Doctoral Dissertation

A Denial-of-Sleep-Resilient Medium Access Control Layer for IEEE 802.15.4 Networks

Author:
Konrad-Felix Krentz, MSc

University:
University of Potsdam

Faculty:
Digital Engineering

Subject of Study:
IT-Systems Engineering

Date of Disputation:
November 27, 2019

Supervisor:
Prof. Christoph Meinel

Co-Supervisor:
Prof. Andreas Polze

External Examiners:
Prof. Thiemo Voigt
Prof. Peter Langendörfer

Published online at the
Institutional Repository of the University of Potsdam:
https://doi.org/10.25932/publishup-43930
https://nbn-resolving.org/urn:nbn:de:kobv:517-opus4-439301

Abstract

With the emergence of the Internet of things (IoT), plenty of battery-powered and energy-harvesting devices are being deployed to fulfill sensing and actuation tasks in a variety of application areas, such as smart homes, precision agriculture, smart cities, and industrial automation. In this context, a critical issue is that of denial-of-sleep attacks. Such attacks temporarily or permanently deprive battery-powered, energy-harvesting, or otherwise energy-constrained devices of entering energy-saving sleep modes, thereby draining their charge. At the very least, a successful denial-of-sleep attack causes a long outage of the victim device. Moreover, to put battery-powered devices back into operation, their batteries have to be replaced. This is tedious and may even be infeasible, e.g., if a battery-powered device is deployed at an inaccessible location. While the research community came up with numerous defenses against denial-of-sleep attacks, most present-day IoT protocols include no denial-of-sleep defenses at all, presumably due to a lack of awareness and unsolved integration problems. After all, despite there are many denial-of-sleep defenses, effective defenses against certain kinds of denial-of-sleep attacks are yet to be found.

The overall contribution of this dissertation is to propose a denial-of-sleep-resilient medium access control (MAC) layer for IoT devices that communicate over IEEE 802.15.4 links. Internally, our MAC layer comprises two main components. The first main component is a denial-of-sleep-resilient protocol for establishing session keys among neighboring IEEE 802.15.4 nodes. The established session keys serve the dual purpose of implementing (i) basic wireless security and (ii) complementary denial-of-sleep defenses that belong to the second main component. The second main component is a denial-of-sleep-resilient MAC protocol. Notably, this MAC protocol not only incorporates novel denial-of-sleep defenses, but also state-of-the-art mechanisms for achieving low energy consumption, high throughput, and high delivery ratios. Altogether, our MAC layer resists, or at least greatly mitigates, all denial-of-sleep attacks against it we are aware of. Furthermore, our MAC layer is self-contained and thus can act as a drop-in replacement for IEEE 802.15.4-compliant MAC layers. In fact, we implemented our MAC layer in the Contiki-NG operating system, where it seamlessly integrates into an existing protocol stack.

Zusammenfassung

Mit dem Aufkommen des Internets der Dinge (IoT), werden immer mehr batteriebetriebene und energieerntende Geräte in diversen Anwendungsbereichen eingesetzt, etwa in der Heimautomatisierung, Präzisionslandwirtschaft, Industrieautomatisierung oder intelligenten Stadt. In diesem Kontext stellen sogenannte Denial-of-Sleep-Angriffe eine immer kritischer werdende Bedrohung dar. Solche Angriffe halten batteriebetriebene, energieerntende oder anderweitig energiebeschränkte Geräte zeitweise oder chronisch ab, in energiesparende Schlafmodi überzugehen. Erfolgreiche Denial-of-Sleep-Angriffe führen zumindest zu einer langen Ausfallzeit der betroffenen Geräte. Um betroffene batteriebetriebene Geräte wieder in Betrieb zu nehmen, müssen zudem deren Batterien gewechselt werden. Dies ist mühsam oder eventuell sogar unmöglich, z.B. wenn solche Geräte an unzugänglichen Orten installiert sind. Obwohl die Forschungsgemeinschaft bereits viele Denial-of-Sleep-Abwehrmechanismen vorgeschlagen hat, besitzen die meisten aktuellen IoT-Protokolle überhaupt keine Denial-of-Sleep-Abwehrmechanismen. Dies kann zum einen daran liegen, dass man des Problems noch nicht gewahr ist, aber zum anderen auch daran, dass viele Integrationsfragen bislang ungeklärt sind. Des Weiteren existieren bisher sowieso noch keine effektiven Abwehrmechanismen gegen bestimmte Denial-of-Sleep-Angriffe.

Der Hauptbeitrag dieser Dissertation ist die Entwicklung einer Denial-of-Sleep-resilienten Mediumzugriffsschicht für IoT-Geräte, die via IEEE-802.15.4-Funkverbindungen kommunizieren. Die entwickelte Mediumzugriffsschicht besitzt zwei Hauptkomponenten. Die erste Hauptkomponente ist ein Denial-of-Sleep-resilientes Protokoll zur Etablierung von Sitzungsschlüsseln zwischen benachbarten IEEE-802.15.4-Knoten. Diese Sitzungsschlüssel dienen einerseits der grundlegenden Absicherung des Funkverkehrs und andererseits der Implementierung zusätzlicher Denial-of-Sleep-Abwehrmechanismen in der zweiten Hauptkomponente. Die zweite Hauptkomponente ist ein Denial-of-Sleep-resilientes Mediumzugriffsprotokoll. Bemerkenswert an diesem Mediumzugriffsprotokoll ist, dass es nicht nur neuartige Denial-of-Sleep-Abwehrmechanismen enthält, sondern auch dem Stand der Technik entsprechende Mechanismen zur Verringerung des Energieverbrauchs, zur Steigerung des Durchsatzes sowie zur Erhöhung der Zuverlässigkeit. Zusammenfassend widersteht bzw. mildert unsere Denial-of-Sleep-resiliente Mediumzugriffsschicht alle uns bekannten Denial-of-Sleep-Angriffe, die gegen sie gefahren werden können. Außerdem kann unsere Denial-of-Sleep-resiliente Mediumzugriffsschicht ohne Weiteres an Stelle von IEEE-802.15.4-konformen Mediumzugriffsschichten eingesetzt werden. Dies zeigen wir durch die nahtlose Integration unserer Mediumzugriffsschicht in den Netzwerk-Stack des Betriebssystems Contiki-NG.

Acknowledgements

This doctoral research would not have been possible without the support of great people. First of all, I would like to thank my supervisor, Prof. Christoph Meinel, for his sustained and competent assistance throughout this doctoral research. Also, I would like to thank my co-supervisor, Prof. Andreas Polze, for his helpful mentorship, which contributed considerably to the successful completion of this doctoral research. For such a broad doctoral research project, it was important to involve additional experts, as well. Therefore, I would like to thank Prof. Thiemo Voigt and Prof. Peter Langendörfer for serving as external examiners. Their helpful advice transcended their role as external examiners. Additional thanks go to Prof. Matthias Wählisch for his valuable feedback. Last but not least, I would like to thank my family for backing me up selflessly.

List of My Publications

1. Konrad-Felix Krentz and Christoph Meinel. Denial-of-sleep defenses for IEEE 802.15.4 coordinated sampled listening (CSL). *Computer Networks*, 148(15):60–71, 2019.

2. Konrad-Felix Krentz, Christoph Meinel, and Hendrik Graupner. Denial-of-sleep-resilient session key establishment for 802.15.4 security: from adaptive to responsive. In *Proceedings of the International Conference on Embedded Wireless Systems and Networks (EWSN 2018)*. Junction, 2018.

3. Konrad-Felix Krentz, Christoph Meinel, and Hendrik Graupner. More lightweight, yet stronger 802.15.4 security through an intra-layer optimization. In *Proceedings of the 10th International Symposium on Foundations & Practice of Security (FPS 2017)*. Springer, 2017.

4. Konrad-Felix Krentz, Christoph Meinel, and Hendrik Graupner. Secure self-seeding with power-up SRAM states. In *Proceedings of the 22nd IEEE Symposium on Computers and Communications (ISCC 2017)*. IEEE, 2017.

5. Konrad-Felix Krentz, Christoph Meinel, and Hendrik Graupner. Countering three denial-of-sleep attacks on ContikiMAC. In *Proceedings of the International Conference on Embedded Wireless Systems and Networks (EWSN 2017)*, pages 108–119. Junction, 2017.

6. Konrad-Felix Krentz, Christoph Meinel, and Maxim Schnjakin. POTR: practical on-the-fly rejection of injected and replayed 802.15.4 frames. In *Proceedings of the International Conference on Availability, Reliability and Security (ARES 2016)*, pages 59–68. IEEE, 2016.

7. Konrad-Felix Krentz and Christoph Meinel. Handling reboots and mobility in 802.15.4 security. In *Proceedings of the 31st Annual Computer Security Applications Conference (ACSAC '15)*, pages 121–130. ACM, 2015.

8. Konrad-Felix Krentz and Gerhard Wunder. 6doku: towards secure over-the-air preloading of 6LoWPAN nodes using PHY key generation. In *Proceedings of the European Conference on Smart Objects, Systems and Technologies (Smart SysTech 2015)*. VDE, 2015.

9. Konrad-Felix Krentz and Gerhard Wunder. 6LoWPAN security: avoiding hidden wormholes using channel reciprocity. In *Proceedings of the 4th International Workshop on Trustworthy Embedded Devices (TrustED '14)*, pages 13–22. ACM, 2014.

10. Konrad-Felix Krentz, Hosnieh Rafiee, and Christoph 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.

11. Benedikt Bock, Jan-Tobias Matysik, Konrad-Felix Krentz, and Christoph Meinel. Link layer key revocation and rekeying for the adaptive key establishment scheme. In *Proceedings of the IEEE 5th World Forum on Internet of Things (WF-IoT)*. IEEE, 2019.

12. Felix Seidel, Konrad-Felix Krentz, and Christoph Meinel. Deep en-route filtering of constrained application protocol (CoAP) messages on 6LoWPAN border routers. In *Proceedings of the IEEE 5th World Forum on Internet of Things (WF-IoT)*. IEEE, 2019.

13. Klara Seitz, Sebastian Serth, Konrad-Felix Krentz, and Christoph Meinel. Demo: enabling en-route filtering for end-to-end encrypted CoAP messages. In *Proceedings of the 15th ACM Conference on Embedded Networked Sensor Systems (SenSys '17)*. ACM, 2017.

This dissertation subsumes my publications 1 till 7.

Contents

Chapter 1

Introduction

Broadly defined, the *Internet of things (IoT)* is the trend of interconnecting embedded devices with the Internet. This trend is driven by fascinating applications in areas such as smart homes, precision agriculture, smart cities, and industrial automation. Behind the scenes, most IoT applications involve two kinds of embedded devices. IoT devices, on the one hand, fulfill sensing and actuation tasks. Typically, IoT devices have low processing capabilities, communicate via radio, and run on batteries or do energy harvesting. IoT gateways, on the other hand, provide IoT devices with connectivity to the Internet. Usually, IoT gateways are mains powered and have capable microprocessors.

An example hardware platform for IoT devices is the CC2538 system on chip (SoC) [14]. The CC2538 SoC includes an ARM Cortex M3 microprocessor, up to 512KB of program memory, up to 32KB of static random-access memory (SRAM), and an IEEE 802.15.4 transceiver. Furthermore, the CC2538 SoC offers 3 low-power modes (LPMs), among which the least energy-consuming one is LPM 3, as shown in Figure 1.1. Yet, switching to LPM 3 will not be always possible if no tasks are pending. This is because the deeper the sleep mode, the longer it takes to wake up again. Besides, waking up from LPM 3 requires an external interrupt. In comparison to the LPMs, significantly more current is drawn if the ARM Cortex M3 microprocessor is active. Moreover, the most energy-consuming modes of the CC2538 SoC are receiving and transmitting. In receive mode, the CC2538 SoC draws between 20 and 24mA, depending on the strength of the input signal. In transmit mode, the CC2538 SoC consumes between 24 and 34mA, depending on the configured transmission power.

Though the CC2538 SoC is only an example hardware platform for IoT devices, the proportions of the energy consumption in low-power, active, receive, and transmit modes are consistent across diverse hardware platforms [15]. That is, receiving and transmitting constitute the most energy-consuming modes on most IoT devices, followed by the active mode [15]. This has far-reaching implications to the design of software, protocols, and security for IoT devices.

As for software, it is essential to switch to the deepest LPM possible between processing, radio, and I/O tasks. Otherwise, the charge of a battery-powered or energy-harvesting IoT device will be depleted quickly. Consider, e.g., a CC2538-based IoT device that is powered by two AA batteries, each of which has a charge of 2600mAh. If this device never enters an LPM, but busy-waits at 32MHz, its batteries will be depleted after $\frac{2\times 2600\text{mAh}}{13\text{mA}} = 16.67\text{d}$ at the latest.

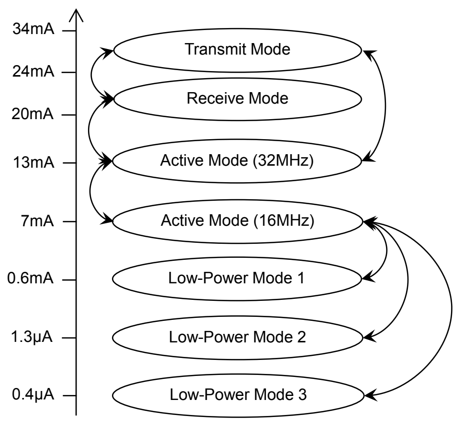


Figure 1.1: Current draw of the CC2538 SoC in various operational modes

Conversely, if this device uses LPM 2, the batteries' lifetime will be upper-bounded by $\frac{2 \times 2600\text{mAh}}{1.3\mu A} = 456.6\text{a}$. Of course, this is only a rough illustrative estimate since it neglects the batteries' self-discharge, the time spend in more energy-consuming modes, the energy consumption of peripherals, as well as the influences of temperatures on the batteries' discharge behavior.

As for protocols, any reduction in traffic may pay off twice, once at the sender side and again at the receiver side. At the sender side, the less data is transmitted, the less time may need to be spent in the energy-consuming transmit mode. At the receiver side, the less data is received, the less time may need to be spent in the energy-consuming receive mode. Thus, communication protocols for IoT devices should minimize their traffic. However, the amount of time spent in transmit and receive mode during transmissions and receptions ultimately depends on the employed medium access control (MAC) protocol since it is in charge of enabling the transceiver and putting it in transmit and receive mode. Hence, MAC protocols for IoT devices should operate in a way that minimizes the time spent in transmit and receive modes.

As for security, the energy constraints of IoT devices limit the solution space in two ways. On the one hand, it is also advisable to minimize the communication overhead due to security. A counterexample is to use a key distribution center (KDC)-based protocol for establishing session keys among neighboring IoT devices of a wireless mesh network [16, 17, 18]. In this case, each time a session key is established, one party has to reach out to the KDC, thereby requiring every IoT device along the path to expend energy for receiving, processing, and forwarding. On the other hand, it is also advisable to avoid processing-intensive tasks, such as public-key cryptography. In fact, despite software optimizations and hardware acceleration, public-key cryptography continues to be a processing-intensive task for IoT devices [17, 19, 20]. For example, a hardware-accelerated Elliptic Curve Digital Signature Algorithm (ECDSA) signature generation and verification, takes 340ms and 521ms on the CC2538 SoC, respectively [21].

1.1 Research Scope

Much effort already went into designing and implementing energy-efficient software, protocols, and security for IoT devices, yet mostly without integrating any defenses against so-called *denial-of-sleep attacks*. A denial-of-sleep attack temporarily or permanently deprives a battery-powered, energy-harvesting, or otherwise energy-constrained device of entering an LPM, thereby draining its charge [22, 23]. In the context of IoT, denial-of-sleep attacks may result in long outages of energy-constrained IoT devices. This risk is unacceptable in safety critical IoT applications, and, at least, undesirable in safety uncritical IoT applications for reasons such as customer satisfaction, quality-of-service guarantees, and reliability requirements.

IoT devices are by far not the only devices that are susceptible to denial-of-sleep attacks. Potentially, any energy-constrained device is susceptible to denial-of-sleep attacks, including smartphones and implanted devices [24, 25, 26]. Besides, there exist attacks similar to denial-of-sleep attacks against cloud infrastructures, too [27, 28]. In that regard, an attacker's goal is to cause a general increase in energy consumption or to provoke energy consumption peaks that overload a cloud infrastructure [27, 28]. This dissertation particularly focuses on protecting IoT devices against denial-of-sleep attacks.

Additionally, we narrow the scope of this dissertation to the IEEE 802.15.4 radio technology for three reasons [29]. First, radio technologies constitute common targets of denial-of-sleep attacks because radio technologies usually comprise a MAC protocol, which enables and disables the energy-consuming receive and transmit modes. Second, among the multitude of radio technologies that lend themselves to low-power IoT applications, IEEE 802.15.4 has become a main choice of practitioners. This success can partly be attributed to the availability of complete protocol stacks on top of IEEE 802.15.4 and partly to the versatility of this radio technology. In fact, IEEE 802.15.4 supports both star and mesh topologies, sub-GHz and 2.4-GHz frequency bands, as well as various MAC protocols. Third, as IEEE 802.15.4 standardizes not only a single MAC protocol, IEEE 802.15.4 transceivers often provide generic interfaces. This spawned a lot of research on further custom-built MAC protocols for IEEE 802.15.4 networks and facilitates the prototyping of security enhancements.

1.2 Problem Statement

There are two functions in IEEE 802.15.4 that may be subjected to denial-of-sleep attacks, namely session key establishment and medium access control.

