Link Layer Key Revocation and Rekeying for the Adaptive Key Establishment Scheme

Benedikt Bock, Jan-Tobias Matysik, Konrad-Felix Krentz, and Christoph Meinel
Hasso-Plattner-Institute/University of Potsdam, Prof.-Dr.-Helmert-Str. 2-3, 14482 Potsdam, Germany

***Abstract*—While the IEEE 802.15.4 radio standard has many features that meet the requirements of Internet of things applications, IEEE 802.15.4 leaves the whole issue of key management unstandardized. To address this gap, Krentz *et al.* proposed the Adaptive Key Establishment Scheme (AKES), which establishes session keys for use in IEEE 802.15.4 security. Yet, AKES does not cover all aspects of key management. In particular, AKES comprises no means for key revocation and rekeying. Moreover, existing protocols for key revocation and rekeying seem limited in various ways. In this paper, we hence propose a key revocation and rekeying protocol, which is designed to overcome various limitations of current protocols for key revocation and rekeying. For example, our protocol seems unique in that it routes around IEEE 802.15.4 nodes whose keys are being revoked. We successfully implemented and evaluated our protocol using the Contiki-NG operating system and aiocoap.**

***Index Terms*—IEEE 802.15.4, key management, key establishment, key revocation, rekeying, link layer security, MAC security**

I. Introduction

The IEEE 802.15.4 radio standard is widely adopted for implementing Internet of things (IoT) applications. Its widespread adoption in this domain mainly comes down to its support for energy-efficient medium access control (MAC) protocols, as required by battery-powered IoT devices. Further features of IEEE 802.15.4 that suit IoT applications are its support for mesh topologies, the cheapness of IEEE 802.15.4 transceivers, and its security mechanisms.

However, while IEEE 802.15.4 standardizes mechanisms for securing radio frames, it leaves the whole issue of key management unstandardized. In practice, this often leads to the usage of predistributed network-wide keys, which are highly susceptible to "on-site attacks", such as electromagnetic side-channel attacks [2] or physical tampering [3]. Using such on-site attacks, an attacker can compromise a network-wide key and subsequently inject IEEE 802.15.4 radio frames unobstructed. To alleviate on-site attacks, Krentz *et al.* proposed the Adaptive Key Establishment Scheme (AKES), which can establish pairwise session keys based on predistributed keying material [1]. Furthermore, AKES supports to trade off compromise resilience against memory efficiency by supporting a variety of key predistribution schemes, such as Blom's scheme or the fully pairwise keys scheme [4].

Yet, aspects left unaddressed by AKES are key revocation and rekeying. Key revocation is "the process of removing keys from operational use [...]", whereas rekeying is the process of putting new keys into operational use [5]. Key revocation and rekeying becomes necessary when predistributed keying material leaks to an attacker, but also if a user wishes to evict an IoT device from an IEEE 802.15.4 network for other reasons, such as due to relocating an IoT device from one IEEE 802.15.4 network to another. In the case of AKES, link layer neighbors of an evicted IoT device need to remove their pairwise session keys with the evicted IoT device and all nodes, except for the node that shall be evicted, potentially need to update their predistributed keying material.

This paper's main contribution is to propose a key revocation and rekeying protocol for AKES. A preliminary version of which appeared in a master's thesis [6]. Our protocol features:

- Energy-efficient operation
- Fast key revocation and rekeying
- Reliable operation thanks to
  - retransmitting messages
  - trying alternative paths if errors occur
  - avoiding paths via evicted nodes
- Live feedback on the progress
- Denial-of-sleep-resilient protocol design

II. Related Work

Several protocols were proposed for key revocation in wireless mesh networks like IEEE 802.15.4 networks. In general, the existing protocols either follow a centralized or a distributed approach. All centralized approaches have in common that a single instance in the network is responsible for deciding which node to evict and for informing all other nodes about an eviction. By contrast, when following a distributed approach, these responsibilities are shared by all nodes. In particular, consensus must be reached on which node to evict.

Distributed key revocation protocols were, e. g., proposed in [7]–[9]. Therein, a common assumption is that nodes have intrusion detection systems (IDSs) so as to detect and pinpoint malicious nodes, thus necessitating extra software components, which may be hard to implement given the severe resource constraints of most IoT devices. Moreover, all distributed key revocation protocols presume a static network topology that is known at time of deployment [7]–[9].

