Handling Reboots and Mobility in 802.15.4 Security

Konrad-Felix Krentz
konrad-felix.krentz@hpi.de

Christoph Meinel
christoph.meinel@hpi.de

Hasso Plattner Institute, Prof.-Dr.-Helmert-Str. 2-3, 14482 Potsdam, Germany

ABSTRACT

To survive reboots, 802.15.4 security normally requires an 802.15.4 node to store both its anti-replay data and its frame counter in non-volatile memory. However, the only non-volatile memory on most 802.15.4 nodes is flash memory, which is energy consuming, slow, as well as prone to wear. Establishing session keys frees 802.15.4 nodes from storing anti-replay data and frame counters in non-volatile memory. For establishing pairwise session keys for use in 802.15.4 security in particular, Krentz et al. proposed the Adaptable Pairwise Key Establishment Scheme (APKES). Yet, APKES neither supports reboots nor mobile nodes. In this paper, we propose the Adaptive Key Establishment Scheme (AKES) to overcome these limitations of APKES. Above all, AKES makes 802.15.4 security survive reboots without storing data in non-volatile memory. Also, we implemented AKES for Contiki and demonstrate its memory and energy efficiency. Of independent interest, we resolve the issue that 802.15.4 security stops to work if a node's frame counter reaches its maximum value, as well as propose a technique for reducing the security-related per frame overhead.

Categories and Subject Descriptors

C.2.0. [**General**]: Security and protection; C.2.1. [**Network Architecture and Design**]: Wireless communication

General Terms

Security, Design.

Keywords

Internet of things, link layer security, rejuvenation, self-adaptiveness, 6LoWPAN.

1. INTRODUCTION

802.15.4 is a radio standard for wireless sensor and actuator networks [1]. Specially-designed protocols enable 802.15.4 nodes to communicate with each other and remote hosts using IPv6 [26, 18, 35, 32]. Such IPv6-enabled 802.15.4 networks are commonly referred to as 6LoWPAN networks, where 6LoWPAN stands for "IPv6 over Low-Power Wireless Personal Area Networks". 6LoWPAN networks are suitable for diverse application areas, including smart cities, industrial monitoring, and precision agriculture [20].

For wireless security, many 6LoWPAN protocols advocate using 802.15.4 security [26, 18, 35, 32]. 802.15.4 security filters out injected and replayed frames by adding both a message integrity code (MIC) and an incrementing frame counter to each outgoing frame. An incoming frame is only accepted if it has an authentic MIC and if its frame counter is greater than that of the last accepted frame from the frame's sender. Optionally, 802.15.4 security also encrypts the payload of outgoing frames. Both the generation of MICs and the encryption of payloads is done with a tweaked version of Counter with CBC-MIC (CCM) that uses AES-128 as block cipher [34].

Unfortunately, keeping track of every neighbor's frame counter is problematic since random-access memory (RAM) on 802.15.4 nodes is severely limited. For example, TelosB motes do only have 10KB of RAM [30]. Hence, in large 802.15.4 networks with mobile nodes, some anti-replay data needs to be swapped to non-volatile memory over time [21]. As the only non-volatile memory on most 802.15.4 nodes is flash memory, swapping usually is time and energy consuming [33]. Swapping may also become necessary in stationary 802.15.4 networks if an attacker tunnels the traffic between non-neighboring nodes verbatim. Such an attack is called a hidden wormhole [22]. In effect, non-neighboring nodes are tricked to believe they were neighbors and have to store anti-replay data about each other forever [25].

However, when aiming to survive reboots, all anti-replay data must be stored in non-volatile memory anyway. Reboots are, e.g., issued by the Contiki operating system when a process gets stuck [12]. Reboots also happen when exchanging batteries. After a reboot, anti-replay data stored in RAM get lost. Thus, to prevent replay attacks after reboots, anti-replay data must be stored across reboots. Moreover, two further issues arise if a node's frame counter starts over after a reboot. On the one hand, this causes a nonce reuse in 802.15.4 security and, on the other hand, neighbors consider the rebooted node's frames as replayed. Therefore, Sastry et al. considered storing a node's frame counter in non-volatile memory, too [31].

Session keys can not only obviate the need for swapping, but also fix all three issues with reboots without storing

ACSAC '15, December 07-11, 2015, Los Angeles, CA, USA

http://dx.doi.org/10.1145/2818000.2818002.

data in non-volatile memory. In fact, establishing session keys invalidates MICs from previous sessions. This, thus, allows nodes to securely delete anti-replay data about disappeared neighbors, rather than swapping this data [21]. Also, establishing new session keys after reboots causes replayed frames from previous sessions to be filtered out without storing anti-replay data across reboots. Finally, when establishing new session keys after reboots, a node's frame counter can start over without special treatment. This is because reusing nonces with new session keys is secure and, in the course of establishing new session keys, neighbors can be informed of a reset frame counter.

To establish pairwise session keys for use in 802.15.4 security in particular, Krentz et al. proposed the Adaptable Pairwise Key Establishment Scheme (APKES) [21]. Yet, APKES has four limitations. First, while APKES allows nodes to delete anti-replay data about disappeared neighbors, APKES never does so. Second, in APKES, nodes ignore so-called `HELLO`s from current neighbors. Since `HELLO`s initiate pairwise session key establishment, ignoring `HELLO`s from current neighbors causes a deadlock after reboots. Third, APKES only broadcasts `HELLO`s at startup and makes no effort to discover new neighbors at runtime. Forth, APKES focusses on deriving pairwise session keys from predistributed pairwise keys and lacks efficient support for predistributed network-wide keys, which are more manageable.

In this paper, we propose the Adaptive Key Establishment Scheme (AKES), which overcomes these limitations of APKES as follows:

- AKES pings neighbors that are incommunicado to check if they are still in range. If a neighbor does not reply, AKES deletes all data about that neighbor.
- AKES also processes `HELLO`s from current neighbors and potentially starts a new session with them. This change makes AKES survive reboots and thereby solves all three issues with reboots in 802.15.4 security.
- To discover new neighbors at runtime, AKES schedules the broadcasting of `HELLO`s, using Trickle [23]. This way, AKES self-adaptively reduces the rate of `HELLO`s if the neighborhood is stable, yet quickly reacts to neighborhood changes.
- Like APKES, AKES leaves its underlying key predistribution scheme exchangeable to account for the current dilemma that there is no one-size-fits-all pairwise key predistribution scheme [21, 3]. However, unlike APKES, AKES efficiently supports the network-wide key scheme, too.

2. RELATED WORK

In this section, we discuss related work on establishing pairwise session keys, avoiding swapping, and authenticating broadcast frames.

2.1 Establishing Pairwise Session Keys

802.15.4 security itself leaves key establishment unspecified, which spawned a lot of research. The major advantage of pairwise keys over a network-wide key is that they achieve graceful degradation in case of node compromises, as well as provide a basis for detecting compromised nodes [21]. However, establishing pairwise session keys using public-key cryptography (PKC), as proposed in [29, 28], is heavyweight on 802.15.4 nodes. PKC consumes much energy and has a high program memory footprint [17, 24]. Hardware-accelerated PKC resolves these issues [28], but incurs higher per unit costs. Key distribution centers (KDCs) avoid PKC [27], but their messaging overhead increases with the number of hops. Therefore, KDCs are even more energy consuming than software-implemented PKC [17].