1.2.1 Denial-of-Sleep-Resilient Session Key Establishment

To protect itself, as well as upper-layer protocols, IEEE 802.15.4 optionally filters out inauthentic and replayed radio frames, as well as encrypts parts of radio frames. Theoretically, these security services of IEEE 802.15.4, collectively called *IEEE 802.15.4 security*, work without establishing session keys among neighboring IEEE 802.15.4 nodes, but it is generally impractical to do so for subtle reasons we discuss in Chapter 4. Nevertheless, IEEE 802.15.4 itself specifies no protocol for establishing session keys among neighboring IEEE 802.15.4

nodes. Prior work filled this gap already [10, 16, 20, 30, 31, 32, 33, 34, 35].

However, all existing proposals on how to establish session keys among neighboring IEEE 802.15.4 nodes leave the issue of denial-of-sleep attacks widely unaddressed. Essentially, there are two possible avenues for denial-of-sleep attacks against protocols for establishing session keys among neighboring IEEE 802.15.4 nodes. First, in a so-called *HELLO flood attack*, an attacker injects a request for session key establishment with a high transmission power, which may trigger energy-consuming processing or communication on all receivers [35, 36]. Second, in what we introduce as a *yo-yo attack* in Chapter 4, an attacker makes links temporarily available or temporarily unavailable so as to provoke more attempts to establish session keys, reestablishments of session keys, or both. Thus far, there are - to the best of our knowledge - only limited attempts to protect against HELLO flood attacks [10, 34, 35], and no defenses against yo-yo attacks.

1.2.2 Denial-of-Sleep-Resilient Medium Access Control

Medium access control is also target of various denial-of-sleep attacks [22, 37, 38]. A basic distinction of denial-of-sleep attacks against MAC protocols we suggest is between *reception-oriented* and *transmission-oriented* ones. While reception-oriented denial-of-sleep attacks mainly cause victim devices to stay longer in the energy-consuming receive mode, transmission-oriented denial-of-sleep attacks mainly cause victim devices to stay longer in the energy-consuming transmit mode. For example, an often applicable reception-oriented denial-of-sleep attack is to send plenty of radio frames to a victim device [22]. This usually causes the victim device's MAC protocol to fully receive these radio frames before they are further validated. On the other hand, an often applicable transmission-oriented denial-of-sleep attack is, e.g., to jam radio transmissions of a victim device, which typically causes its MAC protocol to retransmit [38].

Many proposals on how to protect MAC protocols against denial-of-sleep attacks were made [39, 40, 41, 42, 43, 44]. However, a common hurdle to the adoption of these proposals is that they either assume by now superseded MAC protocols [39, 40, 41], or radio chips with features different from commodity IEEE 802.15.4 transceivers [42, 43, 44]. Moreover, prior proposals in the area of denial-of-sleep-resilient medium access control stay at a conceptual level, thus leaving many integration and implementation issues unaddressed. After all, the effectiveness of most existing denial-of-sleep defenses for MAC protocols is limited and there do not even exist any proposals on how to protect MAC protocols against certain denial-of-sleep attacks.

1.2.3 Basic Issues

Besides protecting session key establishment and medium access control against denial-of-sleep attacks, there are additional basic issues on the way to a fully functional denial-of-sleep-resilient MAC layer for IEEE 802.15.4 networks.

1.2.3.1 Generating Cryptographic Random Numbers

In order to generate session keys, IEEE 802.15.4 nodes typically have to generate *cryptographic random numbers* [10, 20, 30, 31, 33, 34, 35], i.e., random numbers that appear uniformly distributed to a computationally-bounded attacker [45].

The generation of cryptographic random numbers itself is unproblematic since a secret truly random bit string, called *seed*, can efficiently be expanded into a stream of cryptographic random numbers [46, 47]. However, the generation of a seed in the first place poses a challenge in our context since IEEE 802.15.4 nodes often lack common sources of randomness, such as user interaction. A promising replacement is to use power-up SRAM states, which are partly random due to manufacturing variations [48]. Thus far, there, however, seems to be no method for extracting seeds from power-up SRAM states in a practical and information-theoretically secure manner. Moreover, the randomness of power-up SRAM states is susceptible to various factors [49, 50, 51, 52, 53].

1.2.3.2 Adapting to Topology Changes

Owing to mobile nodes and changing surroundings, new IEEE 802.15.4 links may become available over time. To take advantage of these links, session keys should not only be established among neighboring IEEE 802.15.4 nodes at start up, but also at runtime. Nevertheless, most current protocols for establishing session keys among neighboring IEEE 802.15.4 nodes only perform neighbor discovery at start up [10, 16, 20, 30, 31, 34, 35]. The only exception appears to be Ilia et al.'s protocol, where session key establishment is triggered if the upper layer sends a radio frame to a node with whom no session keys were established, yet [33]. Yet, Ilia et al. do not specify what happens if the upper layer broadcasts a radio frame, leaving their solution incomplete.

1.2.3.3 Providing Strong Freshness

Generally, the replay protection of IEEE 802.15.4 security only provides *sequential freshness*, i.e., merely ascertains that no radio frame is accepted more than once [54]. By contrast, *strong freshness* is provided if the receiver of a radio frame can additionally ensure that the radio frame was sent within a limited time span prior to its reception [54]. Strong freshness is desirable in IEEE 802.15.4 networks since delaying data or control traffic may cause unforeseen issues. Consider, e.g., an IEEE 802.15.4-based remote control for a garage door. If a user tries to open his garage door via the remote control, but it does not open, the user may give up and park outside. However, the garage door may not open due to a jamming attack by an attacker. Moreover, sequential freshness does not prevent the attacker from opening the garage door by replaying the radio frames that were missed by the garage door later on.

1.2.3.4 Preventing Acknowledgment Spoofing

Finally, a subtle basic issue relates to so-called acknowledgment frames. Basically, acknowledgment frames are special radio frames that acknowledge the successful reception of a radio frame. In earlier versions of IEEE 802.15.4 [55, 56, 57], acknowledgment frames were always sent unauthenticated, which is why no reliance could be placed on whether a radio frame was actually received when receiving an acknowledgment frame in response [36]. Hence, when such reliance was required, it was necessary to implement acknowledgments in upper layers [58]. This issue was addressed in the 2015 version of IEEE 802.15.4 by optionally securing acknowledgment frames like any other radio frames [29].

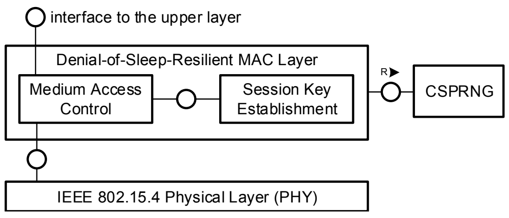

interface to the upper layer
Denial-of-Sleep-Resilient MAC Layer
Medium Access Control
Session Key Establishment
R
CSPRNG
IEEE 802.15.4 Physical Layer (PHY)

Figure 1.2: Main components of our denial-of-sleep-resilient MAC layer

Despite this fix, we will point out in Chapter 5 that even when a node receives a fresh authentic acknowledgment frame, the node can still not be sure whether the corresponding radio frame was successfully received.

1.3 Contributions

The overall contribution of this dissertation is to propose a fully functional denial-of-sleep-resilient MAC layer for IEEE 802.15.4 networks. As shown in Figure 1.2, our denial-of-sleep-resilient MAC layer incorporates a denial-of-sleep-resilient protocol for establishing session keys among neighboring IEEE 802.15.4 nodes and a denial-of-sleep-resilient MAC protocol. The development of both of these components led to various contributions to the areas of session key establishment among neighboring IEEE 802.15.4 nodes and MAC security:

- As for the establishment of session keys among neighboring IEEE 802.15.4 nodes, we propose the Adaptive Key Establishment Scheme (AKES). AKES stands out in that it resists both `HELLO` flood and yo-yo attacks. This is achieved by tailoring the intuitive concept of leaky bucket counters (LBCs) to the prevention of these denial-of-sleep attacks [59].
- Concerning MAC security, our first contribution is to review the four - according to our impression - most prevalent MAC protocols for IEEE 802.15.4 networks regarding their denial-of-sleep resilience, namely ContikiMAC [60], coordinated sampled listening (CSL) [29], time-slotted channel hopping (TSCH) [29], and low-power wireless bus (LWB) [61]. It turned out that ContikiMAC and CSL are more denial-of-sleep-resilient than TSCH and LWB, yet both ContikiMAC and CSL have denial-of-sleep vulnerabilities that need fixing. Hence, as a second contribution to the area of MAC security, we propose denial-of-sleep-protected versions of both ContikiMAC and CSL. Having implemented and evaluated these denial-of-sleep-protected versions of ContikiMAC and CSL, we found our denial-of-sleep-protected version of CSL to be advantageous not only in terms of denial-of-sleep resilience, but also in terms of reliability. Lastly, a third contribution to the area of MAC security is the identification of design patterns of denial-of-sleep defenses, some of which emerged from our literature review, others of which are newly introduced by us.

Beyond that, we also overcome all of the aforementioned basic issues:

- Regarding the generation of cryptographic random numbers, our first main contribution is to propose a superior method for extracting seeds from power-up SRAM states. Unlike current methods, ours is both information-theoretically secure and practical. Additionally, many factors threatening the randomness of power-up SRAM states do not apply when our method is used. As a second main contribution in this regard, we propose the design of a cryptographically-secure pseudo-random number generator (CSPRNG), which seeds itself with power-up SRAM states using our method, as well as with radio noise for additional security. As shown in Figure 1.2, this CSPRNG is separate from our denial-of-sleep-resilient MAC layer and thus can be shared with upper layers.

- To adapt to topology changes, AKES continuously searches new neighbors, on the one hand, and deletes inactive neighbors on the other hand. Furthermore, for saving energy, AKES self-adaptively decreases its efforts to find new neighbors when the network topology stabilizes, and increases its efforts to find new neighbors when the network topology destabilizes.
- Our denial-of-sleep-resilient MAC layer provides strong freshness by integrating the newly devised concept of *wake-up counters* into IEEE 802.15.4 security. As a result, our denial-of-sleep-resilient MAC layer not only filters out inauthentic and replayed radio frames, but also radio frames that were delayed by more than a configurable period of time.
- Finally, when using our denial-of-sleep-resilient MAC layer, reliance can be placed on that a radio frame was successfully received when getting a fresh authentic acknowledgment frame in response.

Altogether, this dissertation thus comes up with a fully functional denial-of-sleep-resilient MAC layer for IEEE 802.15.4 networks. Our implementation of this MAC layer is readily available as an add-on to the Contiki-NG operating system at `https://github.com/kkrentz/contiki-ng`.

1.4 Organization

The rest of this dissertation is organized as follows:

Chapter 2 introduces relevant technologies. We will first contrast the IEEE 802.15.4 radio technology with two radio technologies for low-power wide-area networks (LPWANs) and give insight when IEEE 802.15.4 is favorable for implementing low-power IoT applications. Then, we present the design of the - according to our impression - four most prevalent MAC protocols for IEEE 802.15.4 networks, namely ContikiMAC, CSL, TSCH, and LWB. These MAC protocols will undergo a review regarding their denial-of-sleep resilience later in Chapter 5. Next, we outline widely employed upper-layer protocols for IEEE 802.15.4 networks and highlight their dependence on IEEE 802.15.4 security. Lastly, we sum up the main concepts of two mainstream operating systems for IoT devices, among which we opted for Contiki-NG for prototyping our denial-of-sleep-resilient MAC layer.

Chapter 3 presents our CSPRNG, which seeds itself with power-up SRAM states and radio noise. To begin with, we motivate the need for cryptographic random numbers and argue that IoT devices should be capable of generating cryptographic random numbers entirely by themselves. Furthermore, we point out limitations of current methods for extracting seeds from power-up SRAM states. Then, we go into information-theoretic notions and survey related work. Next, we explain our superior method for extracting seeds from power-up SRAM states. Finally, we describe the design of our CSPRNG and give an experimental evaluation.

Chapter 4 details AKES - our denial-of-sleep-resilient protocol for establishing session keys among neighboring IEEE 802.15.4 nodes. At first, we make a case for session keys and point out limitations of current protocols for establishing session keys among neighboring IEEE 802.15.4 nodes. Then, we introduce related work on which AKES is based. Next, we detail AKES, outline denial-of-sleep attacks against AKES, and propose two sets of corresponding denial-of-sleep defenses. Further, we outline our implementation of AKES and quantify the overhead due to AKES. Also, we give a comparative evaluation of AKES' resilience to denial-of-sleep attacks when using our two different sets of denial-of-sleep defenses. Lastly, we wrap up with completing our discussion of related work.

Chapter 5 elaborates on the problem of denial-of-sleep-resilient medium access control. We begin with analyzing the susceptibility of ContikiMAC, CSL, TSCH, and LWB to denial-of-sleep attacks. Then, we sum up prior work on denial-of-sleep-resilient medium access control and highlight persisting research gaps. Eventually, we conclude that ContikiMAC and CSL are most promising to be developed into a denial-of-sleep-resilient MAC protocol, yet that several research gaps have to be filled in this endeavor.

Chapter 6 reports on our development steps toward a denial-of-sleep-protected version of ContikiMAC.

Chapter 7 presents our denial-of-sleep-protected version of CSL, which constitutes the final design of our denial-of-sleep-resilient MAC protocol. Our version of CSL resists, or at least greatly mitigates, all denial-of-sleep attacks against CSL we found. Beyond that, our version of CSL incorporates a channel hopping extension, as well as a throughput optimization.

Chapter 8 summarizes this dissertation, paraphrases follow-up master's theses, suggests topics for future research, and closes with general observations.

Chapter 2

Technologies for the Internet of Things

This chapter introduces relevant technologies. We will first contrast the IEEE 802.15.4 radio technology with two radio technologies for LPWANs and give insight when IEEE 802.15.4 is favorable for implementing low-power IoT applications. Then, we present the design of the - according to our impression - four most prevalent MAC protocols for IEEE 802.15.4 networks, namely ContikiMAC, CSL, TSCH, and LWB. These MAC protocols will undergo a review regarding their denial-of-sleep resilience later in Chapter 5. Next, we outline widely employed upper-layer protocols for IEEE 802.15.4 networks and highlight their dependence on IEEE 802.15.4 security. Lastly, we sum up the main concepts of two mainstream operating systems for IoT devices, among which we opted for Contiki-NG for prototyping our denial-of-sleep-resilient MAC layer.

2.1 Radio Technologies

As shown in Table 2.1, several radio technologies lend themselves to low-power IoT applications. A basic distinction between these radio technologies is if they are designed for LPWANs or low-power wireless personal area networks (LoWPANs). Examples of radio technologies for LPWANs include Sigfox, long range wide area network (LoRaWAN), Weightless-P, and NarrowBand-IoT (NB-IoT), all of which feature communication ranges of a couple kilometers. Examples of radio technologies for LoWPANs are Z-Wave, IEEE 802.15.4, Bluetooth low energy (BLE), and enOcean, all of which achieve communication ranges of a few hundreds of meters at most. Besides communication range, there are many other criteria to consider when selecting a radio technology for a particular IoT application, such as hardware costs, operational costs, coverage, penetration, energy efficiency, delays, reliability, data rates, mesh support, and security properties. Despite all these choices and criteria to consider, two radio technologies are regularly employed in today's low-power IoT applications, namely IEEE 802.15.4 and LoRaWAN. Besides, NB-IoT is becoming increasingly popular.

Table 2.1: Radio Technologies for Low-Power IoT Applications

Radio technology	Target use case	Frequency band in Europe	Network topology	Data rate (in kbit/s)
Sigfox	LPWAN	868MHz SRD	star	0.1
LoRaWAN	LPWAN	868MHz SRD	star	0.3 - 50
Weightless-P	LPWAN	868MHz SRD	star	0.625 - 100
NB-IoT	LPWAN	LTE or GSM	star	20-250
Z-Wave	LoWPAN	868MHz SRD	mesh	9.6-100
IEEE 802.15.4 (O-QPSK PHY)	LoWPAN	868MHz SRD or 2.4GHz ISM	mesh	100 - 250
IEEE 802.15.4 (SUN OFDM PHY)	LoWPAN	868MHz SRD or 2.4GHz ISM	mesh	50 - 800
BLE	LoWPAN	2.4GHz ISM	star	270
enOcean	LoWPAN	868MHz SRD	star	125

2.1.1 IEEE 802.15.4

To start with, let us take a closer look at IEEE 802.15.4 [29].

2.1.1.1 Overview

IEEE 802.15.4 layers its functionality into a physical layer (PHY) and a MAC layer. The PHY provides raw transceiver functions, such as enabling the transceiver, disabling the transceiver, listening for radio frames, transmitting radio frames, configuring a channel, performing a clear channel assessment (CCA), etc. The MAC layer uses these raw transceiver functions to organize the medium access as per a MAC protocol. Additionally, the MAC layer offers security services. Upper layers can interface with the MAC layer so as to send and receive frames, as well as to configure the MAC layer.

2.1.1.2 Network Topology

Unlike most low-power radio technologies, IEEE 802.15.4 supports mesh topologies. This feature is valuable since mesh topologies (i) contain redundant paths that IoT devices and gateways can resort to and (ii) extend a network's coverage beyond the communication range of IoT gateways. However, as for mesh topologies, IEEE 802.15.4 merely standardizes basic concepts depicted in Figure 2.1. For example, IEEE 802.15.4 defines two classes of nodes, namely reduced-function devices (RFDs) and full-function devices (FFDs). Whereas an RFD can only be linked with a single FFD at a time, an FFD can be linked with multiple FFDs and RFDs in parallel. It is further standardized that an IEEE 802.15.4 network is clustered into one or more subnetworks, called personal area networks (PANs). Each PAN has exactly one FFD that acts as a PAN coordinator. The PAN coordinator has special responsibilities, such as choosing a PAN ID that differs from the PAN IDs of all co-located PANs. For many other aspects, in particular the formation of mesh topologies and the routing of packets, IEEE 802.15.4 declares an upper layer responsible. These tasks can, e.g., be fulfilled by IEEE 802.15.10 or the IPv6 routing protocol for low-power and lossy networks (RPL) [62, 63].