Most centralized key revocation protocols revoke keys via flooding, e. g. by routing a broadcast message from the central instance to each node [10]–[12]. However, this is realized without routing around nodes that shall be evicted, although such nodes may be compromised and hence may deny to

978-1-5386-4980-0/19/$31.00 ©2019 European Union

forward those messages. Moreover, we are not aware of any flooding-based centralized key revocation protocol that provides feedback. In fact, a common assumption is that all broadcast messages do in fact reach their destination. Another problem of flooding-based centralized key revocation protocols relates to the authentication of the broadcasted message. Authenticating it with a symmetric key is inappropriate since this would necessitate the use of a network-wide key. Digital signatures and hash chains constitute alternatives [10]–[12], but are much more resource consuming. Also, using digital signatures might render IoT devices susceptible to denial-of-sleep attacks [1].

Raza *et al.* proposed a centralized key revocation protocol, where the central instance sends a distinct unicast message to each node. This allows nodes to give feedback to the central instance by sending a positive reply or not. Furthermore, unicast messages can be secured via pairwise symmetric keys. However, a remaining issue with Raza *et al.*'s protocol is that it routes unicast messages via evicted nodes.

Our proposed protocol for key revocation belongs to the class of unicast-based centralized key revocation protocols. Yet, in contrast to Raza *et al.*'s protocol, ours (i) routes unicast messages around nodes that shall be evicted for better reliability, and (ii) supports both key revocation and rekeying. Additionally, our protocol inherits all of the advantages of unicast-based centralized key revocation protocols.

III. Proposed Protocol for Key Revocation and Rekeying

As mentioned in Section I, in order to evict a node from an IEEE 802.15.4 network, we aim to revoke any pairwise session key with the node and to update the predistributed keying material of every other node in the network. In the following sections, we will explain (A) which participants exist and their scope, (B) which communication takes place between the participants, and (C) the message contents.

A. Participants

We distinguish between three participants, namely *base station*, *node* and *border node*.

1) Base Station: The base station provides a user interface to control the IoT network and is therefore not part of the network itself. The base station is responsible for controlling the eviction process and for providing feedback about progress and success of the eviction process. If an IDS is used, such an eviction process can be started automatically. Only one base station per network is foreseen in our protocol, which holds a connection to every border node in the network. The base station must have knowledge about every deployed node, but needs no information about the network topology. Furthermore, it has a pairwise shared secret with every node in the network. Finally, the base station is responsible to stop an eviction process.

2) Node: A node is any IoT device within the IEEE 802.15.4 network. In our protocol, a node fulfills two functions. Either it processes protocol messages that are destined to the node itself or forwards them. The goal of processing the request on node u is to disable any further communication with a node m that should be evicted from the network. Either u replaces the former predistributed key with a new one, or m is added to a node revocation list (NRL) to prevent m from re-entering the network. Furthermore, any other keys used for the communication with m on u are removed.

3) Border Node: A border node communicates with the base station. It needs a connection to the base station without any other network nodes in between. It is responsible for implementing source routing of protocol messages to nodes, as we will detail in Section III-B. Although a border node is part of the network, we assume that it has an unlimited power supply due to its increased workload.

B. Communication Flow

Now we will have a look at the overall eviction process by illustrating the communication that takes place. There are two flows of communication, (1) the *control flow* between the base station and the border nodes and (2) the *distribution flow* inside the network, which is triggered by the control flow.

1) Control Flow: The control flow is used by the base station to manage the process of evicting node m. It uses two message types. (i) The *eviction request* is used to start the process, to specify which nodes u in the network should exclude m, and to stop the eviction process. (ii) The *eviction reply* informs the base station about the request's result. If the request was successful, the reply additionally contains new nodes that were discovered by the border node (see Section III-B2). The control flow is built on the Constrained Application Protocol (CoAP).

