on Asynchr. Circuits and Syst. (ASYNC)*. IEEE, May 2015, pp. 85–92.
31. H. Yan, Y. Zhang, Z. Pang, and L. D. Xu, "Superframe Planning and Access Latency of Slotted MAC for Industrial WSN in IoT Environment," *IEEE Trans. on Ind. Inf.*, vol. 10, no. 2, pp. 1242–1251, May 2014.
32. D. Stanislowski, X. Vilajosana, Q. Wang, T. Watteyne, and K. Pister, "Adaptive Synchronization in IEEE802.15.4e Networks," *IEEE Trans. on Ind. Inf.*, vol. 10, no. 1, pp. 795–802, Feb. 2014.
33. M. R. Palattella, N. Accettura, L. A. Grieco, G. Boggia, M. Dohler, and T. Engel, "On Optimal Scheduling in Duty-Cycled Industrial IoT Applications Using IEEE802.15.4e TSCH," *IEEE Sensors J.*, vol. 13, no. 10, pp. 3655–3666, Oct. 2013.
34. TelosB datasheet. [Online]. Available: http://www.willow.co.uk/TelosB_Datasheet.pdf
35. CC2420 datasheet. [Online]. Available: https://inst.eecs.berkeley.edu/~cs150/Documents/CC2420.pdf
36. X. Vilajosana, P. Tuset, T. Watteyne, and K. Pister, *OpenMote: Open-Source Prototyping Platform for the Industrial IoT*. Springer International Publishing, 2015, pp. 211–222.
37. T. Watteyne, X. Vilajosana, B. Kerkez, F. Chraim, K. Weekly, Q. Wang, S. D. Glaser, and K. S. J. Pister, "OpenWSN: a Standards-Based Low-Power Wireless Development Environment," *Wiley Trans. on Emerg. Telecommun. Technol.*, vol. 23, no. 5, pp. 480–493, 2012.
38. S. Sciancalepore, G. Piro, G. Boggia, and L. A. Grieco, "Application of IEEE 802.15.4 Security Procedures in OpenWSN Protocol Stack," *IEEE Standards Education e-Magazine*, vol. 4, no. 2, pp. 1–9, 2014.