Pairwise key predistribution schemes, on the other hand, are particularly lightweight. Such schemes neither involve KDCs nor PKC. Instead, pairwise key predistribution schemes use preloaded keying material to establish pairwise keys. A basic pairwise key predistribution scheme is, e.g., the fully pairwise keys scheme, where each node is preloaded with $n-1$ pairwise keys, each of which is shared with one of the $n-1$ other nodes in the network. Unfortunately, the memory consumption of the fully pairwise keys scheme is very high, which gives rise to diverse more memory-efficient pairwise key predistribution schemes, surveyed in [6]. There is, however, no one-size-fits-all pairwise key predistribution scheme [3, 21]. In this context, APKES contributed the workaround to hide diverse pairwise key predistribution schemes behind a common interface [21]. This enables developers to reuse the bulk of their code across different pairwise key predistribution schemes and users to choose the most appropriate key predistribution scheme for their use case.

2.2 Avoiding Swapping

Luk et al. considered storing anti-replay data in Bloom filters, which saves RAM and hence avoids swapping [25]. Unfortunately, Bloom filters sometimes falsely report that a frame was replayed and, disapprovingly, cause a high energy consumption [19]. After all, Bloom filters cannot help survive reboots, whereas session keys help both to avoid swapping and to survive reboots.

Another solution to avoid swapping appeared in the 802.15.4e amendment to 802.15.4 [2]. 802.15.4e contains the timeslotted channel hopping (TSCH) media access control (MAC) protocol. When using TSCH, every node is time synchronized and aware of the current absolute slot number (ASN). By using ASNs in lieu of frame counters, TSCH can avoid per neighbor anti-replay data altogether. As another benefit, TSCH allows senders to elide their frame counter, which saves energy. However, while TSCH avoids swapping, it neglects situations where one or more clock sources reboot or get compromised. Again, session keys can help keep 802.15.4 security secure, but, in our implementation, we use ContikiMAC [11]. ContikiMAC dispenses with time synchronization, which saves communication overhead. In addition, our implementation elides frame counters by tailoring the last bits (LB) optimization to 802.15.4 security [25, 16], as will be discussed in Section 5.2.

2.3 Authenticating Broadcast Frames

While pairwise session keys can serve for securing unicast frames, they cannot directly be used for securing broadcast frames. One can resort to securing broadcast frames using group session keys, but this solution sacrifices compromise resilience. To this end, Krentz et al. proposed the Easy Broadcast Encryption and Authentication Proto-

col (EBEAP) [21]. The idea of EBEAP is as follows. Suppose a node u wants to securely send a broadcast frame f. Then, u adds a frame counter to f to obtain f' and sends two broadcast frames. The first broadcast frame contains MICs $m_0 \| m_1 \| \ldots$ over f' for each of u's neighbors $v_0, v_1, \ldots$. Thereby, m_i is generated using the pairwise session key between u and v_i. Upon receipt, neighbor v_i extracts its corresponding MIC m_i and buffers it in a ring buffer. For this, v_i has to have its index $i = I_{v_i,u}$ in the neighbor list of u. These indices are distributed by APKES, as well as AKES. The second broadcast is f' itself. Upon receipt, v_i generates a MIC over f' using the pairwise session key between v_i and u. If the generated MIC is buffered and if f' is fresh, f' will be accepted. On demand, EBEAP can also encrypt the payload of f' using a group session key. Thus, a compromised node can decrypt broadcast frames of its neighbors, but can never impersonate its neighbors (provided that the employed pairwise key establishment scheme is inoculated and opaque [21, 9]).

3. AKES: ADAPTIVE KEY ESTABLISHMENT SCHEME

Building upon APKES, AKES leaves its underlying key predistribution scheme exchangeable. The plugged-in key predistribution scheme merely provides AKES with shared secrets. Based on shared secrets, AKES establishes pairwise and, if needed, group session keys. Group session keys can serve for securing both unicast and broadcast frames if compromise resilience is not an issue. In particular, this avoids the overhead of EBEAP when using the network-wide key scheme. On the other hand, if compromise resilience is needed, group session keys may still be required by EBEAP to encrypt the payload of broadcast frames. In the following, we detail the design of AKES, which is depicted in Figure 1. For a summary of our notations, see Appendix A.

3.1 Preloading Configuration Settings

An 802.15.4 node running AKES must be preloaded with addressing information, as well as keying material.

3.1.1 Addressing Information

The addressing scheme of 802.15.4 is organized as follows. Each node is assigned both a 2-byte personal area network (PAN) identifier and a variable-sized address. A PAN identifier identifies a set of nodes that belong to the same PAN. PAN identifiers can serve for separating co-located 802.15.4 networks, as well as for dividing 802.15.4 networks into subnetworks. An address identifies a specific node in a PAN. There are three kinds of addresses, namely 8-byte extended, 2-byte short, and 1-byte simple addresses. Extended addresses are globally unique, whereas short and simple addresses are just PAN unique.

AKES reuses the addressing scheme of 802.15.4 for requesting shared secrets from the plugged-in key predistribution scheme. Specifically, when requesting the shared secret with a node v, AKES inputs both the PAN identifier (denoted by PAN_v) and the address of v (denoted by ID_v) to the plugged-in key predistribution scheme. The plugged-in key predistribution scheme may raise an error if a specific address is not supported or unknown.

We note that AKES imposes two restrictions on the use of addresses. First, the addressing information that were used for requesting a shared secret must be used unchanged throughout a session. This is because AKES makes no effort to check whether a node owns alternative addressing information. Second, AKES is incompatible with current protocols for auto-configuring PAN identifiers, short addresses, and simple addresses [32, 1, 2]. The reason is that these protocols require 802.15.4 security to be up and running, but AKES has to reach this state in the first place. Therefore, AKES requires preloading a node with any addressing information it needs at runtime.

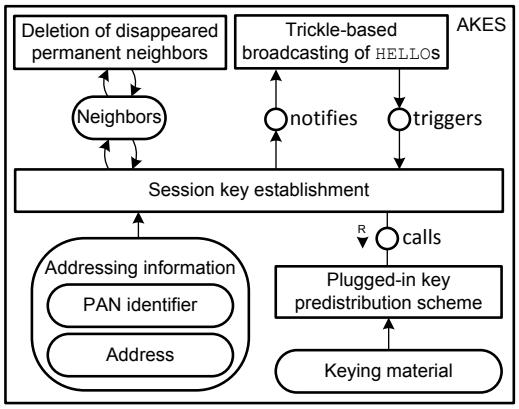


Figure 1: Design of AKES

3.1.2 Keying Material

Additionally, a node must be preloaded with keying material. The preloaded keying material is specific to the plugged-in key predistribution scheme. AKES explicitly supports the network-wide key scheme [6], the fully pairwise keys scheme [6], Blom's scheme [4, 10], as well as the random pairwise keys scheme [5]. All of them provide shared secrets at various trade-offs regarding inoculation, opaqueness, welcomingness, efficiency, and universality (IOWEU) [21]. When using the network-wide key scheme, the shared secret is always the network-wide key itself.