We will now examine the sequence of messages during an eviction process. Fig. 1 depicts the communication between the base station and a border node. If there are multiple border nodes, this will happen for each border node concurrently. To start a new eviction process the base station sends an *eviction request* to the border node b. This request contains only b as the node that should evict m. The border node b will process the request. Once the processing finished, the border node b will send an *eviction reply*. If the processing failed, the base station will stop the process immediately for this border node. If the processing succeeded, the reply contains the direct neighbors of b. The base station adds these nodes to a queue unless they were already marked as succeeded. Whenever a node has successfully evicted the node m, it has to be removed from the queue across all concurrent processes and is marked as succeeded. In further *eviction requests* the border node iterates over all nodes u and starts a distribution (see Section III-B2). The amount of target nodes u included per *eviction request* is limited by the maximum payload per Constrained Application Protocol (CoAP) message. Once all distributions are finished or timed out the border node sends an *eviction reply* to the base station containing nodes u that successfully evicted m and u's neighbors. The base station will try to reach failed nodes u over other border nodes. When the queue is empty and the *eviction replies* do not contain new

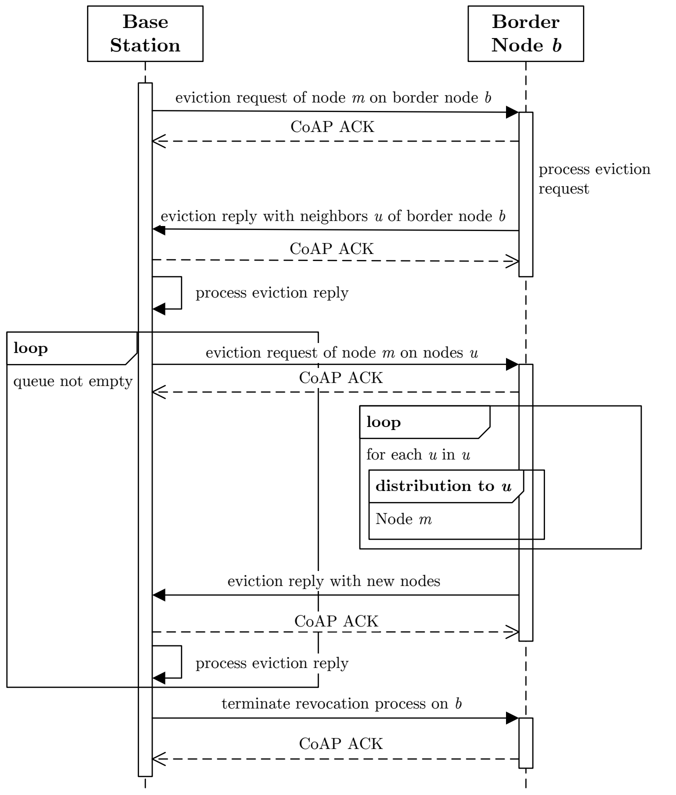
Base Station
Border Node b
eviction request of node m on border node b
CoAP ACK
process eviction request
eviction reply with neighbors u of border node b
CoAP ACK
process eviction reply
loop
queue not empty
eviction request of node m on nodes u
CoAP ACK
loop
for each u in u
distribution to u
Node m
eviction reply with new nodes
CoAP ACK
process eviction reply
terminate revocation process on b
CoAP ACK

Fig. 1: Control flow between the base station and a border node to evict node m from the network