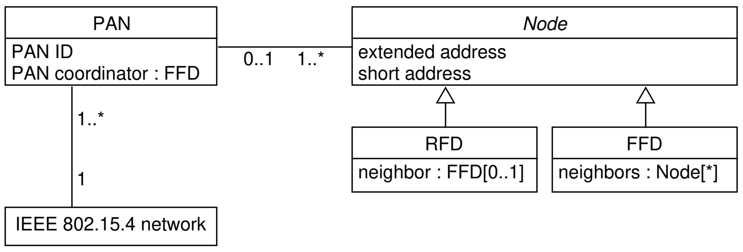


Figure 2.1: Organization of an IEEE 802.15.4 network

2.1.1.3 Frame Format

In order to separate PHY and MAC layer functionalities, both layers use their own headers and footers. When, e.g., using the offset quadrature phase-shift keying (O-QPSK) PHY, the PHY header is to be formatted like shown in Figure 2.2a. The O-QPSK PHY header consists of a synchronization header (SHR) and the Frame Length field. Some other PHYs also append a footer, such as the smart utility network (SUN) orthogonal frequency-division multiplexing (OFDM) PHY. At the MAC layer, the formatting of headers and footers depends on the type of frame being sent. For the scope of this dissertation, the following six frame types are relevant:

Beacon: FFDs may send beacon frames so as to announce the presence of an IEEE 802.15.4 network and to enable other nodes to join that IEEE 802.15.4 network. The format of beacon frames is shown in Figure 2.2b.

Data: Data frames carry upper-layer traffic. Their format also follows the general frame format shown in Figure 2.2b.

Command: In contrast to data frames, command frames are not passed on to the upper layer, but are processed at the MAC layer. Command frames are formatted like shown in Figure 2.2b.

Acknowledgment: The main purpose of acknowledgment frames is to acknowledge the successful reception of a frame. The format of acknowledgment frames is shown in Figure 2.2b.

Multipurpose: Multipurpose frames can take the place of data and command frames. Their format is a bit more concise, as shown in Figure 2.2c.

Extended: The extended frame type is a placeholder for new frame types. The format of extended frames is widely left open, as shown in Figure 2.2d.

Thus, most types of frames are formatted like shown in Figure 2.2b. Therein, the Frame Control field is a bitfield that, e.g., encodes a frame's type and which fields follow. Thereafter, in the case of non-acknowledgment frames, the optional Sequence Number field contains a rolling-over 1-byte counter of outgoing frames, allowing receivers to detect duplicates. Otherwise, in the case of acknowledgment frames, the optional Sequence Number field echoes the sequence number of the frame that is being acknowledged, allowing receivers to match acknowledgment frames with the frames that are being acknowledged. The addressing fields are self-explanatory, yet it is crucial to remark that IEEE 802.15.4 supports two kinds of addresses, namely 8-byte extended addresses that are globally unique and 2-byte short addresses that are unique within a PAN. Whereas extended addresses are burnt into IEEE 802.15.4 transceivers during manufacturing, short addresses are assigned at runtime. The Auxiliary Security Header field helps in unsecuring frames. Thereafter, so-called information elements (IEs) may carry metadata. IEs can be added and processed by both the MAC layer and the upper layer. The Payload field conveys the actual payload. Finally, the frame check sequence (FCS) field holds a checksum.

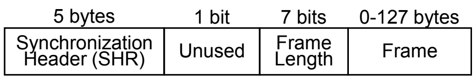
2.1.1.4 Security

The security services of the IEEE 802.15.4 MAC layer are often collectively referred to as *IEEE 802.15.4 security*. Essentially, IEEE 802.15.4 security filters out inauthentic and replayed frames, as well as optionally encrypts parts of frames. Below, we detail (i) how IEEE 802.15.4 security authenticates and encrypts data and (ii) how IEEE 802.15.4 security realizes replay protection.

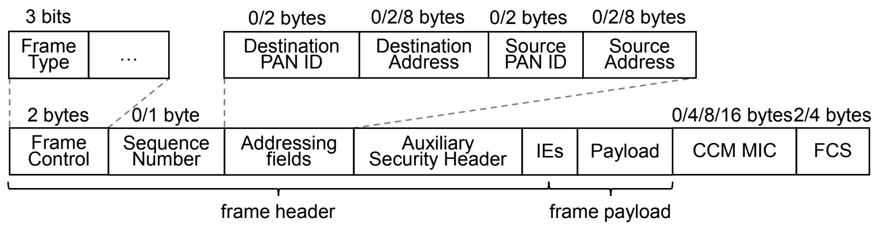
Authentication and Encryption For authentication and encryption, IEEE 802.15.4 security combines a restricted version of the block cipher mode CCM with the block cipher AES-128 [64, 65]. The inputs to CCM-AES-128 are data a to authenticate only, data m to authenticate and encrypt, a 13-byte nonce N, and a 128-bit symmetric key K. Given these inputs, CCM operates in two phases. First, CCM generates a message integrity code (MIC) over a and m via cipher block chaining (CBC). Second, CCM encrypts both m and the generated CBC MIC in counter mode (CTR). It is deemed that encrypting CBC MICs prevents collision attacks against CBC MICs, i.e., the misuse of occasions where different plain texts yield equal CBC MICs [64]. The outputs of CCM are the encrypted data and the encrypted CBC MIC, henceforth called *CCM MIC*.

In IEEE 802.15.4 security, the CCM inputs a and m are chosen in accordance with the selected security level. Table 2.2 lists available security levels and the resulting values for a and m. Basically, upper layers can either just authenticate an outgoing frame, or authenticate an outgoing frame and encrypt parts of it. Also, depending on the selected security level, CCM MICs are either truncated to 4, 8, or 16 bytes, thus enabling practitioners to trade off guessing attack resistance against per-frame overhead. The selected security level is sent along with frames as part of the Auxiliary Security Header field, as shown in Figure 2.2e. Receivers reject an incoming frame if its security level is too low.

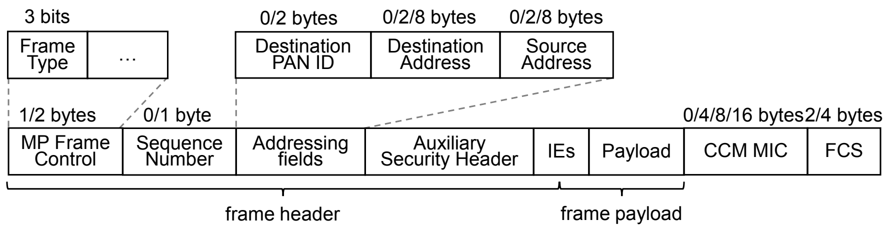
The CCM nonce N is generated by IEEE 802.15.4 security in one of two ways. First, if not using TSCH, N is derived from an incrementing frame counter and the sender's extended address. IEEE 802.15.4 security offers a choice between two kinds of frame counters, namely per-key and per-device frame counters. As their names suggest, per-key frame counters are maintained on a per-key basis, whereas per-device frame counters are not tied to a specific key. Regardless, of which kind of frame counter is used, the actual frame counter value from which N was derived is conveyed as part of the Auxiliary Security Header field. Second, if using TSCH, IEEE 802.15.4 security derives N from the sender's short or extended address, and the index of the current timeslot. More specifically, in TSCH, each transmission happens within a single timeslot, each of which has an



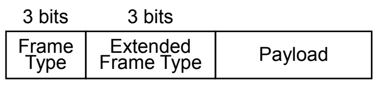
(a)



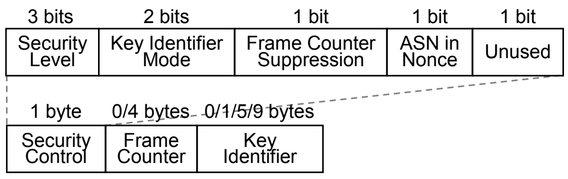
(b)



(c)



(d)



(e)

Figure 2.2: Format of (a) the O-QPSK PHY header, (b) beacon, data, acknowledgment, and command frames, (c) multipurpose frames, (d) extended frames, and (e) the Auxiliary Security Header field

Table 2.2: Security Levels of IEEE 802.15.4 Security

Security level	Length of CCM MIC (in byte)	Data a to authenticate only	Data m to authenticate and encrypt
1	4	frame header and frame payload (see Figure 2.2b and 2.2c)	empty
2	8		
3	16		
5	4	frame header and special fields of the frame payload (see Figure 2.2b and 2.2c)	frame payload except for certain fields therein
6	8		
7	16		

index assigned to it. Since both the sender and the receiver(s) are aware of the current timeslot's index, timeslot indices need not be sent along with frames.

The CCM key K, on the other hand, has to be provided by an upper layer. What IEEE 802.15.4 security standardizes, however, is a mechanism for designating which key was used to secure a frame. For this purpose, the Auxiliary Security Header field contains the Key Identifier field.

Replay Protection Depending on whether TSCH is used or not, IEEE 802.15.4 security realizes replay protection differently. If TSCH is not used, a received frame is considered replayed if and only if its frame counter is smaller or equal than that of the last authentic frame with this frame counter. If TSCH is used, replay protection is actually provided as a side effect of deriving CCM nonces from timeslot indices. This is because, if a frame is replayed in a later timeslot, a receiver will restore a wrong CCM nonce, causing the receiver to consider the CCM MIC of the replayed frame inauthentic. These two methods for realizing replay protection make a difference from a security perspective. Whereas the frame counter-based replay protection only provides sequential freshness, the timeslot index-based replay protection provides strong freshness.

2.1.2 LoRaWAN

The LoRaWAN specification not only spans PHY and MAC layer operations, but also the back end, as shown in Figure 2.3 [66]:

End-devices: LoRaWAN end-devices form a star network, i.e., directly connect to a nearby LoRaWAN gateway. The MAC protocol employed between LoRaWAN end-devices and LoRaWAN gateways is the LoRaWAN MAC protocol, which supports different trade-offs between energy efficiency and delays. Underneath the LoRaWAN MAC protocol, it is possible to either use the long range (LoRa) PHY or the frequency shift keying (FSK) PHY. Both these PHYs support trading off communication range against data rate and transmission power.

Gateways: LoRaWAN gateways relay traffic between the network server and LoRaWAN end-devices. Optionally, LoRaWAN gateways may broadcast beacons so as to support gateway-initiated transmissions to LoRaWAN end-devices within scheduled timeslots.

Network server: The main task of the network server is to relay application-layer traffic between LoRaWAN end-devices and their associated application servers. For this, the network server keeps track of the LoRaWAN gateway via which each LoRaWAN end-device of the LoRaWAN network is connected. Maybe surprisingly, the network server can also take care of tuning the LoRa and FSK PHY to minimize the energy consumption of LoRaWAN end-devices, as well as to minimize channel utilization. The tuning of the LoRaWAN MAC protocol, on the other hand, is up to LoRaWAN end-devices. Also, the network server implements security features, such as replay protection and authentication. The encryption of application-layer traffic, however, is done "end-to-end" between LoRaWAN end-devices and their associated application servers.

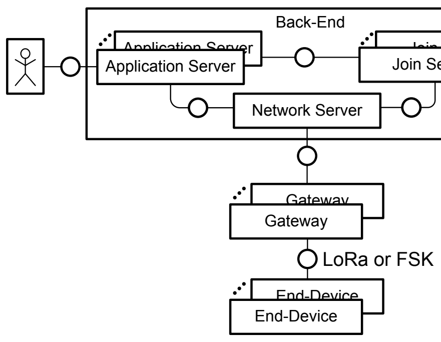


Figure 2.3: Interconnection of a LoRaWAN network with its back-end

Application servers: Application servers implement the actual applications.

Join server: When LoRaWAN end-devices join or rejoin a LoRaWAN network, join servers are potentially asked to assist the network server.

2.1.3 NB-IoT

An NB-IoT network is organized like shown in Figure 2.4 [67, 68]:

UEs: User equipments (UEs) take the role of IoT devices. UEs form a star topology, i.e., directly connect to a nearby IoT gateway, called *evolved node Bs (eNBs)* in the context of NB-IoT. The NB-IoT radio technology used between UEs and eNBs, is based on long term evolution (LTE). In comparison to LTE, NB-IoT requires less signaling overhead, implements medium access control more energy efficiently, achieves better indoor penetration, reduces the hardware costs for UEs, and offers a lower data rate.

eNBs: In order to support NB-IoT, LTE eNBs have to provide an additional air interface for NB-IoT, as well as to interact with the cellular IoT serving gateway node (C-SGN). Adding support for NB-IoT to an LTE eNB usually requires only a software update.

C-SGN: The C-SGN is a streamlined version of the core network of LTE. Its main responsibility is to route packets to their destination.

HSS: Each UE is equipped with a subscriber identity module (SIM) for mutual authentication with the home subscriber service (HSS). The HSS manages a database of subscriber-related information. It performs authentication and authorization, as well as keeps track of the locations of UEs.

SCEF: The service capability exposure function (SCEF) mainly relays non-IP and IP data between application servers and the C-SGN. Using non-IP data is preferable since this leads to less data being transmitted between UEs and eNBs, thereby reducing the energy consumption of UEs.

Application servers: Application servers provide the actual applications.

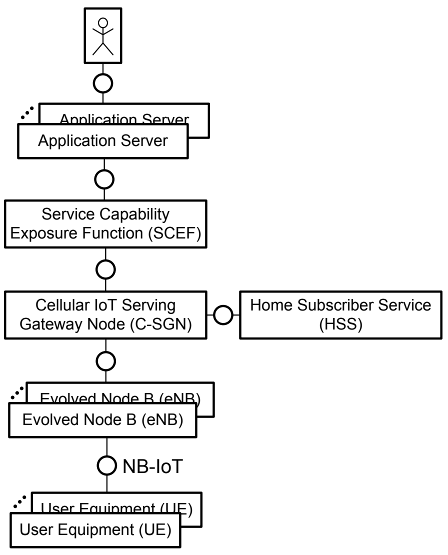


Figure 2.4: Interconnection of an NB-IoT network with application servers

2.1.4 Discussion

IEEE 802.15.4 excels at interconnecting dense deployments of IoT devices since IEEE 802.15.4 enables IoT devices to form mesh networks with redundant paths. Only to some extent, IEEE 802.15.4 can be tailored to networks of medium density by choosing PHYs and frequency bands that yield longer communication ranges, such as the SUN OFDM PHY and the short range device (SRD) band [69]. However, in the SRD band, the transmission time per hour is restricted in Europe [70]. This, in turn, may, e.g., hinder rolling out software updates to IoT devices wirelessly.

On the other hand, both LoRaWAN and NB-IoT are suitable for sparse deployments of IoT devices, given that LoRaWAN and NB-IoT enable IoT devices to directly connect to a distant IoT gateway. Doubts were, however, raised on the scalability of LoRaWAN to dense deployments [71]. This is because the probability of collisions increases dramatically with the number of LoRaWAN end-devices, even though concurrent transmissions only collide if they use the same spreading factor and channel. This issue is aggravated by the low data rate, long communication range, and use of possibly interfered unlicensed frequency bands of LoRaWAN. By contrast, NB-IoT is deemed to scale to dense deployments thanks to its use of licensed frequency bands, which are under the control of a mobile network operator [67].

That said, limitations of NB-IoT lie in its operational costs and fragmented coverage. To illustrate the operational costs, consider, e.g., a current offering by Deutsche Telekom called NB-IoT Access. NB-IoT Access costs €199 and includes 25 SIM cards, each of which may be used for 6 months and has a data volume of 500 kB. Apart from operational costs, the coverage of NB-IoT in the envisaged deployment area may be too low. By contrast, IEEE 802.15.4 and LoRaWAN enable practitioners to set up private networks at will.

Regarding security, a common limitation of IEEE 802.15.4, LoRaWAN, and

NB-IoT is their lack of defenses against denial-of-sleep attacks. As already indicated in the introduction, we suspect that overcoming this limitation is most reachable in the case of IEEE 802.15.4 due to the versatility of IEEE 802.15.4 transceivers. In fact, there exist prototypical implementations of denial-of-sleep defenses for IEEE 802.15.4 already [39, 72]. Furthermore, it is easier to deploy third-party security additions in IEEE 802.15.4 networks than in NB-IoT networks because NB-IoT networks must interoperate with a large number of heterogenous devices, whereas IEEE 802.15.4 networks are private networks that consist of fewer heterogenous devices if any. As for LoRaWAN, deploying third-party security additions seems also feasible, but, to the best of our knowledge, there is no implemented denial-of-sleep defense for LoRaWAN to date.

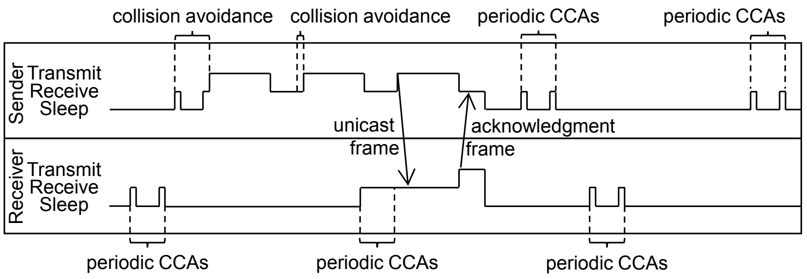
2.2 MAC Protocols for IEEE 802.15.4 Networks

In order to be suitable for low-power IoT applications, MAC protocols for IEEE 802.15.4 networks should minimize the time they spent in the energy-consuming transmit and receive modes. At the same time, MAC protocols for IEEE 802.15.4 networks should achieve high throughputs and low delays, as well as operate reliably. These partly contrasting requirements spawned a lot of research. By now, there are very many MAC protocols for IEEE 802.15.4 networks, which are often classified into *synchronous* (aka schedule-based) and *asynchronous* (aka contention-based) ones [73, 74]. While synchronous MAC protocols depend on network-wide or cluster-wide time synchronization, asynchronous MAC protocols do not. In the following, we present the design of the four - according to our impression - most prevalent MAC protocols for IEEE 802.15.4 networks, namely ContikiMAC, CSL, TSCH, and LWB.