3.2 Establishing Session Keys

AKES uses a three-way handshake to establish session keys. The three messages are sent as three newly defined command frames, entitled `HELLO`, `HELLOACK`, and `ACK`. In contrast to data frames, command frames do not carry upper-layer traffic, but carry payload that is processed by the link layer.

The basic three-way handshake is shown in Figure 2. There, a node u establishes a pairwise session key with a neighbor v. Initially, u generates a cryptographic random number R_u and broadcasts a `HELLO` containing R_u. `HELLO`s are authenticated using either EBEAP or a group session key. As will be detailed shortly, authentic and unauthentic `HELLO`s are processed differently. Upon receipt, v requests the secret $K_{v,u}$ between v and u, and also generates a cryptographic random number R_v. Then, v derives the pairwise session key $K'_{v,u}$ as AES-128$(K_{v,u}, R_u \| R_v)$, thus reusing AES-128 as a key derivation function (KDF) [7]. Lastly, v sends a `HELLOACK` to u including R_v. The `HELLOACK` is authenticated by adding a MIC that is generated with $K'_{v,u}$. Upon receipt, u checks if the added MIC is authentic by deriving the pairwise session key $K'_{u,v}$ analogously. If suc-

u : Generate $R_u \in \{0,1\}^{64}$ randomly

$u \to *$: $\langle \texttt{HELLO}, PAN_u, ID_u, R_u, C_u \rangle$

v : Request shared secret $K_{v,u}$ from the plugged-in key predistribution scheme

v : Generate $R_v \in \{0,1\}^{64}$ randomly

v : $K'_{v,u} = \text{AES-128}(K_{v,u}, R_u \| R_v) \in \{0,1\}^{128}$

$v \to u$: $\langle \texttt{HELLOACK}, PAN_u, ID_u, PAN_v, ID_v, R_v, (\{K_{v,*}\}), [(I_{u,v}), C_v | I_{u,v}, C_{v,u}, C_{v,*}], P_u \rangle$

u : Request shared secret $K_{u,v}$ from the plugged-in key predistribution scheme

u : $K'_{u,v} = \text{AES-128}(K_{u,v}, R_u \| R_v) \in \{0,1\}^{128}$

$u \to v$: $\langle \texttt{ACK}, PAN_v, ID_v, PAN_u, ID_u, (\{K_{u,*}\}), [(I_{v,u}), C_u | I_{v,u}, C_{u,v}, C_{u,*}] \rangle$

u : Generate $R_u \in \{0,1\}^{64}$ randomly

Figure 2: Basic three-way handshake for establishing session keys

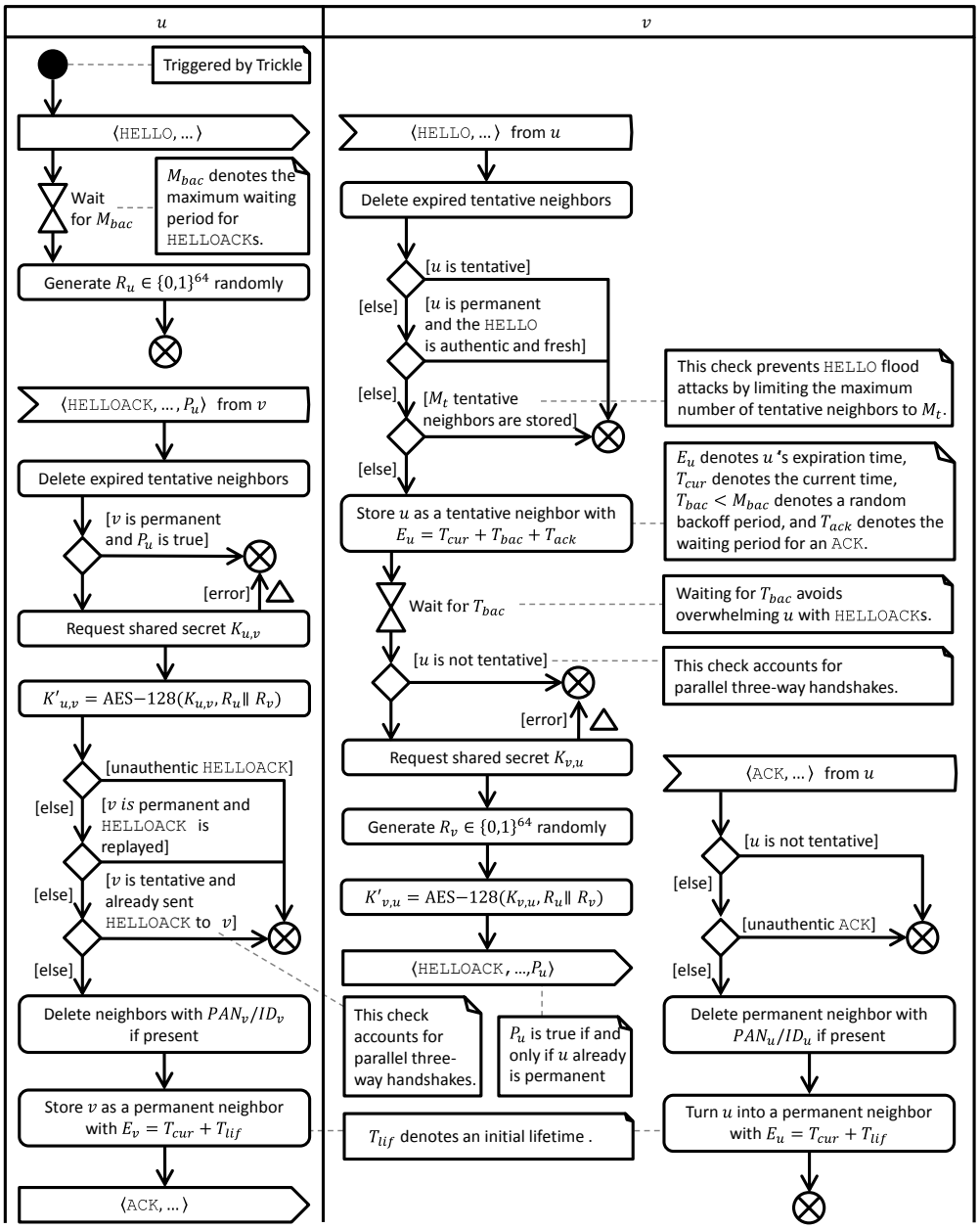


Figure 3: Detailed instructions for processing HELLOs, HELLOACKs, and ACKs

cessful, u sends an ACK to v. The ACK also contains a MIC that is generated with $K'_{u,v}$. Upon receipt, v checks if the contained MIC is authentic. If so, u and v have agreed on their pairwise session key $K'_{v,u} = K'_{u,v}$. Eventually, u changes R_u to prepare for broadcasting the next HELLO.