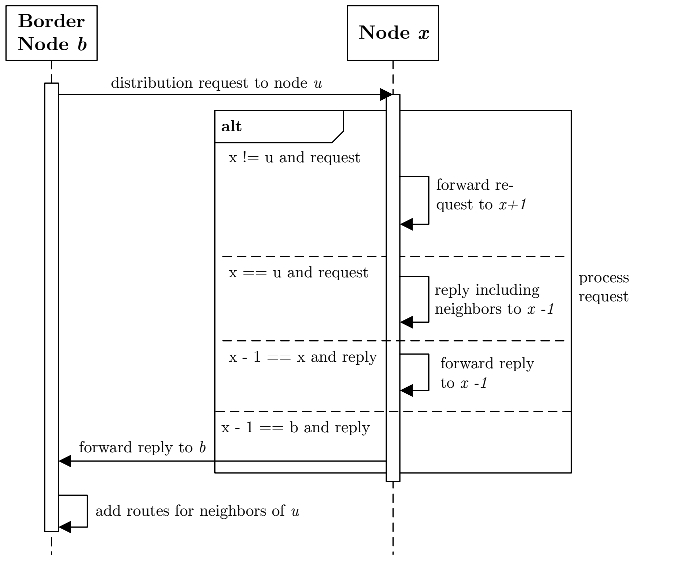

Fig. 2: Distribution flow for a request to node u for evicting node m

nodes, the base station sends one last *eviction request* to the border node b in order to stop the eviction process on this border node. Once the eviction process is stopped on every border node, the whole eviction process is finished. The user feedback is updated whenever an *eviction reply* was processed by the base station. After finishing the whole process, the user gets feedback about nodes that could not be reached, e. g. due to topological constraints or jamming attacks.

2) Distribution Flow: The distribution flow is used within the network to distribute a request to evict a node m from a border node b to a node u along with any new keying material for node u contained in the corresponding *eviction request*. The distribution flow is embedded in the control flow and multiple distribution flows can run concurrently. In order to distribute these so-called *distribution requests*, we use a strict source routing approach at the link layer. This has the advantage that nodes do not need to store any additional information. So-called *distribution replies* are sent along the same route back to the border node. A *distribution reply* indicates success and contains any link layer neighbors of the sender. Since the whole eviction process starts with a border node (cf. Section III-B1) and continues with neighbors of successfully reached nodes, the border node can incrementally build up routing information. The node m that should be evicted is always ignored in the routing information.

We will now examine the distribution of a request to evict node m from a border node b to a node u. This flow is depicted in Fig. 2. Initially, b extracts any new keying material for u from the corresponding *eviction request*. Next, b chooses a route to u in its incremental routing information. Then, b sends a *distribution request* to the first hop x of the route. The node x will check if it is the node u or just a hop on the route. Depending on the result, x will send the *distribution request* to the next hop or process the request. Once the request was processed by u, u sends a *distribution reply* to b, which includes u's link layer neighbors. The reply is routed along the same route as the request. As soon as the border node receives the *distribution reply*, it will update its routing information with u's link layer neighbors.

C. Message Contents

Table I shows the contents of the messages mentioned in Section III-B. As already indicated, *eviction request* and *eviction reply* are used during the control flow. They are built on CoAP. We suggest securing these messages via DTLS [14]. Features like block-wise transfer or CoAP over TCP could be used [15], [16]. The *distribution request* and *distribution reply* are used during the distribution flow. Some contents of the *distribution request* originate from the *eviction request*.

IV. Implementation

In this section, we outline how we implemented our protocol. We implemented our approach based on Contiki-NG for all nodes. It is an open-source, cross-platform operating system for IoT devices with a focus on low-power communication. The used link layer protocol uses addresses with eight byte. We focus on the distribution of the messages in our implementation. Our implementation does not yet (i) authenticate messages, (ii) distribute new keys, and (iii) revoke more than one node within one eviction process. Filling these gaps is left for future work.

TABLE I: Message Formats

(a) *eviction request*

field	description
opts	Defines the purpose and content of the message. It has 8 bits. Starting from the least significant bit, the first bit determines whether the message contains new keys ($= 1$) or not ($= 0$). The second bit indicates a request to stop the process if set to 1.
node m	Link layer address of the node m that should be evicted from the network.
# dsts	Amount of nodes u, that should evict node m.
counter	One counter for each u. The counter is used by node u to detect replay attacks.
destinations	List of *# dsts* link layer addresses of destination nodes u that should evict node m.
dist keys	Key for each node *u* that is used during the distribution from border node *b* to *u*. Each entry contains a key $Key_{b,u}$ twice. Each encrypted for *b* and *u* using the shared secret with the base station (*Kb*, *Ku*) and the respective *counter* as nonce.
new keys	A new key for every node u. The *opts* field specifies if there are new keys. The keys are encrypted by deriving a nonce from the respective *counter*.
MICs	A list of MICs. The MIC is used by u to authenticate the field *new key*. Each MIC is generated by deriving a nonce from the respective *counter*.
$\mathrm{MIC_b}$	This MIC is used by the border node *b* to authenticate all fields. The nonce is derived from the *destinations* field.