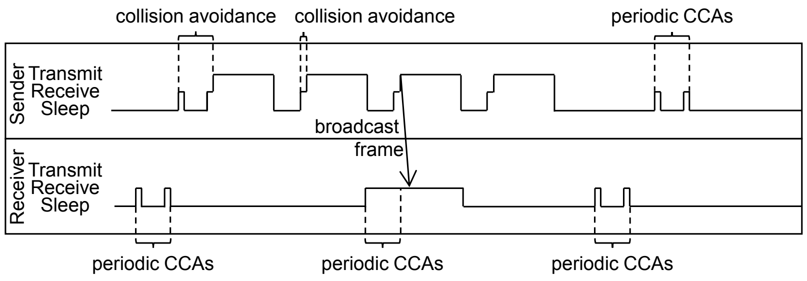
2.2.1 ContikiMAC

ContikiMAC is an unstandardized asynchronous MAC protocol, which, however, plays a key role in the literature [60]. This is because ContikiMAC integrates various ideas of earlier MAC protocols [75, 76, 77, 78], as well as acts as a baseline for several follow-up efforts [79, 80, 81, 82, 83, 84, 85, 86, 87]. The operation of ContikiMAC is shown in Figure 2.5. ContikiMAC regularly performs two CCAs. If any of these CCAs indicates a busy channel, ContikiMAC stays in receive mode to potentially receive a frame. Correspondingly, ContikiMAC repeatedly transmits an outgoing frame for a whole wake-up interval, plus once to cover corner cases. This behavior is often called *strobing*. As for unicast frames, ContikiMAC stops strobing prematurely if the intended receiver replies with an acknowledgment frame within one of the silence periods after single transmissions of unicast frames. If a unicast frame remains unacknowledged, ContikiMAC retries by strobing that unicast frame after a random back off period again, unless a maximum number of retransmissions is reached already.

Additionally, ContikiMAC comprises two optimizations. First, ContikiMAC's phase-lock optimization learns the wake-up times of neighboring nodes so as to start the strobing of a unicast frame right before the intended receiver wakes up. For this, the phase-lock optimization exploits that if an acknowledgment frame is received, the next to last strobed unicast frame must have been transmitted while the receiver woke up. Hence, the time when the transmis-



(a)



(b)

Figure 2.5: Operation of (a) a unicast and (b) a broadcast transmission in ContikiMAC. ContikiMAC not only uses CCAs to avoid collisions, but also to detect incoming transmissions.

sion of the next to last acknowledged unicast frame began can serve to estimate when the receiver will wake up next. Also as part of the phase-lock optimization, once the wake-up time of a receiver is learned, ContikiMAC no longer strobes unicast frames to that receiver for a whole wake-up interval plus once if no acknowledgment frame comes back, but only for a shorter time span plus once. Yet, to account for clock drift, ContikiMAC's phase-lock optimization relearns the wake-up time of a receiver if unicast transmissions to the receiver tend to fail. Second, ContikiMAC's fast-sleep optimization allows receivers to go back to sleep after they obtained a negative CCA at three occasions, namely (i) when radio noise lasts longer than for transmitting a maximum-length frame, (ii) when the silence period after the radio noise takes longer than ContikiMAC's inter-frame period, and (iii) when no frame is detected after the silence period.

2.2.2 CSL

CSL is a standardized asynchronous MAC protocol [29]. As shown in Figure 2.6, CSL wakes up periodically to shortly listen for a so-called *wake-up frame*. Wake-up frames are transmitted by CSL in a continuous sequence before transmitting an actual *payload frame*[1]. Each wake-up frame contains the time when the transmission of the payload frame begins. This hint, called *rendezvous time*,

[1]Instead of transmitting wake-up frames back-to-back, it is optionally possible to leave pauses between them. However, this would require prolonging the time in the energy-consuming receive mode during CSL's periodic wake ups. Hence, we focus on the case of sending wake-up frames back-to-back for the scope of this dissertation.

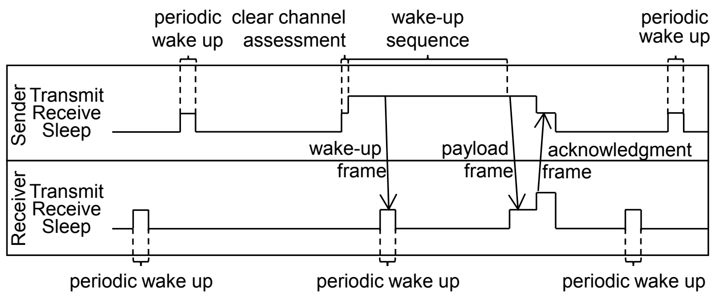


Figure 2.6: Operation of a unicast transmission in CSL

enables the receiver of a wake-up frame to temporarily go back to sleep until the transmission of the payload frame is about to begin. If the payload frame is a unicast frame, CSL expects an acknowledgment frame in response. The absence of an acknowledgment frame causes CSL to retransmit after a random back-off period, unless a maximum number of retransmissions is reached already.

In addition, CSL comprises two crucial optimizations. The first of them relates to the length of wake-up sequences. This length defaults to span a whole wake-up interval, but is shortened in what the IEEE 802.15.4 radio standard calls *synchronized transmissions*. As part of this optimization, CSL piggybacks its current phase, i.e., the time until its next wake up, on acknowledgment frames, as well as, optionally, on payload frames. This piggybacked data enables CSL to learn and keep track of the wake-up times of neighboring nodes. Later, when transmitting a unicast frame to a receiver whose wake-up time is known, CSL only transmits wake-up frames for the clock drift-based uncertainty about the receiver's expected wake-up time and starts this wake-up sequence right before the receiver may wake up at the earliest. The second optimization relates to CSL's throughput. If CSL has multiple outgoing payload frames for the same destination, CSL supports to transmit these frames in bursts. That is, without sending another wake-up sequence, CSL transmits such pending unicast or broadcast frames right after the last received acknowledgment or sent broadcast frame, respectively. Receivers are informed whether more payload frames follow through the Frame Pending flag of the Frame Control field.

2.2.3 TSCH

TSCH is a standardized synchronous MAC protocol for IEEE 802.15.4 networks [29], which was originally developed by Dust Networks Inc. [88]. The idea of TSCH is that each node of a TSCH network wakes up in certain timeslots so as to receive or transmit data on a certain channel as governed by a schedule. By doing channel hopping, TSCH alleviates fading effects, as well as interference [88]. Furthermore, TSCH operates energy efficiently since nodes can (i) enter an LPM during unused timeslots and (ii) quickly go back to sleep if no frame is received within a reception timeslot.

Keeping all nodes of a TSCH network tightly time synchronized is vital as, otherwise, transmissions may fail. In fact, receivers only stay in receive mode for `RxWait`=2.2ms per wake up by default when operating in the 2.4-GHz band. Hence, senders have to schedule frame transmissions so that, if a receiver is

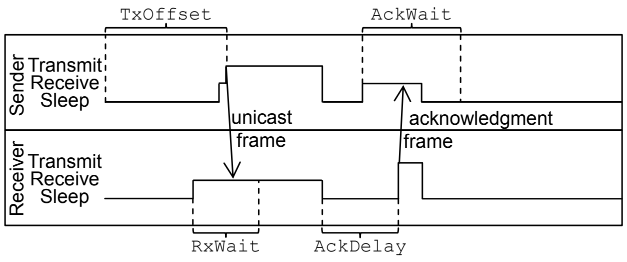


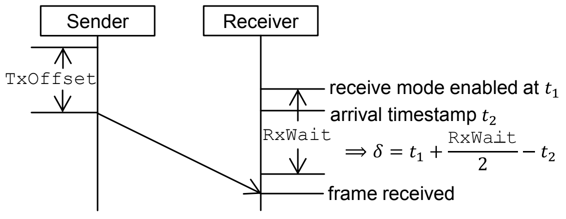
Figure 2.7: Operation of a unicast transmission in TSCH within a timeslot. The sender shortly enables the receive mode before transmitting to sense whether the channel is clear, thereby avoiding collisions with other transmissions.

perfectly synchronized, SHRs arrive $\frac{\texttt{RxWait}}{2}$ after enabling the receive mode, as shown in Figure 2.7. Consequently, the local time of a sender and a receiver must never be skewed by more than $\texttt{GuardTime} = \frac{\texttt{RxWait}}{2}$.

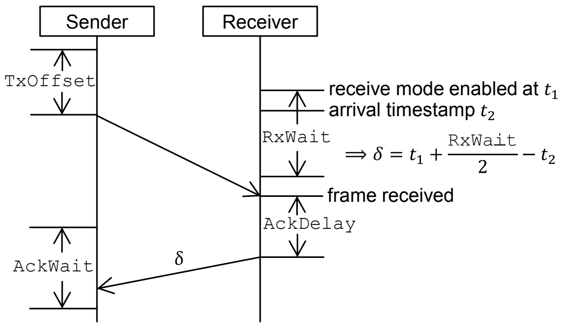
To attain tight time synchronization, TSCH specifies a couple of elementary mechanisms. Initially, to help unsynchronized nodes join a TSCH network, synchronized FFDs may periodically broadcast beacon frames that contain the index of the current timeslot, among others. Unsynchronized nodes shall listen for beacon frames for a limited time span. If, during this time span, an unsynchronized node has received an appropriate beacon frame, the unsynchronized node learns the current timeslot's index, as well as the time within the current timeslot by assuming that the beacon frame's SHR arrived `TxOffset` after the beginning of the timeslot. If, on the other hand, an unsynchronized node has received no appropriate beacon frame, it is typical to re-enter the receive mode to listen for beacon frames for a limited time span, and so on. Once synchronized, TSCH nodes shall resynchronize regularly because of clock drift. For doing so, TSCH specifies two further mechanisms, which are shown in Figure 2.8. In the frame-based resynchronization, a receiver resynchronizes with the local time of a sender by correcting the time offset δ between the expected reception time and the actual reception time. In the acknowledgment-based resynchronization, a sender resynchronizes with the local time of a receiver. The receiver measures the time offset δ upon reception of a unicast frame and reports back on δ to the sender by piggybacking δ on the corresponding acknowledgment frame.

However, while TSCH prescribes mechanisms for becoming and staying synchronized, it leaves the details of synchronizing a whole TSCH network open. Synchronizing a whole TSCH network requires organizing all nodes into a hierarchy so that time propagates without synchronization loops. To this end, the Internet engineering task force (IETF) working group 6TiSCH proposed reusing the routing topology of RPL [89].

Another unspecified aspect is the resynchronization rate. Originally, it was proposed to resynchronize at a preset rate that is calculated as $\frac{\texttt{GuardTime}}{\texttt{MaxDrift}}$, where `MaxDrift` is the maximum drift by which the employed clocks may drift apart [88]. For example, if $\texttt{GuardTime} = 1100\mu\text{s}$ and $\texttt{MaxDrift} = 30\text{ppm}$, nodes will resynchronize at $\frac{1}{36}$Hz. Yet, actually, resynchronizations have to be scheduled slightly more frequently to allow for retransmissions. Moreover, this calculation also neglects that uncertainties propagate and accumulate, which is why the more hops nodes are away from the time source, the more often they need


Sender
Receiver
TxOffset
receive mode enabled at t_1
arrival timestamp t_2
RxWait
$\Rightarrow \delta = t_1 + \frac{\text{RxWait}}{2} - t_2$
frame received

(a)



(b)

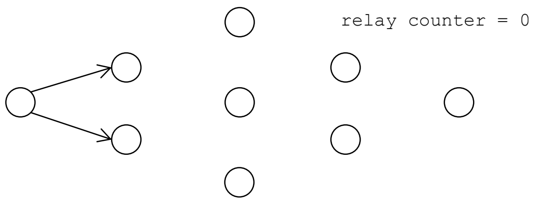
Figure 2.8: TSCH's (a) frame-based and (b) acknowledgment-based resynchronization

to resynchronize. To alleviate the accumulation of uncertainties, Chang et al. suggested coordinating resynchronizations, which, however, incurs a communication overhead [90]. A newer, currently favored proposal is to gradually reduce the resynchronization rate by getting better and better estimates of the clock drift [91, 92]. This greatly reduces the resynchronization rate over time, but does not solve the problem that uncertainties propagate and accumulate.

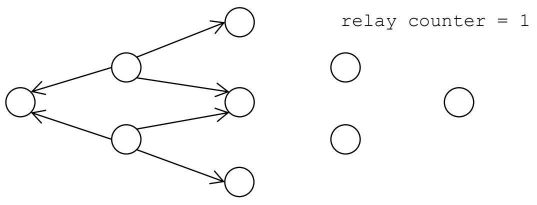
2.2.4 LWB

LWB comprises an unstandardized synchronous MAC protocol [61]. The main idea of LWB is to leverage a physical effect called *constructive interference*. Usually, interference is destructive in the sense that it prevents receivers from demodulating signals correctly. However, Ferrari et al. found that, when using the 2.4-GHz O-QPSK PHY of IEEE 802.15.4, receivers can correctly demodulate signals if all senders transmit the same signals almost simultaneously. According to their analysis and experimental results, transmissions interfere constructively as long as the temporal displacement between them is below 0.5μs.

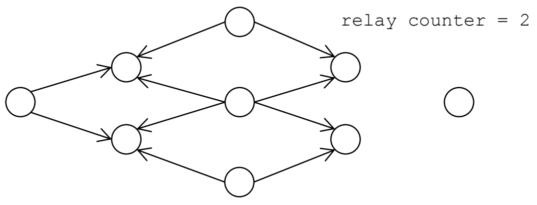
This finding gives rise to the flooding protocol Glossy [93], which is a building block of LWB. In Glossy, all nodes periodically and synchronously wake up for a period of time to listen for a Glossy-related frame. If a node receives a Glossy-related frame, the node retransmits that frame unmodified, except for the `relay counter` therein, which is incremented, as shown in Figure 2.9. The duration of the silence period between receptions and retransmissions is a network-wide constant so that retransmissions happen concurrently and hence interfere constructively. Once a node has retransmitted the same frame a certain number of times, it stops to participate in the flood. As the initiator of a



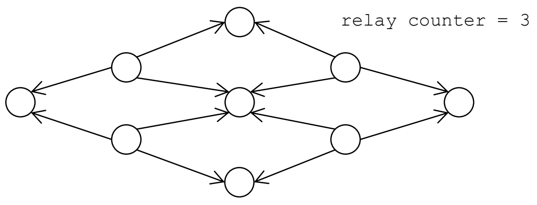
(a)



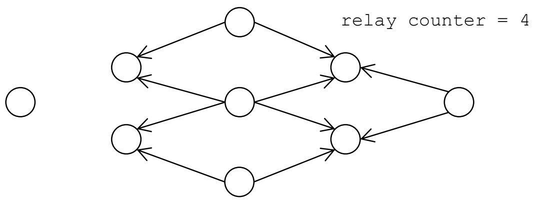
(b)



(c)



(d)



(e)

Figure 2.9: First steps of a Glossy flood

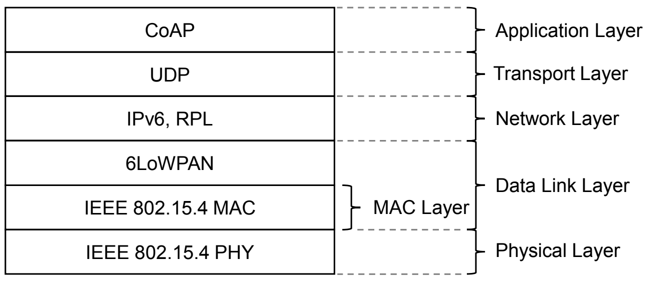


Figure 2.10: The 6LoWPAN protocol stack

Glossy flood embeds the flood's start time in the frame being flooded, receivers can synchronize with the initiator's clock precisely with the help of the `relay counter`. This also enables unsynchronized nodes to become synchronized by listening for a Glossy frame. In fact, an unsynchronized Glossy node indefinitely stays in receive mode until the reception of a Glossy frame.

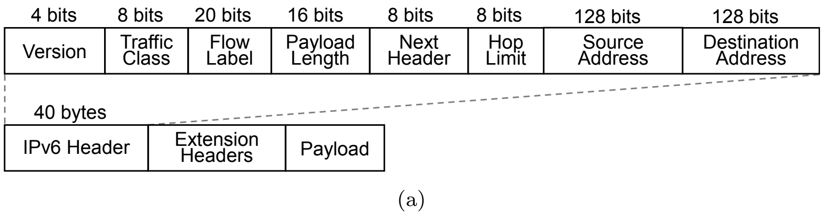
As such, Glossy is just a flooding protocol, but Ferrari et al. evolved Glossy into a combined data link and network layer protocol named LWB [61]. LWB splits time into rounds of varying durations. Each round consists of a number of communication slots, which are assigned to nodes according to their traffic demands. Within such a communication slot, the assigned node can send outgoing packets as a Glossy flood to all other nodes. In fact, in LWB, a node sends every outgoing packet as a Glossy flood to all other nodes. Also, all nodes participate in every Glossy flood, but only further process packets that are destined to themselves. The assignment of communication slots is performed by a special node, called host, which broadcasts its assignments along with the round duration via Glossy floods at the beginning and the end of each round.