During the three-way handshake, u and v can piggyback their group session keys $K_{u,*}$ and $K_{v,*}$ on ACKs and HELLOACKs, respectively. Thereby, $K_{u,*}$ and $K_{v,*}$ are encrypted using $K'_{u,v}$ and $K'_{v,u}$, respectively. Additionally, EBEAP requires piggybacking the indices $I_{u,v}$ and $I_{v,u}$ on HELLOACKs and ACKs, respectively (see Section 2.3). These indices are also required by our optional LB optimization (to be explained in Section 5.2). Without LB optimization, each node adds its frame counter to outgoing frames and keeps track of its neighbors' frame counters, as usual. Table 1 summarizes all per neighbor data.

Detailed instructions for processing HELLOs, HELLOACKs, and ACKs are given in Figure 3. Thereby, AKES distinguishes between tentative and permanent neighbors. While tentative neighbors are potentially created upon receipt of HELLOs, permanent neighbors are potentially created upon receipt of HELLOACKs and ACKs. The main difference between tentative and permanent neighbors is that data frames may only be sent from and to permanent neighbors.

A crucial change to APKES is that a node v also handles HELLOs from a permanent neighbor u. For example, if u reboots, v will receive an unauthentic HELLO from u. This causes v to store u as a tentative neighbor, but, at the same time, to keep u as a permanent neighbor. As the ACK from u arrives, v deletes the permanent neighbor u and turns the tentative neighbor u into a permanent one. This effectively starts a new session between u and v. The more common case is that v receives fresh authentic HELLOs from u, which are silently discarded. However, a subtlety arises in conjunction with EBEAP. If EBEAP's first broadcast frame containing the MICs is missed, a HELLO from u is considered unauthentic by v. In this case, v will send a HELLOACK to u, but with the P_u flag set, which indicates that u is currently stored as a permanent neighbor by v. Upon receipt, u discards HELLOACKs from permanent neighbors with the P_u flag set right away, which avoids starting a new session unnecessarily.

3.3 Handling Mobility

Owing to mobility or changing surroundings, neighbors can get out of range and new ones can come into range. Below, we detail how AKES self-adapts to such neighborhood changes by deleting disappeared permanent neighbors and "Trickling" HELLOs to discover new neighbors.

3.3.1 Deletion of Disappeared Permanent Neighbors

When a permanent neighbor expires, AKES checks if that neighbor is still in range by exchanging two command frames, as shown in Figure 4. Specifically, to check if a permanent neighbor v is still in range, a node u sends an authenticated UPDATE to v. Upon receipt of a fresh authentic UPDATE, v replies with an authenticated UPDATEACK. Finally, as u receives v's fresh authentic UPDATEACK, u prolongs v's expiration time. If, on the other hand, no authentic UPDATEACK returns after resending the UPDATE a few times, u eventually gives up and deletes v. Note that when v receives an authentic UPDATE from u, v prolongs u's expiration time, too. Prolonging a permanent neighbor's expiration time is also done implicitly upon receiving any fresh authentic frame, which reduces explicit UPDATE/UPDATEACK interactions.

$$u \to v : \langle \texttt{UPDATE}, PAN_u, ID_u, PAN_v, ID_v, [C_u | C_{u,v}, C_{u,*}] \rangle$$
$$v \to u : \langle \texttt{UPDATEACK}, PAN_u, ID_u, PAN_v, ID_v, [C_v | C_{v,u}, C_{v,*}] \rangle$$

Figure 4: Interaction for checking if a permanent neighbor is still in range

Table 1: Data stored by u per neighbor v

Variable	Description
status	Stores if v is either permanent or tentative and, if tentative, whether a HELLOACK was already sent to v
PAN_v	v's PAN identifier
ID_v	v's extended, short, or simple address
E_v	v's expiration time
$K'_{u,v}$	u and v's pairwise session key - can be freed after the three-way handshake when using group session keys only
$K_{v,*}$	v's optional group session key
$I_{u,v}$	Part of EBEAP - u's index in the neighbor list of v
C_v	If LB optimization is off, C_v stores the frame counter of the last accepted frame from v
$C_{v,u}$	Part of the LB optimization - frame counter of the last accepted unicast frame from v
$C_{u,v}$	Part of the LB optimization - frame counter of the last outgoing unicast frame to v
$C_{v,*}$	Part of the LB optimization - frame counter of the last accepted broadcast frame from v
B_v	Part of the LB optimization - flag that stores if the last accepted frame from v was a broadcast frame or not
H_v	Part of Trickle - flag that stores if v sent a fresh authentic HELLO since the last time u broadcasted a HELLO

3.3.2 Trickle-Based Broadcasting of HELLOs

Establishing session keys with new neighbors requires broadcasting a HELLO. Thereby, the challenge is to broadcast as few HELLOs as possible, while quickly reacting to neighborhood changes. AKES adopts Trickle to achieve exactly this [23]. Trickle is an algorithm for disseminating information in wireless sensor networks. Though Trickle does not lend itself to the problem of scheduling the broadcasting of HELLOs, it can be tailored for this problem.

Trickle's parameters and variables are given in Table 2 and 3, respectively. Initially, Trickle sets $c = 0$ and $I = I_{\min}$. Furthermore, Trickle selects a random instant $t \in [\frac{I}{2}, I)$. Every time Trickle receives a consistent transmission, c is incremented. The notion of "consistent transmission" is application specific. At time t, Trickle transmits if $c < k$. What is being transmitted is also application specific. Finally, at time I, the end of the current interval, Trickle sets $c = 0$, $I = \min\{I \times 2, I_{\max}\}$, and selects a random instant $t \in [\frac{I}{2}, I)$. Subsequently, proceeds with transmitting

Table 2: Trickle's parameters

Parameter	Description
$I_{\min}$	Minimum interval size
$I_{\max}$	Maximum interval size
k	Redundancy constant

Table 3: Trickle's variables

Variable	Description
c	Counter
I	Current interval size
t	Instant within the current interval

at time t if $c < k$ and so on. Whenever an inconsistency is observed, Trickle can be reset by starting over with $c = 0$, $I = I_{\min}$, and some $t \in [\frac{I}{2}, I)$.

AKES configures Trickle to schedule the broadcasting of `HELLO`s as follows. AKES defaults to $I_{\min} = 30s$ and $I_{\max} = I_{\min} \times 2^8 = 128$min. In general, $I_{\min}$ should be greater than M_{bac} to avoid broadcasting another `HELLO` while still waiting for `HELLOACK`s. k defaults to 2, which emerged as a sensible default [23]. As consistent transmissions, AKES counts fresh authentic `HELLO`s. More specifically, upon receipt of a fresh authentic `HELLO` from v, u increments c if and only if v sent no fresh authentic `HELLO` since the last time u broadcasted a `HELLO`. For this, AKES maintains the H_v flag per permanent neighbor v, as shown in Table 1. Altogether, this configuration dramatically lowers the frequency of `HELLO`s if the neighborhood is stable. To quickly react to neighborhood changes, AKES resets Trickle when having added $\max\{\lfloor \frac{n}{4} \rfloor, 1\}$ permanent neighbors during the current interval, where n is the current total number of permanent neighbors. New permanent neighbors turned out to be a good indicator of far-reaching neighborhood changes.