(b) *eviction reply*

field	description
border id	Link layer address of the border node that built the message.
r	Number of reached destinations from the *eviction request*.
n	Number of nodes that were discovered while processing the *eviction request*.
reached destinations	List of link layer addresses that successfully evicted node m.
MICs	List of MICs for each *reached destination*.
new nodes	List of neighbors aggregated from the *distribution replies*.

(c) *distribution request*

field	description
hi	Hop index. It is incremented by 1 with every hop.
hc	Amount of hops from sender to destination.
opts	Same field as in *eviction request*. There are options, e. g. the termination bit, that do not have any effect on the nodes.
counter	The corresponding counter for the destination node u from the *eviction request*.
route	Link layer addresses of all hops from the sender to the destination. There are $hc + 1$ addresses.
node m	Copied from the *eviction request*.
$\{\mathrm{Key_{b,u}}\}_{\mathrm{Ku}}$	The corresponding *dist key* from the *eviction request*.
new key	The corresponding *new key* for the destination node u from the *eviction request*.
MIC	The corresponding *MIC* for the destination node u from the *eviction request*.

(d) *distribution reply*

field	description
hi	See *distribution request*.
hc	See *distribution request*.
route	See *distribution request* but reversed.
node m	See *distribution request*.
nc	Number of neighbors of the destination node u.
neighbors	List of link layer addresses of u's neighbors.
MIC	Used by the base station to verify that the correct node processed the request. The MIC is generated by deriving a nonce from the incremented *counter* of the respective request.
$\mathrm{MIC_b}$	This MIC is used by the border node *b* to authenticate all fields. The MIC is generated using $\{\mathrm{Key_{b,u}}\}_{\mathrm{Ku}}$ and $counter + 1$ as nonce.

A. Node

For implementing the node-related parts, we extended the AKES implementation that was integrated into the Contiki-NG operating system [1]. Specifically, we added an *AKES revocation* module, which receives, processes, and sends *distribution requests* and *distribution replies*. Both *distribution requests* and *distribution replies* are sent as link layer command frames.

B. Border Node

For implementing the border node-related parts, we reused our *AKES revocation* module and, additionally, made use of Contiki-NG's CoAP implementation. More specifically, the communication with the base station is implemented as follows. When AKES starts, we additionally start a CoAP server, which receives and processes *eviction requests* at the resource *akes/evict*. While processing an *eviction request*, we store relevant data in two structures. (i) For source routing we use a linked list to store routes. Each element of this list stores the link layer address of a discovered node along with a reference to the node which discovered that node first. (ii) The progress of a single *eviction request* is stored in a custom structure. This structure contains all necessary data for creating an *eviction reply*.

C. Base Station

As described in Section III the base station is not part of the network but rather a distinct device or even a virtual machine. It is implemented with Python and *aiocoap* as CoAP library. The base station consists of 2 modules. (i) The *NodeStore* stores the current state of the network consisting of a list of nodes in the network and the border nodes. (ii) The *EvictionProcess* is a singleton, since multiple concurrent evictions are not supported and could lead to deadlocks. Its responsibility is to control the node eviction by sending *eviction requests* to border nodes. It has three distinct caches for storing (i) already reached nodes, (ii) nodes that were not yet notified, and (iii) nodes that are currently being notified. To try reaching a node over a different border node if the previous tries were not successful, the latter two caches save additionally a border node for each entry. Once a node confirmed the eviction it gets added to the reached nodes cache and removed from all other caches including entries from other border nodes. Both modules are implemented in a thread safe manner. A separate CoAP server is started to receive the *eviction replies*. The server parses the messages and hands them over to the *EvictionProcess* in a new thread.

V. Evaluation

In this section, we will first motivate the experiments we conducted, then explain our experimental setup, and finally discuss the results.

A. Motivation

The duration of an eviction is the first characteristic we will explore. After a compromised node is detected, it is crucial to quickly exclude this node from any further communication in order to avert further harm to the network. Hence, we measure the time that an eviction takes from initiation by the base station till all border nodes reported back to be finished to the base station. Throughout, we omit the time it takes until upper layers have adapted to the new topology as this heavily depends on the concrete protocol stack.