2.3 The 6LoWPAN Protocol Stack

On top of IEEE 802.15.4, several upper-layer protocols were specified, namely ZigBee IP, Thread, WirelessHART, ISA100, WIA-PA, WIA-FA, and IPv6 over LoWPANs (6LoWPAN). In this section, we outline 6LoWPAN [94, 95], as well as its companion protocols RPL and the Constrained Application Protocol (CoAP) [63, 96]. Together, IEEE 802.15.4, 6LoWPAN, RPL, and CoAP form the widely employed *6LoWPAN protocol stack* [97], which is depicted in Figure 2.10. Lastly, we highlight the dependence of this stack on IEEE 802.15.4 security.

2.3.1 6LoWPAN

The joint goal of 6LoWPAN and RPL is to seamlessly connect IEEE 802.15.4 networks with IPv6 networks [94, 95]. In this endeavor, the responsibility of 6LoWPAN is to convey IPv6 packets over IEEE 802.15.4 links. To do so, 6LoWPAN, on the one hand, compresses IPv6 packets and, on the other hand, fragments and reassembles IPv6 packets that do not fit within a single IEEE 802.15.4 frame. Besides, a rarely used feature of 6LoWPAN is its support for mesh-under routing, where routing decisions are taken at the data link layer.



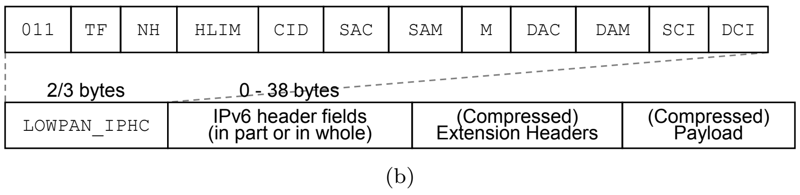


Figure 2.11: Format of (a) an IPv6 packet and (b) a 6LoWPAN compressed IPv6 packet

For illustration, let us look at how 6LoWPAN compresses an IPv6 header. The format of an IPv6 header is shown in Figure 2.11a. Instead of conveying an IPv6 packet “as is”, 6LoWPAN prepends its `LOWPAN_IPHC` header to a compressed version of the IPv6 packet, as shown in Figure 2.11b. The compression of the individual fields of the IPv6 header works as follows:

Version: The Version field is always omitted and assumed to be 6.

Traffic Class and Flow Label: If both the Traffic Class and the Flow Label field are zeroed, both fields are omitted and `TF` is set to 11. The `TF` values 01 and 10 encode other common cases. If no such case applies, both the Traffic Class and Flow Label field are carried in-line and `TF` is set to 00.

Payload Length: The Payload Length field is always omitted since it can either be inferred from the PHY header or from fragmentation headers.

Next Header: While the Next Header field is never omitted, it can be reformatted so as to signal whether the subsequent header is being compressed. More specifically, if the subsequent header is being compressed the Next Header field is re-formatted and the `NH` flag is set. For example, user datagram protocol (UDP) headers are being compressed by using 4-bit identifiers that resolve to the actual 16-bit port numbers and by eliding UDP’s Length field.

Hop Limit: Similar to the Traffic Class and Flow Label fields, if the value of the Hop Limit field is 1, the Hop Limit field is omitted and `HLIM` is set to 01. The `HLIM` values 10 and 11 encode other common cases. If no such case applies, the Hop Limit field is carried in-line and `HLIM` is set to 00.

Network Prefixes: Some predefined network prefixes, such as `FE80::/64`, can be elided by setting certain bits of the `LOWPAN_IPHC` header. Other network prefixes can, at least, be compressed to 4 bits if mappings between 4-bit context identifiers and those network prefixes are configured.

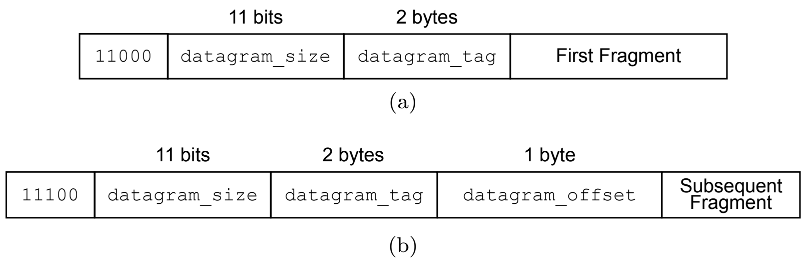


Figure 2.12: Format of the (a) first fragment of a 6LoWPAN fragmented IPv6 packet and (b) a subsequent fragment of a 6LoWPAN fragmented IPv6 packet

Interface Identifiers: Generally, an interface identifier (IID) can only be omitted or compressed if it is derived from an IEEE 802.15.4 address as per 6LoWPAN. Furthermore, the source IID of an IPv6 packet can only be omitted if the originator of the IPv6 packet is also the MAC layer sender since the source IID can then be inferred from the MAC header. Likewise, the destination IID of an IPv6 packet can only be omitted if the destination of an IPv6 packet also is the MAC layer receiver. Finally, an IID can be compressed to 2 bytes if it is derived from an IEEE 802.15.4 short address. That said, the derivation of IIDs from IEEE 802.15.4 addresses, especially short ones, is discouraged for security reasons [98].

If an IPv6 packet does not fit within a single IEEE 802.15.4 frame even after compressing it, 6LoWPAN splits it up into fragments. The format of such fragments is shown in Figure 2.12. Note that the naming of the fields is misleading as 6LoWPAN can not only fragment UDP datagrams, but any IPv6 packet. The field `datagram_size` contains the overall length of the fragmented IPv6 packet. The field `datagram_tag` includes the value of a counter that is incremented each time an IPv6 packet is fragmented. Finally, the `datagram_offset` field carries the offset into the fragmented IPv6 packet.

2.3.2 RPL

RPL organizes an IEEE 802.15.4 network into one or more destination-oriented directed acyclic graphs (DODAGs), each of which is directed toward a single root node, as shown in Figure 2.13 [63]. Typically, but not necessarily, a root node provides a path to another network. Within a DODAG, RPL either routes IPv6 packets upward, i.e., toward the root node, or downward, i.e., away from the root node. This usually means that if a node sends an IPv6 packet to another node inside the same DODAG, the IPv6 packet first travels upward and then downward. RPL only avoids this detour if the destination is on the upward route or if both the source and the destination node are one-hop neighbors anyway.

To avoid detours of IPv6 packets within DODAGs, as well as to enable routing certain IPv6 packets along special DODAGs, RPL supports the formation of additional independent sets of DODAGs. Each independent set of DODAGs (called *RPL Instance* in RPL parlance), may designate different root node(s) and be optimized according to a different routing metric (called *Objective Function* in RPL parlance). However, each additional RPL Instance also consumes additional resources, especially in terms of energy and RAM.

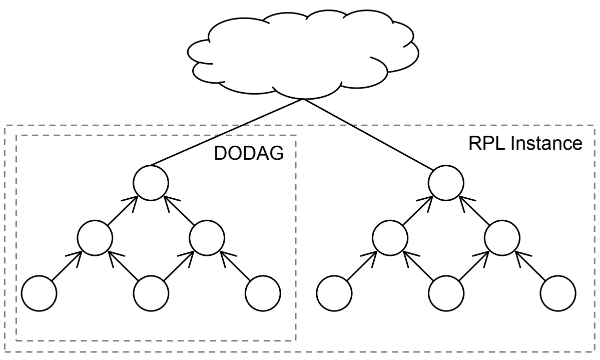


Figure 2.13: Formation of an IEEE 802.15.4 network into two DODAGs

Essentially, any DODAG is formed as follows. Initially, the root node broadcasts a DODAG information object (DIO) message. A DIO message is an Internet control message protocol version 6 (ICMPv6) message whose main contents are (i) an identifier of the corresponding RPL Instance, (ii) an identifier of the DODAG itself, (iii) an identifier of the used Objective Function, and (iv) an indication of the sender's "rank" within the DODAG. This information enables receivers to select a DODAG of a RPL Instance and to build up a set of parents within the selected DODAG. From then on, receivers may also start broadcasting DIO messages so as to invite farther nodes to join the DODAG. In order to save energy, the frequency of DIO messages decreases as a DODAG stabilizes, but increases if it destabilizes so as to quickly adapt to topology changes.

After joining a DODAG, a node can immediately send IPv6 packets upward, whereas downward routing requires additional steps. Specifically, a RPL Instance can decide to implement downward routing in one of two modes. First, in storing mode, each non-leaf node builds up a routing table that contains possible next hops to nodes that the non-leaf node can reach. Second, in non-storing mode, each root node stores possible routes to nodes of its DODAG and performs source routing, thus freeing other nodes from storing routing tables.

Apart from setting up downward and upward routes, RPL also takes care of detecting and removing loops. Loops generally occur if a node moves within a DODAG and if certain RPL messages are missed before such movements. To detect loops, RPL validates that IPv6 packets make progress toward their destinations by adding information to IPv6 extension headers. Upon detecting a loop, RPL initiates a repair operation. However, since loops can not occur in downward routes if RPL operates in non-storing mode, RPL does not add information to downward IPv6 packets when operating in non-storing mode.

2.3.3 CoAP

CoAP can be thought of as a concise version of the Hypertext Transfer Protocol (HTTP) [96]. Unlike HTTP, CoAP is not typically run on top of the Transmission Control Protocol (TCP), but on top of UDP. By obviating the complexity of TCP's connection establishment, connection termination, retransmission mechanism, and congestion control, CoAP implementations become relatively lightweight in terms of traffic, program memory, and RAM demands compared

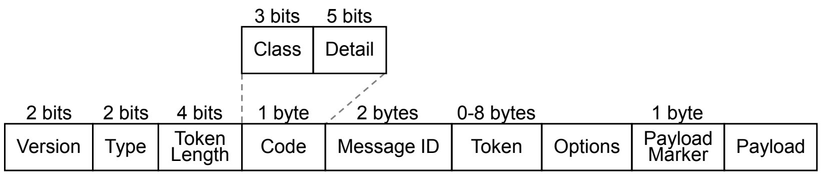


Figure 2.14: Format of CoAP messages

to HTTP counterparts. Nevertheless, CoAP not only supports unreliable message transmissions, but also reliable message transmissions. That is, retransmissions are enabled only on demand for selected CoAP messages, which saves bandwidth and RAM when retransmissions are unnecessary. Other features that set CoAP apart from HTTP are (i) that CoAP allows servers to postpone responses and (ii) that CoAP has a built-in resource discovery mechanism.

More specifically, CoAP defines four types of messages, namely Confirmable messsages (CONs), Non-confirmable messages (NONs), Acknowledgment messages (ACKs), and Reset messages (RSTs), all of which share the binary format shown in Figure 2.14. CONs are retransmitted till a corresponding ACK or RST returns or a maximum number of retransmissions is reached, whatever happens first. By contrast, NONs are not acknowledged, yet may be sent multiple times so as to improve the delivery ratio of NONs. ACKs are sent in response to CONs and may either indicate a success or failure. Additionally, ACKs may contain a piggybacked response. Finally, RSTs are sent in response to a CON or NON if the CON or NON could not be processed.

CoAP further refines the use of CONs, NONs, ACKs, and RSTs like follows. A client always initiates the communication by sending a request to a server. Such a request is either a CON or NON and uses one of four possible methods, namely either `GET`, `PUT`, `POST`, or `DELETE`. Regardless of the message type and method, the client expects one response from the server. If the client sends a CON and if the server can respond immediately, the server responds with an ACK and piggybacks its response on the ACK. On the other hand, if the server receives a CON and wishes to respond later, the server immediately responds with an empty ACK and, at a later time, sends its actual response as a separate CON to the client. Else, if the server encounters problems with a CON, the server replies either with an RST or a negative ACK. As for NONs, the server may either respond with a NON, CON, or RST.

Thus far, we only touched on parts of CoAP that appear in the fundamental CoAP request for comments (RFC) [96]. Apart from this fundamental RFC, there are a couple of complementary RFCs. For example, RFC 7049 defines the Concise Binary Object Representation (CBOR), which is commonly used for encoding the payload of CoAP messages [99]. Also noteworthy, RFC 7641 enables the observation of CoAP resources, and RFC 7959 specifies methods for avoiding the fragmentation of CoAP messages at lower layers [100, 101].

2.3.4 Dependence on IEEE 802.15.4 Security

The 6LoWPAN protocol stack can be complemented with various security measures, such as with IEEE 802.15.4 security at the MAC layer, with RPL's own security mechanisms or IPsec at the network layer [63, 102], with Datagram

Transport Layer Security (DTLS) at the transport layer [103], and with Object Security for Constrained RESTful Environments (OSCORE) at the application layer [104]. To highlight the dependence of the 6LoWPAN protocol stack on IEEE 802.15.4 security, let us consider which attacks become possible if IEEE 802.15.4 security is disabled.

As for 6LoWPAN, two attacks will become possible if IEEE 802.15.4 security is disabled [105, 106]. The first of them exploits that 6LoWPAN cancels the reassembly of an IPv6 packet upon reception of an overlapping fragment. Thus, to block the reception of a 6LoWPAN-fragmented IPv6 packet, an attacker just needs to inject an overlapping 6LoWPAN fragment with the same `datagram_tag` as the 6LoWPAN-fragmented IPv6 packet. Such an attack is called a *fragment duplication attack* [105]. The second attack exploits that, upon reception of a fragment, 6LoWPAN allocates memory for putting it together with the other fragments. Thus, by injecting fragments with different `datagram_tag`s, an attacker can exhaust the memory resources of a victim node. This second attack is called a *buffer reservation attack* [105]. Both attacks will not work out, if IEEE 802.15.4 security filters out any injected and replayed frames.

As for RPL, security may not break down immediately if IEEE 802.15.4 security is disabled. This is because RPL has own optional security mechanisms as a substitute [63]. However, RPL's own security mechanisms are usually not implemented due to gaps in the RFCs [107]. Another substitute is IPsec, but its combination with RPL was not specified, yet [32, 102]. Thus, if IEEE 802.15.4 security is disabled, RPL can typically be attacked in various ways (see [107] for a survey of attacks against RPL). Most of these attacks require an attacker to first inject one or more malicious nodes into a DODAG. Regardless of whether this attack step is required, all attacks against RPL can be classified into three categories [107]. The first category encompasses attacks that make RPL expend energy, memory, or processing resources. This can either be achieved directly by injecting malicious RPL messages, or indirectly by provoking a partly or complete reorganization of a DODAG. The second category encompasses attacks that degrade a DODAG's topology in the sense that suboptimal routes are used or in the sense that single nodes or a subset of nodes become isolated. This can be achieved by injecting malicious RPL messages, manipulating RPL's information contained in IPv6 extension headers, dropping IPv6 packets instead of forwarding them, or a combination thereof. The third category encompasses attacks concerning network traffic, such as (i) overhearing RPL control traffic as a stepping stone to other attacks, (ii) attracting traffic by advertising false ranks in DIOs, or (iii) injecting traffic under the name of non-compromised nodes.

As for CoAP, despite when using an end-to-end security solution, such as IPsec [102], DTLS [103], or OSCORE [104], CoAP will still be affected if IEEE 802.15.4 security is disabled. For instance, an attacker may then launch so-called path-based denial-of-service (PDoS) attacks against CoAP [108]. In the context of CoAP, a PDoS attack works by injecting or replaying a CoAP message that is destined to a distant destination, thereby making each node along the path expend energy for receiving, processing, and forwarding the injected or replayed CoAP message. Only at the final destination, an end-to-end security solution filters out injected and replayed CoAP messages. Moreover, CoAP may suffer from spoofed acknowledgment frames. Specifically, CoAP relies on that NONs are forwarded on a best-effort basis. Yet, at the MAC layer, the forwarding of NONs is stoppable by spoofing acknowledgment frames [36]. Unfortunately, even if IEEE 802.15.4 security is enabled, this will only complicate acknowledgment spoofing attacks, as we discuss in Chapter 5.

Given that IEEE 802.15.4 security can prevent all of the above attacks, it is worthwhile to overcome its limitations, namely the lack of (i) a protocol for establishing session keys among neighboring IEEE 802.15.4 nodes, (ii) strong freshness (unless using TSCH), and (iii) effective defenses against acknowledgment spoofing attacks. However, even if all these limitations are overcome, the security provided by IEEE 802.15.4 security will still depend on the secrecy of the employed CCM keys. That said, if CCM keys leak, IEEE 802.15.4 security can still be useful, provided that pairwise CCM keys are employed. This is because pairwise CCM keys will, at least, contain the repercussions of key losses. Beyond that, the use of pairwise CCM keys provides a sufficient condition for intrusion detection: the sender of a malicious fresh authentic frame can be considered compromised [10]. Likewise, the sender of an abnormal high number of fresh authentic frames can be considered compromised.

2.4 Operating Systems

6LoWPAN stacks are, e.g., available for the open-source operating systems Contiki and RIOT [109, 110]. We considered both these operating systems for prototyping our denial-of-sleep-resilient MAC layer as they are mainstream. Eventually, we opted for Contiki due to the availability of implementations of ContikiMAC and a predecessor of AKES [10, 60]. Meanwhile, a variant of ContikiMAC became available for RIOT, too. Below, we present the main concepts of Contiki, which was recently renamed to Contiki-NG after a major revision, and RIOT.

2.4.1 Contiki-NG

Contiki-NG has an event-driven kernel, meaning that every process has a set of event handlers, each of which runs to completion [109]. Since event handlers run to completion, all processes can share a single stack. In particular, Contiki-NG neither has to store the state of the microprocessor nor the call stack after running an event handler. Also, there is no need for locking as a result.