4. SECURITY ANALYSIS

Observe that AKES does not consider unauthentic `HELLO`s to be consistent transmissions. This is to prevent attacks where an attacker suppresses `HELLO`s so as to prevent nodes from discovering each other. By counting only fresh authentic `HELLO`s, such attacks are complicated. Still, an attacker could have compromised nodes and broadcast large numbers of fresh authentic `HELLO`s. To also withstand such situations, AKES only increments c upon receipt of a fresh authentic `HELLO` from v if H_v is unset and thereupon sets the H_v flag. These flags are unset when broadcasting a `HELLO`. In effect, each node definitely broadcasts a `HELLO` at some point.

A related issue are `HELLO` flood attacks, where an attacker aims to expend energy and memory of receivers. AKES mitigates `HELLO` flood attacks by limiting the number of tentative neighbors. As soon as the maximum number of tentative neighbors is reached, AKES sheds `HELLO`s. To reinforce this mitigation technique, the maximum number of tentative neighbors M_{ten} should be low, the maximum random backoff period M_{bac} should be long, and the waiting period for `ACK`s T_{ack} should be long, too.

Furthermore, AKES prevents the replay of `HELLOACK`s and `ACK`s. The replay of `HELLOACK`s from permanent neighbors is directly prevented through frame counters. Also the replay of `HELLOACK`s from deleted neighbors is impossible since the random number R_u will have changed, which affects the derived pairwise session key and hence the expected MIC. The replay of `ACK`s is pointless since a tentative neighbor is turned into a permanent neighbor just once. The situation is different when a permanent neighbor was deleted. Then, an attacker could try becoming a permanent neighbor by first injecting a `HELLO` and second replaying a corresponding `ACK`. However, the random number R_v will have changed and therefore the derived pairwise session key, too. Thus, the MIC of the replayed `ACK` will be unauthentic.

Another attack on AKES is a hidden wormhole, where an attacker tunnels the traffic between non-neighboring nodes verbatim. A hidden wormhole enables an attacker to switch a link on and off, as well as to deny to forward selected frames. These two incarnations of hidden wormholes are referred to as transient and selective hidden wormholes, respectively. AKES establishes session keys through hidden wormholes unobstructed. Fortunately, transient hidden wormholes do not cause AKES to constantly add and remove permanent neighbors because the expiration time of a permanent neighbor acts as a hysteresis. To counter selective hidden wormholes, we suggest authenticating acknowledgement frames. This way, senders can ensure that a frame was received. However, this solution is only suitable for unicast frames, but not for broadcast frames. Avoiding hidden wormholes in the first place is a subject of current research [22].

5. IMPLEMENTATION

This section details our implementation of AKES for the Contiki operating system [12]. In Section 5.1, we first describe how we integrated AKES into Contiki's 6LoWPAN stack. In Section 5.2, we propose a technique for reducing the security-related per frame overhead. Finally, in Section 5.3, we explain how we prevent 802.15.4 security from crashing when a frame counter reaches its maximum value.

5.1 Integrating AKES into Contiki

Contiki's 6LoWPAN stack follows the Chameleon architecture [13], which enables developers to exchange the implementation of each of its layers at compilation time. Also, the implementation of link layer security can be exchanged by configuring a different `llsec_driver`. Accordingly, our implementation of AKES is part of a new `llsec_driver` named `adaptivesec_driver`.

The `adaptivesec_driver` provides many configuration options. At the highest level, the user can choose between a non-compromise-resilient and a compromise-resilient configuration. In a non-compromise-resilient configuration, the `adaptivesec_driver` secures both unicast and broadcast frames using group session keys. Conversely, in a compromise-resilient configuration, the `adaptivesec_driver` secures unicast frames using pairwise session keys and broadcast frames using EBEAP.

The underlying key predistribution scheme of AKES is implemented by an exchangeable `akes_scheme`, as shown in Figure 5. The `init` function is called at startup and used for initialization purposes. The functions `get_secret_with_hello_sender` and `get_secret_with_helloack_sender` are used for requesting shared secrets from the plugged-in key predistribution scheme. Thereby, the pointer `addr` points either to an extended or short address. Currently, we ignore PAN identifiers and do not support simple addresses since Contiki only supports network-wide PAN identifiers, as well as lacks support for simple addresses anyway. As `akes_-`

```
struct akes_scheme {
  void (* init)(void);
  uint8_t* (* get_secret_with_hello_sender)
      (const linkaddr_t *addr);
  uint8_t* (* get_secret_with_helloack_sender)
      (const linkaddr_t *addr);
};
```

Figure 5: Structure of a pluggable key predistribution scheme

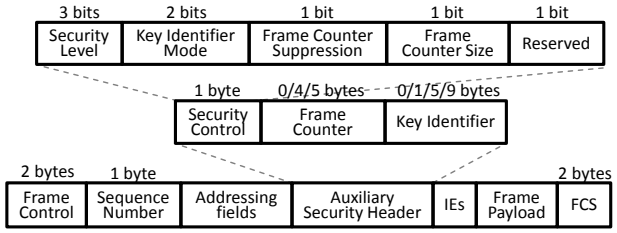


Figure 6: Format of a secured 802.15.4 frame.

`schemes`, we implemented the network-wide key scheme and the fully pairwise keys scheme. While the network-wide key scheme should go with a non-compromise-resilient configuration, the fully pairwise keys scheme should go with a compromise-resilient configuration.

5.2 Reducing the Security-Related Per Frame Overhead

The format of a secured 802.15.4 frame is depicted in Figure 6. The security-related fields are Frame Control, Auxiliary Security Header, and Frame Payload. The Frame Control field contains a flag that signals the presence of the Auxiliary Security Header field. The Auxiliary Security Header field is subdivided into the Security Control, Frame Counter, as well as Key Identifier fields. The Security Control field encodes the length of the Frame Counter field, the length of the Key Identifier field, as well as the employed security level. The optional Key Identifier field carries a reference to the employed key. The security level implies if the Frame Payload is encrypted, as well as the length of the added MIC. MICs are carried within the Frame Payload field.

To save energy, the `adaptivesec_driver` optionally elides the Auxiliary Security Header field of data frames completely. This works as follows. The security level defaults to a preloaded one. Alternatively, a default security level could be negotiated during session key establishment, just as well. Key identifiers are redundant since the `adaptivesec_driver` looks up session keys based on the addressing information of the sender. Finally, the Frame Counter field is elided by tailoring the LB optimization to 802.15.4 security [25, 16].

The idea of the LB optimization is to just send the x least significant bits of the frame counter. Receivers can estimate the higher order bits based on the last accepted frame counter. Similarly, what our `adaptivesec_driver` does is to set the Sequence Number field to the least significant byte of the frame counter. Since the Sequence Number field is sent anyway, the Frame Counter field becomes obsolete. However, making the LB optimization work flawlessly involves three intricacies.

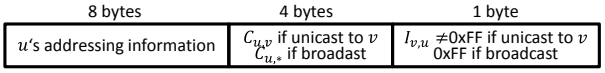


Figure 7: 13-byte CCM nonce of a sender u

First, each node has to maintain a number of separate counters. Specifically, each node u has to have both an overall counter for outgoing unicast frames (denoted by $C_{u,\bullet}$) and an overall counter for outgoing broadcast frames (denoted by $C_{u,*}$). Additionally, each node u has to maintain a separate counter for outgoing unicast frames per neighbor v (denoted by $C_{u,v}$). $C_{u,v}$ is initialized to $C_{u,\bullet}$, which is the only purpose of $C_{u,\bullet}$.