A second important characteristic for the applicability of protocols in the IoT is their energy consumption. Yet, the energy consumption greatly varies with the used MAC protocol. To make our measurements independent from the used MAC protocol, we measure the energy consumption indirectly. Specifically, we count the number of frames that are sent and received on a per-node basis. This choice is motivated by the fact that the energy consumption of an IoT device is typically dominated by the radio transceiver [17].

B. Methods

Using Contiki-NG's simulation environment *Cooja*, three scenarios were created with 25, 49, and 100 nodes. The nodes were arranged in a rectangular shape, as shown in Fig. 3. The border node was situated centered the west side of the network. As our approach additionally supports the use of multiple border nodes, a second border node was placed at the exact opposite position than the first border node in another version of each scenario. Furthermore, as the location of the evicted node influences the measurements, the evicted node was always the one in the upper left corner.

In our implementation, the maximum number of destinations per eviction request was set to 2. Extra code was inserted

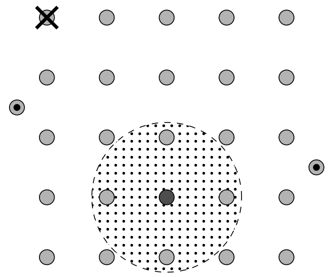
Fig. 3: Simulated network topology. ⊙ border nodes, ○ nodes, ✕ compromised node, ⠿ range of wireless signal of ●

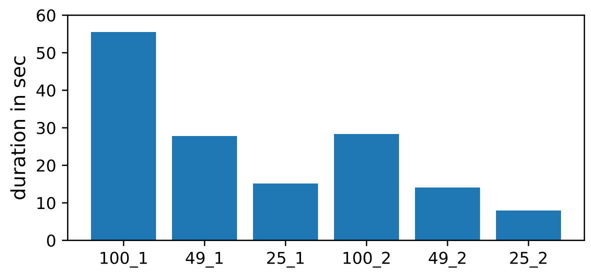

Fig. 4: Total duration of an eviction in a network with 25, 49, or 100 nodes with one or two border nodes

in order to count the number of sent frames. Retransmitted frames were not counted.

At the beginning of each measurement, the nodes had time to establish session keys with each other. Afterwards, the simulation speed was set to 100% and an eviction was started. During each measurement, the base station process was connected with the Cooja simulator via Unix sockets, yet the delays due to interprocess communication were neglected.

C. Results

As shown in Fig. 4, our protocol takes about 55.5 seconds to complete an eviction in the scenario with 100 nodes and a single border node. Reducing the node count by half reduces the duration for an eviction by about half, as well. If a second border node is used, the duration gets halved once more. This shows the high speed and good scalability of our protocol.

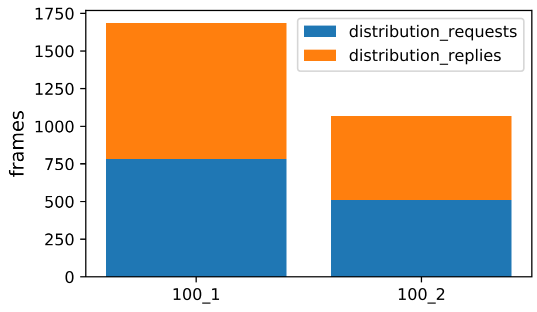

Fig. 5: Number of sent distribution frames

In the scenario with 100 nodes and a single border node, a total of 1685 frames was sent, split into 784 *distribution* *requests* and 901 *distribution replies*. By adding a second border node, we achieved a reduction in the number of sent frames by 619, as depicted in Fig. 5. In the scenario with two border nodes, 513 *distribution requests* and 553 *distribution replies* were sent. From an energy perspective, our protocol gets way more energy efficient when using multiple border nodes, as shown in Fig. 6. In any case, our protocol only involves unicast frames, which a MAC protocol typically conveys more energy efficiently than broadcast frames [18].