However, while an event-driven kernel saves RAM and obviates locking, it incurs three problems. First, usually an event-driven kernel leads to hard-to-read code because it necessitates (i) plenty of event handlers and (ii) separating logically related code. However, Contiki-NG solves this first problem via Protothreads [111]. The idea of Protothreads is to hide event handlers from developers by means of sophisticated preprocessor macros. For example, `PT_WAIT_UNTIL(pt, condition)` pauses a Prothread until a condition is met. Second, long-running event handlers monopolize the microprocessor. One solution to this second problem is to implement preemptive multi-threading on top of an event-driven kernel [109]. Alternatively, Contiki-NG enables pausing a Protothread voluntarily via `PT_YIELD(pt)` [111]. Third, an event-driven kernel makes it very difficult to adhere to real-time constraints. Unfortunately, we are not aware of any solution to this third problem other than running real-time code within appropriately prioritized interrupt service routines (ISRs).

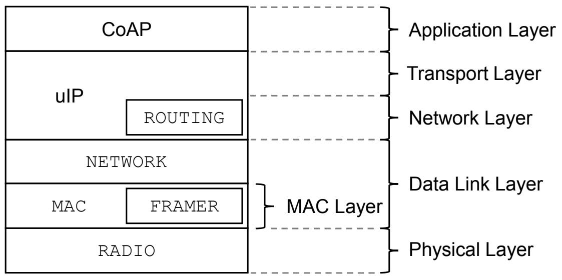


Figure 2.15: Contiki-NG's 6LoWPAN Stack

A critical feature of any operating system for low-power IoT devices is the ability to set an IoT device to sleep if nothing is left to be done, rather than busy waiting in the meantime. In Contiki-NG, this is taken care of by the `lpm` module. This module enters an appropriate LPM until the next timer event.

Contiki-NG's network stack resembles the 6LoWPAN protocol stack, as shown in Figure 2.15. The lowest layer of Contiki-NG's network stack constitutes an exchangeable `RADIO` driver, which implements the PHY in cooperation with the radio transceiver. On top of the `RADIO` driver, Contiki-NG has an exchangeable `MAC` driver, which implements the MAC protocol. Internally, an exchangeable `FRAMER` handles the assembling and parsing of radio frames. On top of the `MAC` driver, an exchangeable `NETWORK` driver may, e.g., implement 6LoWPAN. On top of the `NETWORK` driver, uIP (pronounced "micro IP") implements IPv6, UDP, ICMPv6, as well as other network and transport layer protocols. The implementation of the routing protocol, however, is separate from uIP and encapsulated behind an interface. Finally, on top of uIP, application layer protocols, such as CoAP, can run.

2.4.2 RIOT

As opposed to Contiki-NG, RIOT is especially designed for adhering to real-time constraints [112]. The basic approach of RIOT is to abstain from an event-driven kernel and to implement true multi-threading instead. Consequently, RIOT's scheduler has to store the microprocessor's state before context switches and each thread has to have its own call stack. Moreover, threads need to use locking as they may preempt each other, depending on their priority.

In order to provide real-time guarantees, the scheduler of RIOT is implemented so that scheduling decisions are taken in constant time. For this, the scheduler maintains 32 circular linked lists, each of which stores paused threads for each of the 32 priority levels. Thus, for finding the paused thread with the highest priority, the scheduler can just take the highest-priority circular linked list with paused threads and take the first paused thread therefrom in $\mathcal{O}(1)$.

For locking and communication among threads, RIOT offers mutexes and synchronous message passing, respectively. Mutexes are realized as queues of waiting threads. Threads that have to wait for a mutex add themselves to the corresponding queue and then issue a context switch. Synchronous message passing works similarly in that a queue of pending messages is maintained. A sender thread can enqueue messages while receiver threads can dequeue messages in a blocking manner. Both mutexes and synchronous message passing work without allocating memory other than on the thread's stack. This is essential for providing real-time guarantees. Furthermore, queues are sorted with respect to priorities, which avoids deferring higher-priority threads for longer than necessary.

To save energy if nothing is left to be done, RIOT sets up a timer interrupt until when the device can sleep and then enters an LPM. While this is unsurprising, this actually deviates from the common approach followed in real-time operating systems of configuring a periodic timer interrupt.

RIOT also comes with a network stack, named Generic (GNRC) [113]. Applications interact with GNRC via a high-level network interface, such as `sock_ip` for raw IPv6 traffic or `sock_udp` for UDP traffic. Internally, GNRC is even more modular than Contiki-NG's network stack. In GNRC, each protocol has its own thread and interacts with other protocols via the `netapi` interface. The lowest layer protocol, which typically implements the MAC protocol, interacts with the hardware via the `netdev` interface.

2.5 Summary

In this chapter, we have looked at three radio standards employed in today's low-power IoT applications in more detail. Among them, IEEE 802.15.4 excels at interconnecting dense deployments of IoT devices. Furthermore, IEEE 802.15.4 can bridge longer distances if an appropriate PHY and frequency band are chosen. What makes IEEE 802.15.4 particularly attractive for us, is the possibility to easily experiment with third-party security additions. We have also mentioned standards that build on IEEE 802.15.4 and outlined the 6LoWPAN protocol stack. This protocol stack seamlessly connects IEEE 802.15.4 networks with IPv6 networks and enables RESTful interactions with IoT devices. However, the 6LoWPAN protocol stack depends on IEEE 802.15.4 security, which presently has three limitations one should overcome. Finally, we have presented the main concepts of Contiki-NG and RIOT. For practical reasons, we have selected Contiki-NG for prototyping our denial-of-sleep-resilient MAC layer.

Chapter 3

Cryptographically-Secure Pseudo-Random Number Generator

This chapter presents our CSPRNG, which seeds itself with power-up SRAM states and radio noise. To begin with, we motivate the need for cryptographic random numbers and argue that IoT devices should be capable of generating cryptographic random numbers entirely by themselves. Furthermore, we point out limitations of current methods for extracting seeds from power-up SRAM states. Then, we go into information-theoretic notions and survey related work. Next, we explain our superior method for extracting seeds from power-up SRAM states. Finally, we describe the design of our CSPRNG and give an experimental evaluation.

3.1 Extended Problem Statement

Random numbers are a preliminary to many cryptographic operations. For example, the generation of a CCM key requires a random number. Yet, to be suitable in a cryptographic context, random numbers generally have to fulfill a special requirement. Concretely, such random numbers should appear uniformly distributed to a computationally-bounded attacker [45]. We refer to random numbers that fulfill this requirement as *cryptographic random numbers*.

For generating cryptographic random numbers, an efficient method is to expand a secret truly random bit string (called *seed*) into a stream of cryptographic random numbers [45]. The mathematical constructs that follow this method are often called *CSPRNGs*. The challenging part of implementing a CSPRNG is usually not the implementation of the CSPRNG itself, but the generation of a seed in the first place. This is especially true for IoT devices as most IoT devices lack common sources of randomness, such as user interaction or hard disks.

A straightforward solution is to ask the user to preload his IoT devices with seeds [34]. This, however, is tedious if done via cables. Several wireless preloading schemes approach this usability problem (see [114] for a survey), but may require cryptographic random numbers for securing the wireless preloading process itself [8, 115, 116], making them often inapplicable to preloading seeds. Moreover, a subtlety of preloading seeds is reboots. After a reboot, a CSPRNG must not start over with the same seed as, otherwise, the same sequence of cryptographic random numbers is regenerated. Hence, some data has to be kept across reboots. Yet, the only non-volatile memory on most IoT devices is flash memory, which is slow, energy consuming, as well as prone to wear [117].

After all, an IoT device may have to be able to reseed its CSPRNG for two reasons anyway. First, at some point, a seed is exhausted and has to be replaced [46]. Second, reseeding is also necessary to achieve forward and backward security [45]. Forward and backward security is the property of a CSPRNG that, if an attacker compromises the internal state of a CSPRNG, he is unable to predict past and future outputs of the CSPRNG, respectively [45].

Fortunately, though IoT devices lack common sources of randomness, they may have alternative sources of randomness available, such as radio noise [118, 119, 120], sensor readings [121, 122, 123], clock differences [124, 125], dynamic RAM (DRAM) decays [125], or power-up SRAM states [49, 50, 52, 126, 127, 128]. Especially power-up SRAM states are widely available on low-power IoT devices due to the better energy efficiency of SRAM compared to that of DRAM. Thus far, there, however, seems to be no method for extracting seeds from power-up SRAM states in a practical and information-theoretically secure manner. In fact, using a deterministic hash function, as suggested by van der Leest et al. [50, 126, 127, 128], is not information-theoretically secure, whereas the information-theoretically secure extractor suggested by Holcomb et al. is impractical due to its dependence on additionally provisioned truly random bits [49]. Another issue with power-up SRAM states is that their randomness is adversely affected by various factors [49, 50, 51, 52, 53].

In this chapter, we make four contributions:

- First, we point out a trick that enables sampling power-up SRAM states not only once at boot time, but multiple times, even at runtime. Our trick is to leverage that certain hardware platforms only retain a fraction of their SRAM in deep LPMs, such as the CC2538 or the SiM3U167-B-GDI [14, 129]. Thus, by switching to a deep LPM and back again, one can obtain fresh power-up SRAM states at runtime, too.

- Second, based on the above trick, we propose a superior method for extracting seeds from power-up SRAM states. Unlike current methods, ours is both information-theoretically secure and practical. Additionally, many factors threatening the randomness of power-up SRAM states do not apply when our method is used.

- Third, we propose the design of a CSPRNG, which seeds itself with power-up SRAM states using our method, as well as with radio noise for additional security. Besides, our proposed CSPRNG provides forward and backward security if it is configured to reseed itself regularly at runtime.

- Forth, we compare the energy consumption of seeding an IoT device either with radio noise or power-up SRAM states. While seeding with power-up SRAM states turned out to be more energy efficient, we advise against using any of these two entropy sources alone since both have weaknesses.

3.2 Preliminaries

The usage of a CSPRNG happens in three phases, namely entropy gathering, randomness extraction, and pseudo-random number generation.

3.2.1 Entropy Gathering

During entropy gathering, the user (e.g., an IoT device) obtains an observation from an entropy source. An *entropy source* X is a discrete random variable. We use $x \leftarrow X$ to denote that x is drawn at random according to the probability distribution of X. An attacker would try to predict an observation by choosing its most likely value. Therefore, the *predictability* of an entropy source X is $\max_x \Pr[X = x]$. Correspondingly, the *min-entropy* $H_\infty(X)$ of X is

$$H_\infty(X) \overset{\text{def}}{=} -\log_2(\max_x \Pr[X = x]) \tag{3.1}$$

For comparison, the *Shannon entropy* $H_1(X)$ of X is

$$H_1(X) \overset{\text{def}}{=} -\sum_x \Pr[X = x] \log_2 \Pr[X = x] \tag{3.2}$$

While Shannon entropy is a measure of the average level of surprise when seeing an observation $x \leftarrow X$, min-entropy is a measure of the minimum level of surprise when seeing an observation $x \leftarrow X$.

3.2.2 Randomness Extraction

To serve as seeds, observations of an entropy source have to be distilled into shorter close-to-uniformly distributed bit strings. The functions for doing so are often referred to as extractors. Basically, an extractor Ext is a function $\text{Ext} : \mathcal{N} \times \{0,1\}^d \rightarrow \{0,1\}^m$ that takes an observation $x \in \mathcal{N}$, as well as public truly random bits (called *extractor seed*) $s_{\text{Ext}} \in \{0,1\}^d$ as input, and outputs a bit string $s \in \{0,1\}^m$, where $d < m$. Such a function is called an (r, m, ϵ)-*strong extractor* if for all entropy sources X with $H_\infty(X) \geq r$ the statistical distance between the outputs of Ext and the uniform distribution over $\{0,1\}^m$ is smaller or equal than ϵ [130]. For our purposes, it is important that extractors necessarily are randomized functions. Otherwise, in the case of a deterministic function, there will, e.g., exist entropy sources with high min-entropy that are mapped to outputs that are trivially distinguishable from random [131]. This raises a conflict since, to be able to generate a seed, the user (e.g., an IoT device) has to have an extractor seed already. There are two solutions to this:

Limiting input distributions: On the one hand, if an entropy source follows a specific distribution, a seedless extractor may exist [132, 133, 134]. Most prominently, if an entropy source is composed of i.i.d. Bernoulli trials, the seedless von Neumann extractor is applicable [132]. Let X be an entropy source that is composed of Bernoulli trials $X_{1,1}, X_{1,2}, X_{2,1}, X_{2,2}, \ldots \sim \text{Bernoulli}(p)$, where p is the probability of observing a value of 1. Given an observation $x_{1,1}, x_{1,2}, x_{2,1}, x_{2,2}, \ldots \leftarrow X$, the von Neumann extractor iterates through each pair $(x_{i,1}, x_{i,2})$. If $x_{i,1} = 0$ and $x_{i,2} = 1$, the von Neumann extractor appends 1 to the resulting output. If $x_{i,1} = 1$ and $x_{i,2} = 0$, the von Neumann extractor appends 0 to the resulting output. Otherwise, if $x_{i,1} = x_{i,2}$, the von Nemann extractor appends nothing to the resulting output. Since $\Pr[X_{i,1} = 1, X_{i,2} = 0] = \Pr[X_{i,1} = 0, X_{i,2} = 1] = p^2$, the von Neumann extractor produces uniformly distributed outputs.

Limiting attackers: On the other hand, Barak et al. considered using a fixed extractor seed, and even sharing it among multiple users, enabling the fixed extractor seed to be stored in the firmware of IoT devices [135, 136]. Barak et al. found this approach to work out if an attacker has limited capabilities. Specifically, the attacker must not be able to modify more than a limited number of boolean properties of the distribution of the employed entropy source and such modifications must not cause the min-entropy of the employed entropy source to fall below a threshold.

Altogether, information-theoretically secure randomness extraction without additionally provisioned extractor seeds is feasible. Nevertheless, practitioners tend to use deterministic hash functions for randomness extraction. This entails the risk that despite of using an entropy source with high min-entropy, the distilled seeds may not be close-to-uniformly distributed.

3.2.3 Pseudo-Random Number Generation

With a seed in place, the user (e.g., an IoT device) can efficiently generate a stream of cryptographic random numbers $r_1, r_2, \ldots$by, e.g., invoking a block cipher in output feedback mode (OFB) [34, 47, 137]. Concretely, let $\mathrm{Enc} : \{0,1\}^k \times \{0,1\}^l \to \{0,1\}^l$ be a block cipher, where k denotes key length and l block size. Furthermore, let $s = s_{\mathrm{Enc}} \| r_1$ be a seed, where $s_{\mathrm{Enc}} \in \{0,1\}^k$ and $r_1 \in \{0,1\}^l$. OFB generates $r_{i+1} \in \{0,1\}^l$ inductively as $\mathrm{Enc}(s_{\mathrm{Enc}}, r_i)$.

3.3 Related Work

In the following, we will first sum up prior efforts on identifying and evaluating available entropy sources on IoT devices. Then, we point out another use of power-up SRAM states, namely as physically unclonable functions (PUFs).

3.3.1 Entropy Sources on IoT Devices

Holcomb et al. were first among many to consider power-up SRAM states as an entropy source [49, 50, 52, 126, 127, 128]. The motivation behind using power-up SRAM states as an entropy source is that, after powering up, each SRAM cell eventually settles to either 0 or 1. The tendency of whether an SRAM cell settles to 0 or 1 mainly depends on the individual characteristics of each cell. These characteristics stem from uncontrollable variations during the manufacturing of each individual SRAM cell. Yet, at runtime, six additional factors were found to influence the probability of an SRAM cell to settle to either 0 or 1 at powering up. First, when exposed to low temperatures, many SRAM cells become either 0-skewed or 1-skewed [49, 50]. Second, burn-in effects occur when an SRAM cell stores the same bit for several hours [49]. Third, data remanence refers to the issue that data stored in an SRAM cell persists across very short power outages [51]. Forth, Liao et al. observed that, when powering up after a long period

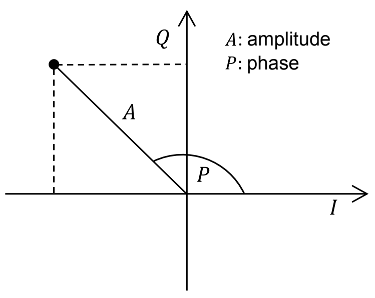


Figure 3.1: Conversion of an I/Q reading to amplitude and phase

of time, SRAM cells have a different tendency of settling to either 0 or 1 as compared to powering up after a short power outage [52]. Fifth, silicon aging causes the randomness of power-up SRAM states to increase over long periods of time [53]. Finally, an attacker may be able to predictably influence power-up SRAM states by tampering with the supply voltage [49].

Alternatively, several manuals of IEEE 802.15.4 transceivers suggest using radio noise as an entropy source [14, 138]. Radio noise can be measured in one of two ways. First, a common feature of IEEE 802.15.4 transceivers is their support for energy scans, which return the average amplitude of the current channel over 8 symbol periods. Unfortunately, according to a small experiment we conducted, the outcomes of energy scans can be forced to a constant value by emitting interference, thereby nullifying the min-entropy of energy scans. Second, some transceivers, such as that of the CC2538, also provide I/Q readings [14]. I/Q readings are cartesian representations of the amplitude and phase of the sinusoidal wave in the channel, as depicted in Figure 3.1. According to experiments of Yan et al., influencing the least significant bits (LSBs) of I/Q readings is very hard [139]. Yan et al. could only influence the LSBs of I/Q readings by connecting the antennas of a software-defined radio (SDR) platform and a CC2538 radio transceiver via a SubMiniature version A (SMA) cable.