Second, replay protection and CCM nonces need adjustments. A unicast frame from v to u must contain a higher frame counter $C_{v,u}$ than was previously accepted. Likewise, a broadcast frame from v to u must contain a higher frame counter $C_{v,*}$ than was previously accepted. To avoid nonce reuses, the `adaptivesec_driver` creates nonces, as shown in Figure 7. The first 8 bytes contain the addressing information of the sender u. Thereafter, either $C_{u,v}$ or $C_{u,*}$ follows, depending on whether u sends a unicast frame to a neighbor v or a broadcast frame, respectively. Lastly, $I_{v,u}$ or, in case of a broadcast frame, 0xFF follows. Since 0xFF is not allowed as an index, unicast frames and broadcast frames always use distinct nonces. Including $I_{v,u}$ is necessary because a single group session key is used to secure unicast frames destined to different permanent neighbors.

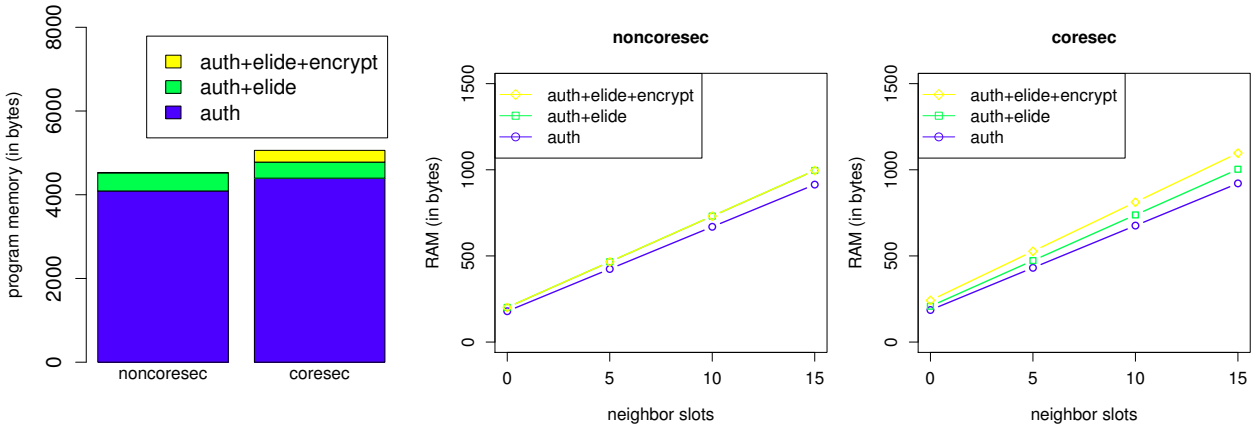
Third, a node u may lose track of the counters $C_{v,u}$ or $C_{v,*}$ of a permanent neighbor v. In effect, u no longer receives fresh authentic unicast or broadcast frames from v. To recover from this, we tweak the deletion of permanent neighbors. When LB optimization is enabled, v's lifetime is only prolonged upon receipt of a fresh authentic unicast frame from v if the last accepted frame from v was a broadcast frame. Likewise, v's lifetime is only prolonged upon receipt of a fresh authentic broadcast frame from v if the last accepted frame from v was a unicast frame. Thus, if any of u's counters gets out of sync with v's counters, u will send an `UPDATE` as v's lifetime expires. Since the corresponding `UPDATEACK` from v contains both $C_{v,u}$ and $C_{v,*}$ in whole, u can then resynchronize with v's counters. Fortunately, getting out of sync is unlikely since a node has to miss 2^8 consecutive unicast or broadcast frames [25, 16]. Still, an attacker may provoke such an occasion by launching a jamming attack.

5.3 Dealing with Used Up Frame Counters

Currently, 802.15.4 security stops to work when a frame counter reaches its maximum value, e.g., 2^{32} in case of 4-byte frame counters. Although this limitation does not seem to be of practical relevance since 2^{32} is large, a node's lifetime may exceed expectations and should not be limited. Since AKES survives reboots, the `adaptivesec_driver` simply issues a reboot when a frame counter reaches its maximum value. This effectively resets all frame counters and causes AKES to establish new session keys.

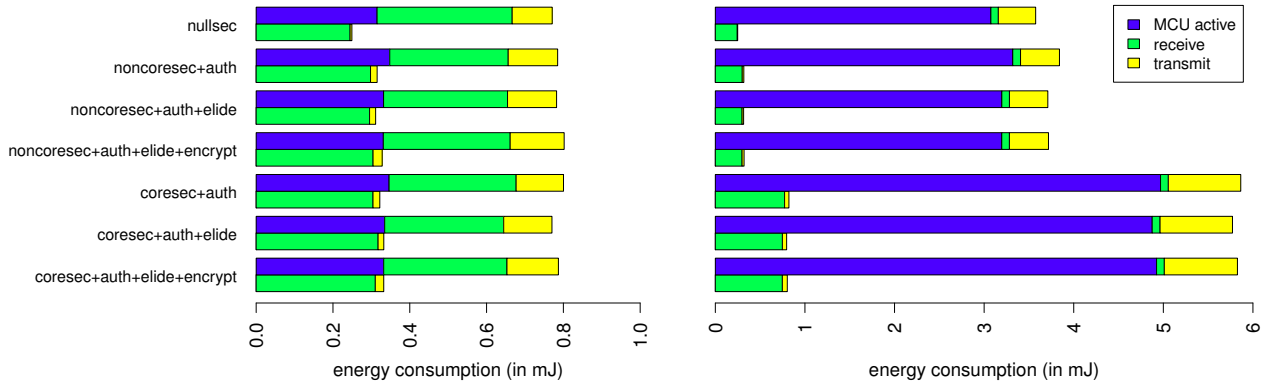
6. EVALUATION

In this section, we demonstrate the memory and energy efficiency of our `adaptivesec_driver`. Furthermore, we show-



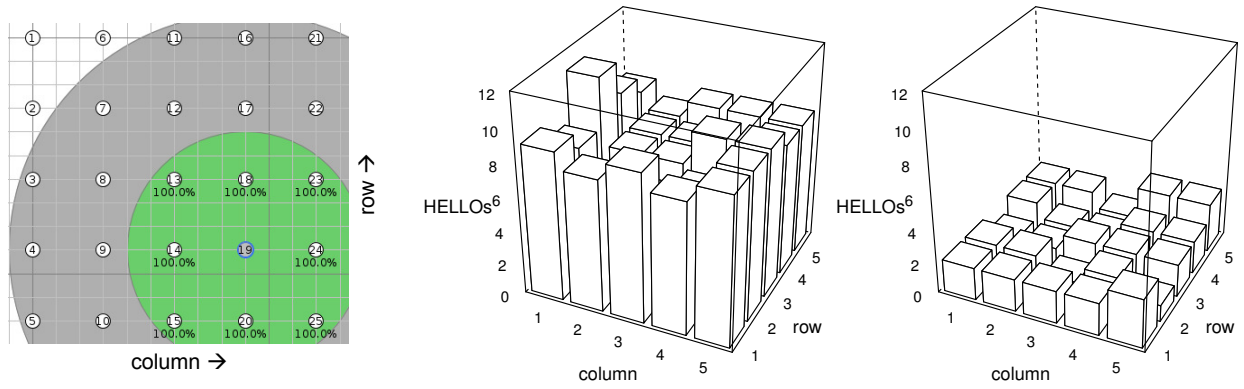
(a) (b) (c)

Figure 8: Memory footprint of the `adaptivesec_driver` on TelosB motes: (a) Program memory footprint (b) RAM footprint of a non-compromise-resilient configuration (c) RAM footprint of a compromise-resilient configuration



(a) (b)

Figure 9: Energy consumption of sending and receiving (a) a unicast or (b) a broadcast frame. While the upper bar of each pair shows the sender's energy consumption, the lower one shows the receiver's energy consumption. Each bar gives the mean over 100 samples.