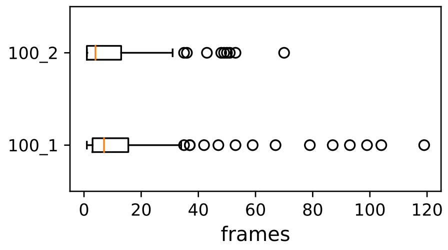

Fig. 6: Distribution of the number of sent distribution frames per node in the scenario with 100 nodes and one or two border nodes

VI. Conclusion and Future Work

A complete key management solution covers key establishment, key revocation, and rekeying. Currently, AKES only covers the establishment of session keys in IEEE 802.15.4 networks and lacks a key revocation and rekeying protocol. Thus, a node can only be evicted from an IEEE 802.15.4 network by redeploying the whole network. Moreover, all previously proposed protocols for key revocation and rekeying route messages via nodes that shall be evicted, although compromised nodes may deny to forward messages. By contrast, we have proposed a protocol for key revocation and rekeying, which does not route messages via evicted nodes. Furthermore, our protocol operates energy efficient, provides feedback, scales to big networks, and evicts quickly. Our protocol only incurs a small overhead in terms of traffic. Together with AKES, our protocol makes up a complete key management solution for IEEE 802.15.4 networks. For future work, two issues remain. First, long routes may exceed the maximum payload of link layer frames, which could be overcome by means of fragmentation. Second, after evicting a node, the employed routing protocol needs to update its routes. This recover time could be optimized.

References

[1] K.-F. Krentz, C. Meinel, and H. Graupner, "Denial-of-Sleep-Resilient Session Key Establishment for IEEE 802.15. 4 Security: From Adaptive to Responsive," 2018.

[2] D. Dinu and I. Kizhvatov, "EM analysis in the IoT context: lessons learned from an attack on Thread," 2018.

[3] R. Anderson and M. Kuhn, "Tamper resistance - a cautionary note," in *Proceedings of the Second USENIX Workshop on Electronic Commerce*, 1996, pp. 1–11.

[4] C.-Y. Chen and H.-C. Chao, "A survey of key distribution in wireless sensor networks," 2011.

[5] Y. W. Law and M. Palaniswami, "Key Management in Wireless Sensor Networks," in *GUIDE TO WIRELESS SENSOR NETWORKS*, 2009.

[6] D. Werner, "Key Revocation and Rekeying for the Adaptive Key Establishment Scheme," Hasso-Plattner-Institut, Universität Potsdam, Master's thesis, 2018.

[7] T. Moore, J. Clulow, S. Nagaraja, and R. Anderson, "New strategies for revocation in ad-hoc networks," in *European Workshop on Security in Ad-hoc and Sensor Networks*, 2007, pp. 232–246.

[8] P.-J. Chuang, S.-H. Chang, and C.-S. Lin, "A pkc-based node revocation scheme in wireless sensor networks," in *Future Generation Communication and Networking*, 2007, pp. 256–261.

[9] H. Chan, A. Perrig, and D. Song, "Random key predistribution schemes for sensor networks," in *Symposium on Security and Privacy*, 2003, pp. 197–213.

[10] Y. Wang, B. Ramamurthy, and X. Zou, "Keyrev: An efficient key revocation scheme for wireless sensor networks," in *IEEE International Conference on Communications*, 2007, pp. 1260–1265.

[11] G. Dini and I. M. Savino, "An efficient key revocation protocol for wireless sensor networks," in *Proceedings of the 2006 International Symposium on World of Wireless, Mobile and Multimedia Networks*, 2006, pp. 450–452.

[12] G. Jolly, M. C. Kusçu, P. Kokate, and M. Younis, "A low-energy key management protocol for wireless sensor networks," in *Proceedings of the Eighth IEEE International Symposium on Computers and Communications*, 2003, pp. 335–.

[13] S. Raza, L. Seitz, D. Sitenkov, and G. Selander, "S3k: Scalable security with symmetric keys–dtls key establishment for the internet of things," 2016.

[14] Z. Shelby, K. Hartke, and C. Bormann, "The constrained application protocol (coap)," RFC 7252, 2014.

[15] C. Bormann and Z. Shelby, "Block-wise transfers in the constrained application protocol (coap)," RFC 7959, 2016.

[16] C. Bormann, S. Lemay, H. Tschofenig, K. Hartke, B. Silverajan, and B. Raymor, "CoAP (Constrained Application Protocol) over TCP, TLS, and WebSockets," RFC 8323, 2018.

[17] 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*, 2007, pp. 28–32.

[18] A. Dunkels, "The ContikiMAC radio duty cycling protocol," Tech. Rep. T2011:13, 2011.