A similar approach is the gathering of entropy from incoming frames [118, 119, 120]. In particular, received signal strength indicators (RSSIs) and link quality indicators (LQIs) of incoming IEEE 802.15.4 frames can serve as entropy sources [118, 120]. Also, erroneous IEEE 802.15.4 frames, i.e., IEEE 802.15.4 frames with false checksums, were considered as an entropy source [119, 120]. Yet, RSSIs, LQIs, as well as erroneous frames are susceptible to frame injection attacks [120]. Moreover, RSSIs are vulnerable to eavesdropping, too [120].

Hennebert et al. considered sensor readings as entropy sources [121]. Unfortunately, they found sensor readings to be weak entropy sources. Only special sensors, such as accelerometers and microphones, serve well as entropy sources [122, 123], but are not available on all IoT devices.

More commonly available entropy sources are those based on clocks. For instance, a straightforward approach is to read the LSB of one clock at intervals that are measured by another clock [124]. Besides, Mowery et al. proposed using execution times as an entropy source [125]. Yet, it remains to be investigated how suitable these entropy sources for cryptographic purposes are [124, 125].

Also, Mowery et al. considered using DRAM decays as an entropy source [125]. Indeed, they conclude that DRAM decays is a good entropy source on

DRAM-based IoT devices. But, on low-power IoT devices, SRAM is usually preferred over DRAM since SRAM is more energy efficient than DRAM.

In sum, entropy sources on IoT devices fall into three classes. First, there are weak entropy sources, such as energy scans. Second, there are strong entropy sources that are only available on IoT devices with special hardware, such as accelerometers. Finally, there are also a few strong entropy sources that are commonly available on IoT devices, such as power-up SRAM states.

3.3.2 SRAM-based Physically Unclonable Functions

Power-up SRAM states have also found adoption as PUFs [48, 126, 140, 141]. Basically, a PUF is a physical system that, in response to a challenge, generates a response and fulfills four requirements, namely reliability, unclonability, tamper evidence, and unpredictability [48]. Reliability requires the responses of a PUF to be relatively stable across the range of operating conditions. Unclonability requires that, without access to a PUF, it is very difficult to obtain the response to a challenge of the PUF. Tamper evidence requires that if an attacker physically tampers with a PUF, the PUF will become unusable, e.g., due to producing responses that deviate from the original pattern. Finally, unpredictability requires the generated responses to have a high min-entropy with respect to all possible responses of the class of PUFs under consideration.

One example use case of a PUF is a secure key storage [126]. Usually, long-term keys need to be stored in non-volatile memory, where they are accessible when a device is off, as well as when it is running. This security weakness is avoidable by means of a PUF. The idea is as follows. In an enrollment phase, a special, so-called fuzzy extractor is given (i) one response of the PUF and (ii) an extractor seed [130]. The fuzzy extractor then produces two outputs: (i) public helper data P, usually including the employed extractor seed, and (ii) a symmetric key R that can, e.g., serve for encrypting and authenticating cryptographic material stored in non-volatile memory. The public helper data P is constructed so that it assists the fuzzy extractor in restoring R given a fresh PUF response. Thus, at runtime, R can be restored by means of P so as to, e.g., decrypt and authenticity check cryptographic material stored in non-volatile memory. Another example use case of SRAM PUFs, in particular, is to use their noisy responses as entropy sources [49, 50, 52, 126, 127, 128].

3.4 Proposed Method for Extracting Seeds from Power-Up SRAM States

Our proposed method for extracting seeds from power-up SRAM states involves four steps, as shown in Figure 3.2:

1. Step 1 is to reset the SRAM. For resetting the SRAM, our method relies on that the underlying hardware platform does not retain the complete SRAM in a deep LPM. This enables resetting the SRAM by switching to that deep LPM and back again after a configurable time t_{lpm}. Note that this step is also necessary at boot time. This is because our method may be executed after a warm reboot or a long power outage. In the first case, some SRAM cells may no longer store their power-up values. In the

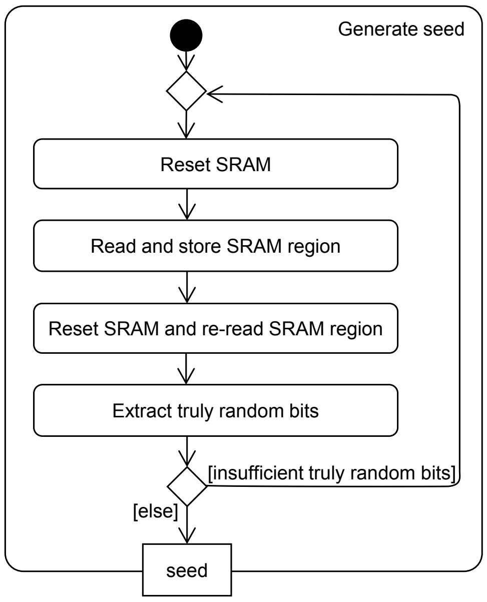

Generate seed
Reset SRAM
Read and store SRAM region
Reset SRAM and re-read SRAM region
Extract truly random bits
[insufficient truly random bits]
[else]
seed

Figure 3.2: Proposed method for extracting seeds from power-up SRAM states

latter case, the SRAM cells' tendencies of settling to either 0 or 1 may be different as compared to powering up after a short power outage [52]. This would be detrimental to our method as will become apparent shortly.

2. Step 2 is to read a region within the just reset SRAM and to store the result within an SRAM region that is retained in deep LPMs.

3. Step 3 is to reset the SRAM again and to re-read the same SRAM region.

4. Step 4 is to compare, for each SRAM cell in the twice read SRAM region, whether its value has flipped from 0 to 1, from 1 to 0, or stayed unchanged. In the case of a flip from 0 to 1, our method appends a 1 to the resulting seed. In the case of a flip from 1 to 0, our method appends a 0 to the resulting seed. In the case of no flip, our method appends nothing to the resulting seed. If all comparisons are done and the resulting seed does not yet have the desired length, our method repeats from Step 1.

Our method has the following features:

Information-theoretical security: Observe that the probability that an SRAM cell flips from 0 to 1 is the same as that the SRAM cell flips from 1 to 0. This is because, after a short power outage of the order of seconds, each SRAM cell has a stable probability of settling to a value of 1 [52]. Our method takes advantage of this fact by applying the proven von Neumann extractor to consecutive power-up values of an SRAM cell.

Resistance to freezing attacks: In the face of low temperatures, many SRAM cells become either 0-skewed or 1-skewed [49, 50]. Nevertheless, our method terminates as long as some flips occur in each round. According to results of von Herrewege et al., this is the case for both SRAM

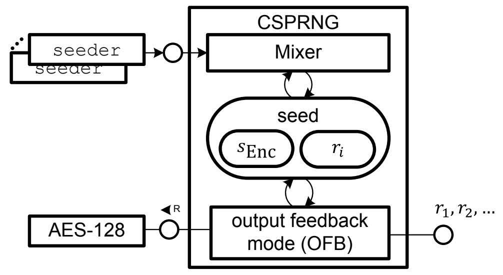


Figure 3.3: Design of our CSPRNG

chips they tested at $-30°C$ [50]. Likewise, all seven SRAM chips tested by Schrijen et al. do not lose all their randomness at $-40°C$ [142].

Resistance to burn-in effects: Our method also resists burn-in effects [49]. This is because our method makes use of an SRAM region where no data is stored for long. On the contrary, the SRAM region that is not retained in a deep LPM can only contain short-term data as it is deleted when entering the deep LPM, which should happen regularly for saving energy.

Resistance to data remanence: If the parameter t_{lpm} is configured long enough, data remanence is not an issue to our method [51].

3.5 Implementation

We implemented our method as part of an entirely new CSPRNG for Contiki-NG. Figure 3.3 gives an overview of our CSPRNG. At the core of our CSPRNG, we use OFB-AES-128 to generate cryptographic random numbers [34, 47, 137]. Hence, the internal state of our CSPRNG consists of an 128-bit AES key s_{Enc} and the next 128-bit cryptographic random number r_i that will be provided to applications. So-called `seeders` generate 256-bit seeds and pass them to our CSPRNG. When a new seed is passed to our CSPRNG, we mix it with the internal state by XORing both. Thus, if the internal state leaks, the attacker is unable to predict future cryptographic random numbers beyond the point where a seed unknown to the attacker is being XORed. Likewise, if the internal state leaks, the attacker is unable to restore past cryptographic random numbers beyond the point where a seed unknown to the attacker was XORed.

So far, we implemented two `seeders`, namely the `sram_seeder` and the `iq_seeder`. Both of them have CC2538 SoCs as their target platform [14]. CC2538 SoCs are built into various IoT prototyping platforms, such as Re-Motes and OpenMotes. We worked with OpenMotes, such as the one in Figure 3.4.

The `sram_seeder` follows our proposed method for extracting uniformly distributed seeds from power-up SRAM states. For resetting the SRAM, the `sram_seeder` leverages that CC2538 SoCs do not retain data stored in the second half of their SRAM in LPM 2. Hence, to reset the SRAM, the `sram_seeder` switches to LPM 2 and back again after $t_{\mathrm{lpm}} = 2\mathrm{s}$. The duration of 2s was chosen because with shorter durations, such as 1s, we encountered data remanence.

The `iq_seeder` extracts seeds from I/Q LSBs. For randomness extraction, we assume that an attacker cannot modify more than 64 boolean properties of

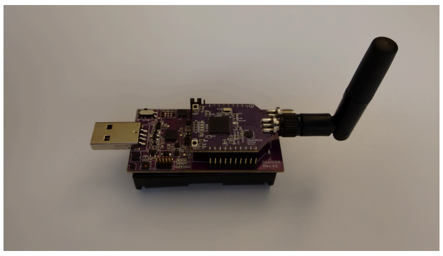


Figure 3.4: OpenMote

the distribution of I/Q LSBs, and that 200 I/Q LSBs (i.e., 200 I LSBs and 200 Q LSBs) always have a min-entropy of at least 256. These assumptions allow us to use Barak et al.'s Toeplitz matrix-based extractor $\mathrm{Ext}_{\mathrm{Toeplitz}} : \{0,1\}^n \times \{0,1\}^d \rightarrow \{0,1\}^m$ [135]. $\mathrm{Ext}_{\mathrm{Toeplitz}}$ multiplies an $\mathcal{F}^{m' \times n'}$ Toeplitz matrix with the observation $x \in \mathcal{F}^{n' \times 1}$ of an entropy source, where $\mathcal{F}$ is a finite field, and the first column and first row of the Toeplitz matrix are defined by a fixed extractor seed $s_{\mathrm{Ext}} \in \mathcal{F}^{m'+n'-1}$. Our `iq_seeder` sets $\mathcal{F} = GF(2^8)$, $m = 8m' = 128$, $n = 8n' = 400$, and $d = 8(m' + n' - 1) = 520$. If invoked, the `iq_seeder` samples 400 I/Q LSBs at times when the channel seems clear according to CCAs and calls $\mathrm{Ext}_{\mathrm{Toeplitz}}$ twice: the first 200 I/Q LSBs yield s_{Enc} and the other 200 I/Q LSBs yield r_1. According to results of Skorski, our `iq_seeder` achieves with probability $1 - 10^{-9}$ over the choice of s_{Ext} that the statistical distance between s_{Enc} and the uniform distribution U_{128} over $\{0,1\}^{128}$ is at most 10^{-9} [136]. Likewise, the statistical distance between r_1 and U_{128} is at most 10^{-9} with probability $1 - 10^{-9}$ over the choice of s_{Ext} [136].

While the `iq_seeder` can also be invoked at runtime, we note that the `iq-_seeder` is more appropriate for seeding at boot time. This is because the `iq-_seeder` explicitly enables the receive mode, which may conflict with the MAC protocol. Hence, for reseeding, we suggest implementing an additional `seeder` that samples I/Q LSBs at times when the receive mode is enabled by the MAC protocol anyway. Not only will this avoid conflicts, but also save energy.

3.6 Evaluation

In the following, we first show the viability of seeding a CC2538-based IoT device like explained above by showing that the min-entropy of power-up SRAM states and I/Q LSBs is sufficiently high. Subsequently, we compare the energy consumption of seeding with I/Q LSBs, power-up SRAM states, or both.

3.6.1 Entropy Assessment

To estimate the min-entropy of power-up SRAM states, the following sequence was repeated 1000 times. An OpenMote (i) switched to LPM 2, (ii) back to the active mode after 2s, and (iii) dumped a contiguous 512-byte SRAM region that was not retained in LPM 2. Throughout, the temperature was 22°C.

Figure 3.5 shows a histogram of the observed empirical probabilities per each SRAM cell of settling to 1 at powering up and the resultant empirical min-entropys. Most SRAM cells are either 1- or 0-skewed and hence yield very

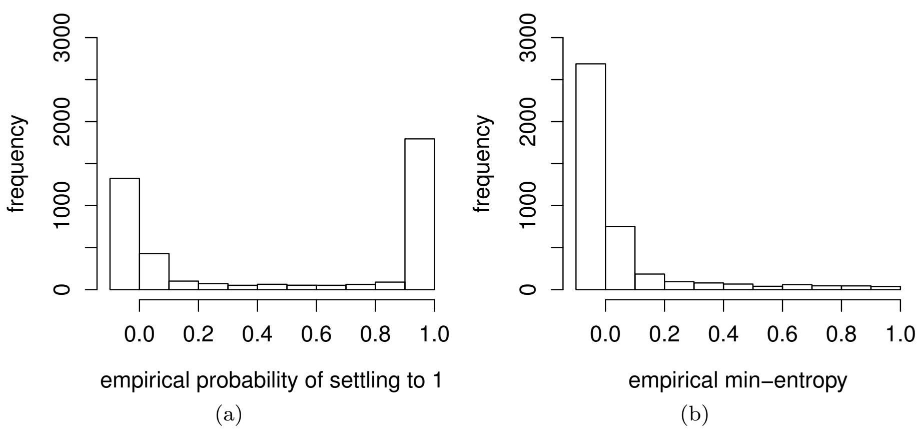


Figure 3.5: Histogram of (a) the empirical probabilities per each SRAM cell of settling to 1 at powering up and (b) the resultant empirical min-entropys

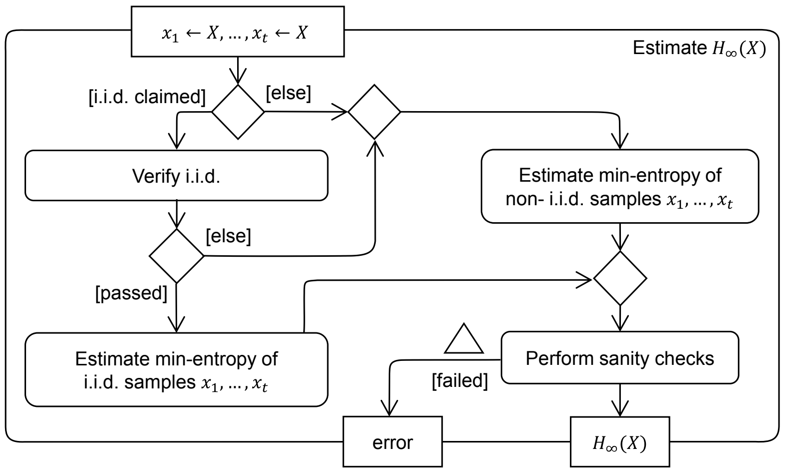

$x_1 \leftarrow X, \ldots, x_t \leftarrow X$
Estimate $H_\infty(X)$
[i.i.d. claimed]
[else]
Verify i.i.d.
Estimate min-entropy of non- i.i.d. samples $x_1, \ldots, x_t$
[else]
[passed]
Estimate min-entropy of i.i.d. samples $x_1, \ldots, x_t$
Perform sanity checks
[failed]
error
$H_\infty(X)$

Figure 3.6: Workflow of NIST 800-90B

little min-entropy. On the other hand, there are numerous neutral-skewed cells, too. The mean over all empirical min-entropys is 0.073. Of course, this value is specific to the concrete CC2538 SoC and memory region we used, but, fortunately, our method for extracting seeds from power-up SRAM states is agnostic to these choices. This is because our method adaptively increases the number of rounds in the face of few flips. Our method only depends on that some flips occur in each round, as, otherwise, our method may not terminate.

The min-entropy of I/Q LSBs was estimated using an open-source implementation of NIST 800-90B [143]. Figure 3.6 depicts the workflow of NIST 800-90B. From a user's perspective, if X is the entropy source under test, NIST 800-90B takes values $x_1 \leftarrow X, \ldots, x_t \leftarrow X$ as input and outputs either an error or an estimation of $H_\infty(X)$. Choosing $t = 1,000,000$ is the minimum required by NIST 800-90B. Internally, NIST 800-90B has two tracks - an i.i.d. track for the case that $x_1, \ldots, x_t$ are mutually independent and identically distributed (i.i.d.),

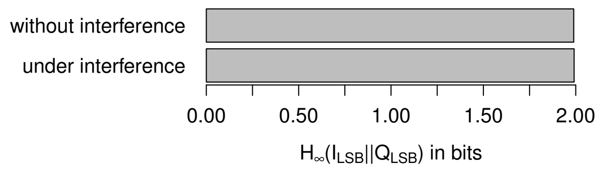


Figure 3.7: Min-entropy of pairs of I/Q LSBs

and a non-i.i.d. track for the opposite case. On the i.i.d. track, the min-entropy is estimated by calculating the empirical min-entropy like above. Yet, prior to doing so, NIST 800-90B verifies the i.i.d. property by calculating various statistics, namely an over/under runs score, an excursion score, a directional runs score, a covariance score, and a collision score. On the non-i.i.d. track, the min-entropy is estimated by taking the minimum over five min-entropy estimators, namely a collision test, a partial collection test, a Markov test, a compression test, and a frequency test. Finally, if the estimated min-entropy $H_\infty(X)$ passes two additional sanity checks, NIST 800-90B outputs $H_\infty(X)$.