(a) (b) (c)

Figure 10: Distribution of `HELLO`s: (a) Emulated network (b) Number of `HELLO`s between hour 0 and hour 6 (c) Number of `HELLO`s between hour 6 and hour 12. For raw data, see Appendix B.

case the efficacy of the Trickle-based scheduling of HELLOs.

6.1 Methods

The program memory and RAM footprint of the adaptivesec_driver was determined with the tools msp430-size and msp430-ram-usage, respectively. Thereby, three configurations were considered. First, in the *nullsec* configuration, link layer security was disabled as a baseline for determining the overhead of the adaptivesec_driver. Second, in the *noncoresec* configuration, the adaptivesec_driver was configured for non-compromise resilience. Third, in the *coresec* configuration, the adaptivesec_driver was configured for compromise resilience. More specifically, three modes of noncoresec and coresec were looked at. In the *auth* mode, both encryption and the eliding of Auxiliary Security Header fields was disabled. In the *auth+elide* mode, the eliding of Auxiliary Security Header fields was enabled. Finally, in the *auth+elide+encrypt* mode, encryption was enabled, too. Throughout, 128-bit keys, 4-byte frame counters, 8-byte MICs, short addresses, and TelosB motes were used.

Using the same configurations, modes, and hardware, the energy consumption of securing and unsecuring data frames was determined with Contiki's Energest tool [14]. Energest measures the durations of individual hardware activities and converts each duration t to an energy consumption E. For this, Energest employs the formulae $E = U \times I \times t$, where U is voltage and I is the current consumption of the respective hardware activity. As voltage, 2.2V was assumed. As current consumption, datasheet values of TelosB motes were assumed, i.e., 1.32mA when the MCU is active, 18.8mA when receiving, and 17.4mA when transmitting. The experimental setup was compromised of two TelosB motes - a sender and a receiver. On the sender side, the energy consumption was measured between sending a data frame and getting notified that the data frame was sent. On the receiver side, the energy consumption was measured between the receipt of the data frame and its arrival at the upper layer. All data frames carried 50 bytes of payload. The energy consumption of EBEAP's additional broadcast frame was included and ContikiMAC's burst support was disabled.

Finally, to showcase the efficacy of the Trickle-based scheduling of HELLOs, Contiki's network emulator Cooja was used [15]. The emulated network topology is shown in Figure 10a. It was emulated for 12 virtual hours.

6.2 Results

As shown in Figure 8, the program memory and RAM overhead of the adaptivesec_driver is significant. However, recall that the adaptivesec_driver not only implements AKES, but also the securing and unsecuring of frames. Furthermore, we left key lengths, the number of neighbor slots, etc. configurable. This enables users to trade off security against memory footprint. For example, Figure 10 shows that tuning the number of neighbor slots has a great effect on the RAM footprint. Also, a small amount of program memory and RAM can be saved by disabling encryption or the eliding of Auxiliary Security Header fields. Intriguingly, our adaptivesec_driver outperforms the "as is" implementation of 802.15.4 security by Daidone et al. in terms of program memory and RAM consumption [8].

The overhead in energy consumption for securing and unsecuring data frames is shown in Figure 9. The reason why broadcast frames consume much more energy than unicast frames is due to ContikiMAC [11]. Basically, ContikiMAC repeatedly sends a unicast frame until the intended receiver wakes up and sends an acknowledgement frame. By contrast, ContikiMAC repeatedly sends a broadcast frame for a whole wake-up interval. The difference in energy consumption between nullsec and noncoresec is insignificant. An exception to this is coresec since it requires sending an additional broadcast frame. We can also observe that eliding Auxiliary Security Header fields saves a small amount of energy. Surprisingly, eliding Auxiliary Security Header fields mainly saves processing overhead, presumably because all layers have to cope with less data. Lastly, since our adaptivesec_driver leverages a hardware-accelerated AES-128 implementation, which is a common feature of 802.15.4 transceivers, its processing overhead is small.

Especially when using coresec, broadcasting a secured frame, such as a HELLO, consumes a considerable amount of energy. This is why we put special emphasis on reducing the number of HELLOs in the design of AKES. Figure 10b shows the number of HELLOs after 6 hours. Once the network is stabilized every node only sends 1-3 HELLOs in 6 hours, as shown in Figure 10c. Of course, even more energy-efficient configurations are possible, but at the cost of slower response to neighborhood changes.

7. CONCLUSIONS AND FUTURE WORK

Normally, 802.15.4 security requires swapping and cannot survive reboots without storing data in non-volatile memory. Moreover, 802.15.4 security crashes when a frame counter reaches its maximum value. We have proposed AKES to overcome all these problems at once. AKES obviates the need for swapping and makes 802.15.4 security survive reboots without storing data in non-volatile memory. In particular, AKES enables a node to reboot when any of its frame counters reaches its maximum value. On the other hand, the more dynamic an 802.15.4 network is, the more energy is consumed by AKES for establishing session keys. That said, future work might improve on the mobility support of AKES. Besides, future work should address key revocation and rekeying.

8. REFERENCES

[1] IEEE Standard 802.15.4, 2011. http://standards.ieee.org/getieee802/download/802.15.4-2011.pdf.

[2] IEEE Standard 802.15.4e, 2012. http://standards.ieee.org/getieee802/download/802.15.4e-2012.pdf.

[3] C. Alcaraz, J. Lopez, R. Roman, and H.-H. Chen. Selecting key management schemes for WSN applications. *Computers & Security*, 31(38):956–966, 2012.

[4] R. Blom. An optimal class of symmetric key generation systems. In *Advances in Cryptology - EUROCRYPT 84*, pages 335–338. Springer, 1984.

[5] H. Chan, A. Perrig, and D. Song. Random key predistribution schemes for sensor networks. In *Proceedings of the 2003 IEEE Symposium on Security and Privacy*, pages 197–213. IEEE, 2003.

[6] C.-Y. Chen and H.-C. Chao. A survey of key distribution in wireless sensor networks. *Security and Communication Networks*, 2011.

[7] L. Chen. Recommendation for Key Derivation Using Pseudorandom Functions, 2009. NIST Special Publication 800-108.

[8] R. Daidone, G. Dini, and G. Anastasi. On evaluating the

performance impact of the IEEE 802.15.4 security sub-layer. *Computer Communications*, 47(0):65 – 76, 2014.

[9] J. Deng, C. Hartung, R. Han, and S. Mishra. A practical study of transitory master key establishment for wireless sensor networks. In *Proceedings of the First IEEE/CreateNet Conference on Security and Privacy for Emerging Areas in Communication Networks (SecureComm 2005)*, pages 289 – 302. IEEE, 2005.

[10] W. Du, J. Deng, Y. S. Han, P. K. Varshney, J. Katz, and A. Khalili. A pairwise key predistribution scheme for wireless sensor networks. In *Proceedings of the 10th ACM Conference on Computer and Communications Security (CCS '03)*, pages 42–51. ACM, 2003.

[11] A. Dunkels. The ContikiMAC radio duty cycling protocol. Technical Report T2011:13, Swedish Institute of Computer Science, 2011.

[12] A. Dunkels, B. Grönvall, and T. Voigt. Contiki - a lightweight and flexible operating system for tiny networked sensors. In *Proceedings of the 29th Annual IEEE International Conference on Local Computer Networks (LCN 2004)*, pages 455–462. IEEE, 2004.

[13] A. Dunkels, F. Österlind, and Z. He. An adaptive communication architecture for wireless sensor networks. In *Proceedings of the 5th International Conference on Embedded Networked Sensor Systems (SenSys '07)*, pages 335–349. ACM, 2007.

[14] A. Dunkels, F. Osterlind, N. Tsiftes, and Z. He. Software-based on-line energy estimation for sensor nodes. In *Proceedings of the 4th Workshop on Embedded Networked Sensors (EmNets '07)*, pages 28–32. ACM, 2007.

[15] J. Eriksson, F. Österlind, N. Finne, N. Tsiftes, A. Dunkels, T. Voigt, R. Sauter, and P. J. Marrón. COOJA/MSPSim: interoperability testing for wireless sensor networks. In *Proceedings of the 2nd International Conference on Simulation Tools and Techniques (Simutools '09)*. ICST, 2009.

[16] M. G. Gouda, Y. ri Choi, and A. Arora. Antireplay protocols for sensor networks. In J. Wu, editor, *Handbook on Theoretical and Algorithmic Aspects of Sensor, Ad Hoc Wireless, and Peer-to-Peer Networks*, pages 561–574. CRC, 2005.

[17] J. Großschädl, A. Szekely, and S. Tillich. The energy cost of cryptographic key establishment in wireless sensor networks. In *Proceedings of the 2nd ACM Symposium on Information, Computer and Communications Security (ASIACCS '07)*, pages 380–382. ACM, 2007.

[18] J. Hui and P. Thubert. Compression Format for IPv6 Datagrams over IEEE 802.15.4-Based Networks. RFC 6282, 2011. Updates RFC 4944.

[19] D. Jinwala, D. Patel, S. Patel, and K. Dasgupta. Optimizing the replay protection at the link layer security framework in wireless sensor networks. 2012.

[20] E. Kim, D. Kaspar, and J. Vasseur. Design and Application Spaces for IPv6 over Low-Power Wireless Personal Area Networks (6LoWPANs). RFC 6568, 2012.

[21] K.-F. Krentz, H. Rafiee, and C. Meinel. 6LoWPAN security: adding compromise resilience to the 802.15.4 security sublayer. In *Proceedings of the International Workshop on Adaptive Security & Privacy Management for the Internet of Things (ASPI '13)*. ACM, 2013.

[22] K.-F. Krentz and G. Wunder. 6LoWPAN security: avoiding hidden wormholes using channel reciprocity. In *Proceedings of the International Workshop on Trustworthy Embedded Devices (TrustED '14)*. ACM, 2014.

[23] P. Levis, T. Clausen, J. Hui, O. Gnawali, and J. Ko. The Trickle Algorithm. RFC 6206, 2011.

[24] A. Liu and P. Ning. TinyECC: a configurable library for elliptic curve cryptography in wireless sensor networks. Technical Report TR-2007-36, North Carolina State University, 2008.

[25] M. Luk, G. Mezzour, A. Perrig, and V. Gligor. MiniSec: a secure sensor network communication architecture. In *Proceedings of the 6th International Conference on Information Processing in Sensor Networks (IPSN '07)*, pages 479–488. ACM, 2007.

[26] G. Montenegro, N. Kushalnagar, J. Hui, and D. Culler. Transmission of IPv6 Packets over IEEE 802.15.4 Networks. RFC 4944, 2007.

[27] A. Perrig, R. Szewczyk, J. Tygar, V. Wen, and D. E. Culler. SPINS: security protocols for sensor networks. *Wireless networks*, 8(5), 2002.

[28] A. d. l. Piedra, A. Braeken, and A. Touhafi. Extending the IEEE 802.15.4 security suite with a compact implementation of the NIST P-192/B-163 elliptic curves. *Sensors*, 13(8):9704–9728, 2013.

[29] G. Piro, G. Boggia, and L. A. Grieco. Layer-2 security aspects for the IEEE 802.15.4e MAC. Internet-Draft, 2014. Version 3.

[30] J. Polastre, R. Szewczyk, and D. Culler. Telos: enabling ultra-low power wireless research. In *Proceedings of the 4th International Symposium on Information Processing in Sensor Networks (IPSN 2005)*, pages 364–369. IEEE, 2005.

[31] N. Sastry and D. Wagner. Security considerations for IEEE 802.15.4 networks. In *Proceedings of the 3rd ACM Workshop on Wireless Security (WiSe '04)*, pages 32–42. ACM, 2004.

[32] Z. Shelby, S. Chakrabarti, E. Nordmark, and C. Bormann. Neighbor Discovery Optimization for IPv6 over Low-Power Wireless Personal Area Networks (6LoWPANs). RFC 6775, 2012.

[33] N. Tsiftes, A. Dunkels, Z. He, and T. Voigt. Enabling large-scale storage in sensor networks with the coffee file system. In *Proceedings of the 2009 International Conference on Information Processing in Sensor Networks (IPSN '09)*, pages 349–360. IEEE, 2009.

[34] D. Whiting, R. Housley, and N. Ferguson. Counter with CBC-MAC (CCM). RFC 3610, 2003.

[35] T. Winter, P. Thubert, A. Brandt, J. Hui, R. Kelsey, P. Levis, K. Pister, R. Struik, J. Vasseur, and R. Alexander. RPL: IPv6 Routing Protocol for Low-Power and Lossy Networks. RFC 6550, 2012.

APPENDIX

A. NOTATIONS

Symbol	Meaning
$\|$	Concatenation
$u \to v$	u sends a unicast frame to v
$u \to *$	u sends a broadcast frame
$\langle X \rangle$	Authenticated content X
$\{X\}$	Encrypted content X
(X)	Optional content X
$[X\|Y]$	Either X or Y
AES-128(K, P)	AES encryption of a 16-byte plain text P with a 128-bit key K

B. RAW DATA OF FIGURE 10B AND 10C

8	7	8	8	8
9	7	7	6	8
11	7	7	9	9
8	6	6	7	9
9	8	9	8	9

2	2	1	3	3
2	1	1	2	2
1	1	2	2	2
2	2	1	2	1
2	2	2	2	3