We sampled $t = 1,000,000$ pairs of I/Q LSBs and passed them to NIST 800-90B. As a result, NIST 800-90B reported that I/Q LSBs are i.i.d., as well as that they are almost perfectly random, as shown in Figure 3.7. We also repeated this experiment with the difference being that a nearby OpenMote continuously transmitted while sampling I/Q LSBs. As shown in Figure 3.7, the results did not change much. However, in both cases, NIST 800-90B did not report that I/Q LSBs are perfectly random, which is why it makes sense to apply a randomness extractor prior to using I/Q LSBs as seeds. Another cautionary note about these results is that they do not exclude that the min-entropy of I/Q LSBs reduces under other sorts of interference or eavesdropping. In fact, Yan et al. observed abnormal runs of zeroes and ones under extremely strong interference [139]. As a countermeasure, our `iq_seeder` only samples I/Q LSBs at times when the channel seems clear according to CCAs.

3.6.2 Energy Consumption

To measure the energy consumption of our `seeders`, an OpenMote, a μCurrent Gold, and a Rigol DS1000E oscilloscope were connected in series, as shown in Figure 3.8. This enables capturing an OpenMote's current draw over time like detailed in [144]. However, deviating from [144], we added a 0.1Ω shunt resistor across the input terminals of the μCurrent Gold in order to reduce the noise on the oscilloscope. Using this experimental setup, the current draw while seeding with the `iq_seeder` and the `sram_seeder` was sampled piece by piece for each phase of the seeding process. 100 samples per piece were taken. Throughout, the `sram_seeder` was configured to use a 1024-byte SRAM region so that the `sram_seeder` never had to perform more than one round.

Figure 3.9 sums up the results. The `iq_seeder` consumes a considerable amount of energy for entropy gathering since it needs to enable the energy-consuming receive mode for sampling I/Q LSBs. Moreover, the randomness extraction with Barak et al.'s Toeplitz matrix-based extractor makes up the bulk of the energy consumption of the `iq_seeder`. By contrast, the `sram_seeder` consumes much less energy, mainly because the `sram_seeder` gathers

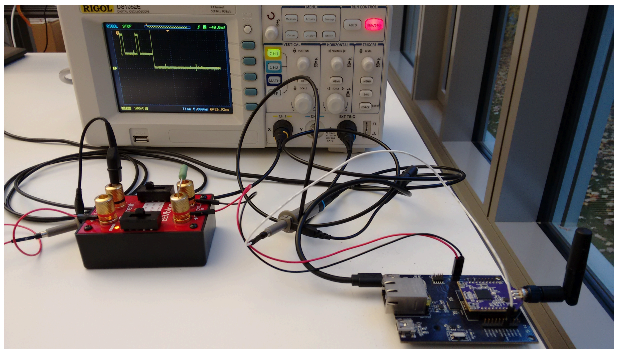


Figure 3.8: Experimental setup for measuring the current draw over time

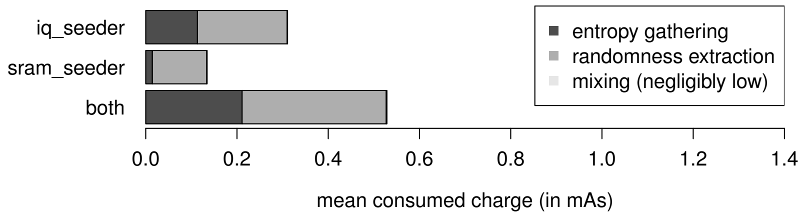


Figure 3.9: Mean energy consumption of self-seeding

entropy by reading the SRAM, which costs very little in terms of energy. Also, the `sram_seeder` consumes less energy for randomness extraction.

3.7 Summary

In this chapter, we have made two main contributions. First, we have proposed a superior method for extracting seeds from power-up SRAM states. Currently proposed methods for extracting seeds from power-up SRAM states (i) read a region of the SRAM at boot time and (ii) pass the result to a deterministic hash or extractor function. By contrast, our method is based on the novel approach of taking multiple samples of power-up SRAM states, as is supported by certain hardware platforms. Compared to current methods for extracting seeds from power-up SRAM states, ours has the advantages of being (i) both information-theoretically secure and practical, (ii) resistant to freezing attacks, (iii) resistant to burn-in effects, as well as (iv) resistant to data remanence. Second, we have presented the design of a self-seeding CSPRNG for IoT devices. Our CSPRNG can mix multiple entropy sources, provides forward and backward security if configured to reseed itself regularly at runtime, and efficiently expands seeds into a stream of cryptographic random numbers via OFB-AES-128.

Chapter 4

Denial-of-Sleep-Resilient Session Key Establishment

This chapter details AKES - our denial-of-sleep-resilient protocol for establishing session keys among neighboring IEEE 802.15.4 nodes. At first, we make a case for session keys and point out limitations of current protocols for establishing session keys among neighboring IEEE 802.15.4 nodes. Then, we introduce related work on which AKES is based. Next, we detail AKES, outline denial-of-sleep attacks against AKES, and propose two sets of corresponding denial-of-sleep defenses. Further, we outline our implementation of AKES and quantify the overhead due to AKES. Also, we give a comparative evaluation of AKES' resilience to denial-of-sleep attacks when using our two different sets of denial-of-sleep defenses. Lastly, we wrap up with completing our discussion of related work.

4.1 Extended Problem Statement

IEEE 802.15.4 security leaves key management unspecified. *Key management* is an umbrella term that subsumes four components, namely key establishment, key refreshment, key revocation, and rekeying [145]. Among these components, *key establishment* is a scheme as per which two or more parties agree on a shared secret key [145]. A special way of realizing key establishment is *key predistribution*, where the established keys are entirely predetermined by preloaded keying material [145]. Actually, key predistribution is a tempting choice for realizing key establishment in IEEE 802.15.4 networks due to its low, if any, communication and processing overhead. Yet, as for key predistribution, the validity of the established keys never expires. This incurs two conflicts with the frame counter-based replay protection of IEEE 802.15.4 security.

The first conflict concerns topology changes. Recall that IEEE 802.15.4 security requires each node to keep track of the frame counters of its neighbors so as to detect replayed frames. Normally, this anti-replay data is kept in RAM, but, in bigger IEEE 802.15.4 networks with mobile nodes, not all anti-replay data may fit within the constrained RAM of IEEE 802.15.4 nodes over time. Hence, some anti-replay data may have to be swapped to non-volatile memory over time. However, swapping is problematic because the only non-volatile memory on most IEEE 802.15.4 nodes is flash memory, which is slow, energy consuming, as well as prone to wear [117]. After all, swapping may also become necessary in bigger IEEE 802.15.4 networks without mobile nodes if an attacker tunnels the traffic between non-neighboring nodes verbatim. This can, e.g., be achieved by (i) placing two transceivers in distant parts of the victim IEEE 802.15.4 network and (ii) using these two transceivers to relay the traffic between those network parts [146]. Such an attack is known as a *hidden wormhole* [146]. In effect, non-neighboring nodes are tricked to believe they were neighbors and have to store anti-replay data about each other forever [147].

The second conflict concerns reboots. In order to prevent replay attacks after reboots, all anti-replay data must be stored across reboots, i.e., in non-volatile memory. Again, this is highly problematic because flash memory is slow, energy consuming, as well as prone to wear [117]. Besides, two more issues arise if a node's frame counter starts over after a reboot. First, this causes a nonce reuse in IEEE 802.15.4 security and, second, neighbors consider a rebooted node's frames replayed [58, 148]. Therefore, Sastry et al. considered storing a node's frame counter in non-volatile memory, too [58].

By contrast, when establishing session keys, neither of the above conflicts arises. To understand this, note that if an attacker replays an IEEE 802.15.4 frame that was secured with a session key that differs from the current one, receivers will consider the frame's CCM MIC inauthentic. Thus, session keys enable nodes to securely delete anti-replay data pertaining to expired sessions, rather than swapping this data [58]. Likewise, if a node establishes new session keys after a reboot, it will discard replayed IEEE 802.15.4 frames from previous sessions due to inauthentic CCM MICs anyway, without having to store data across reboots. Furthermore, if a node establishes new session keys after a reboot, the node's frame counter can start over after a reboot. This is because reusing CCM nonces together with different CCM keys is secure and, in the course of establishing new session keys, neighbors can be informed of a frame counter reset.

However, establishing session keys among neighboring IEEE 802.15.4 nodes is challenging because of special requirements in this context. In particular, one special requirement is to resist denial-of-sleep attacks. Another special requirement is to not just establish session keys with neighboring nodes at start up, but also at runtime so as to adapt to topology changes.

Krentz et al.'s Adaptable Pairwise Key Establishment Scheme (APKES) addresses the former of these special requirements comparatively well [10]. Essentially, APKES derives pairwise session keys from predistributed keys in the course of a three-way handshake between each pair of neighboring IEEE 802.15.4 nodes. Thus, unlike public-key cryptography (PKC)- and KDC-based alternatives [16, 20, 30, 31, 32, 33], APKES operates in a distributed manner without PKC. This already makes APKES relatively resilient to denial-of-sleep attacks since requests for session key establishment, called `HELLO`s in APKES, do neither trigger energy-consuming communication nor processing. Additionally, APKES sheds `HELLO`s when they arrive in bursts.

On the other hand, APKES has three limitations. First, APKES is not designed to survive reboots, but crashes after a reboot, as we discuss in Section 4.2.2. Second, APKES only broadcasts `HELLO`s at start up and makes no effort to discover new neighbors at runtime. Even in the broader context of establishing session keys among neighboring IEEE 802.15.4 nodes, there is only one proposal on how to discover new neighbors at runtime to the best of our knowledge [33]. That proposal, however, has problems with broadcast frames, as we detail in Section 4.6. Third, in APKES, sessions never expire. Therefore, APKES actually aggravates the need for swapping since APKES not only requires swapping anti-replay data to non-volatile memory, but also session keys.

Our AKES makes the following improvements over APKES:

- AKES survives reboots through three changes to APKES. Most importantly, AKES avoids a deadlock after reboots by processing `HELLO`s from neighbors with whom session keys were already established.

- To discover new neighbors at runtime, AKES schedules the broadcasting of `HELLO`s using Trickle [149]. By means of Trickle, AKES self-adaptively reduces the rate of `HELLO`s if the neighborhood is stable and increases the rate of `HELLO`s if the neighborhood is instable. As a result, AKES saves energy when the neighborhood is stable, yet quickly adapts to topology changes. In this regard, we also propose and evaluate two defenses against a resultant denial-of-sleep attack against AKES, where an attacker destabilizes the network topology in order to cause AKES to send more `HELLO`s and hence to consume more energy. Since such attacks manifest themselves in temporarily available or temporarily unavailable links, we refer to them as *yo-yo attacks*.

- To obviate the need for swapping data to non-volatile memory, AKES includes a mechanism for detecting inactive neighbors. If an inactive neighbor is detected, AKES deletes him along with associated data, such as session keys and anti-replay data, thereby freeing RAM.

4.2 Background

Below, we first motivate the design of APKES by reconsidering the *IOWEU dilemma*. Then, we introduce APKES, the predecessor of AKES. Also, we introduce a special technique that we evaluate for authenticating broadcast frames.

4.2.1 The IOWEU Dilemma

Ideally, a scheme for establishing keys among neighboring IEEE 802.15.4 nodes should perform well in terms of all of the properties inoculation, opaqueness, welcomingness, efficiency, and universality (IOWEU) [10, 34]:

Inoculation: Inoculation is the property of a key establishment scheme that an attacker is unable to authenticate himself to a non-compromised node with a different identity than that of a compromised node.

Opaqueness: Opaqueness is the property of a key establishment scheme that, in the event of a node compromise, only keys of links from and to the compromised node leak.

Welcomingness: Welcomingness describes to what degree a key establishment scheme supports the addition of new nodes after deployment.

Table 4.1: Key Predistribution Schemes and their Trade-offs regarding Inoculation, Opaqueness, Welcomingness, Efficiency, and Universality

Scheme	I	O	W	E	U
Single key [150]	$\times$	$\times$	$\surd$	$\surd$	$\surd$
Fully pairwise key [150]	$\surd$	$\surd$	$\surd$	$\times$	$\surd$
Random-pairwise keys [151]	$\surd$	$\surd$	$\surd$	$\surd$	$\times$
Blom's scheme [152, 153]	$(\surd)$	$(\surd)$	$\surd$	$\mathcal{O}(\lambda)$	$\surd$

Efficiency: Efficiency describes to what degree a key establishment scheme operates memory and energy efficiently.

Universality: Universality describes to what degree a key establishment scheme takes assumptions on the network topology.

Owing to the energy inefficiency of PKC- and KDC-based key establishment schemes [17], key predistribution schemes are preferable for establishing keys among neighboring IEEE 802.15.4 nodes. Table 4.1 evaluates four fundamental key predistribution schemes according to the IOWEU criteria:

Single key scheme: In the single key scheme, each node is preloaded with the same network-wide key. Hence, the single key scheme consumes very little memory. Also, since the single key scheme requires no communication at runtime, like all of the key predistribution schemes listed in Table 4.1, the single key scheme operates energy efficiently. On the other hand, the single key scheme lacks both inoculation and opaqueness. Inoculation is not fulfilled since if an attacker extracts the network-wide key from one node, he can authenticate himself as any other node and can thus inject nodes with arbitrary identities at will. Opaqueness is not fulfilled because if an attacker extracts the network-wide key from one node, the keys of all links in the network are immediately compromised, as well.

Fully pairwise key scheme: In the fully pairwise key scheme, each node is preloaded with distinct pairwise keys for communication with any other node. The fully pairwise key scheme fulfills inoculation and opaqueness. Inoculation is fulfilled since, in the event of a node compromise, the attacker can only authenticate himself as the compromised node to non-compromised nodes, thus restricting him to inject nodes into the victim network that use the identity of the compromised node. Likewise, opaqueness is fulfilled since, in the event of a node compromise, the attacker only obtains keys that pertain to links from and to the compromised node. On the other hand, the fully pairwise key scheme is very memory consuming.

Random-pairwise keys scheme: The random-pairwise keys scheme can be thought of as a memory-efficient variant of the fully pairwise key scheme. Instead of preloading pairwise keys for communication with any other node, each node is preloaded with a random subset of pairwise keys, called key ring. Furthermore, using a heuristic of Du et al., one can minimize the number of keys per key ring k so that a network of size n with minimum node degree d is connected with some high probability c [153]. Unfortunately, while being inoculated and opaque, the random-pairwise keys scheme only makes use of a fraction of all links. Hence, the random-pairwise keys scheme is only suitable for networks with high node degrees.

Blom's scheme: Blom's scheme keeps up inoculation and opaqueness as long as no more than λ nodes are compromised, as well as makes no assumptions on the network topology. Specifically, let l be the key length in bits, λ be the number of tolerable node compromises, n be the maximum number of nodes in the network, and $q \geq 2^l$ be a prime power. In a pre-deployment step, Blom's scheme randomly generates (i) a symmetric matrix $D \in \mathrm{GF}(q)^{(\lambda+1)\times(\lambda+1)}$ and (ii) a matrix $G \in \mathrm{GF}(q)^{(\lambda+1)\times n}$ with linearly independent columns. Next, each node A is preloaded with the ID_A-th column vector of $(DG)^T$ (denoted by $(DG)^T_{ID_A,-}$) and the ID_A-th row vector of G (denoted by G_{-,ID_A}), where $ID_A \in \{1, \ldots, n\}$ is an identifier of A, such as A's MAC address. At runtime, two nodes A and B calculate their pairwise key as $(DG)^T_{ID_A,-}G_{-,ID_B} = (DG)^T_{ID_B,-}G_{-,ID_A}$. For this, A and B need to exchange their row vectors of G, except when G is chosen as a Vandermonde matrix since this enables any node to recompute G entirely [153]. Consequently, if G is a Vandermonde matrix, Blom's scheme operates very energy efficiently. On the negative side, the memory consumption and processing overhead of Blom's scheme increases linearly with the tolerable number of node compromises λ.

While many advanced versions of the above key predistribution schemes exist (see [150] for a survey), the IOWEU dilemma is that there is no key predistribution scheme that fulfills all of the IOWEU criteria [10, 154].

4.2.2 APKES: Adaptable Pairwise Key Establishment Scheme

The basic approach of APKES is to derive pairwise session keys from predistributed keys in the course of a three-way handshake between each pair of neighboring IEEE 802.15.4 nodes. Furthermore, the main contribution of APKES is to decouple its three-way handshake from the key predistribution scheme in use. This enables practitioners to reuse the bulk of their code across different key predistribution schemes, as well as to select the most appropriate key predistribution scheme for their use case. APKES is split into three phases:

Optional preloading of short addresses: In the first phase, each node is optionally preloaded with an IEEE 802.15.4 short address that uniquely identifies the node in its IEEE 802.15.4 network. The motivation behind this phase is as follows. For looking up the predistributed key shared with a neighboring node, most key predistribution schemes have to be given an identifier of that node. To this end, APKES chooses to reuse IEEE 802.15.4 MAC addresses. That is, when APKES requires the predistributed key shared with a neighboring node, APKES passes the node's extended address and, if assigned, short address as parameters to the underlying key predistribution scheme. An extended address can always be passed to the underlying key predistribution scheme since extended addresses are burnt into IEEE 802.15.4 transceivers during manufacturing. However, some key predistribution may be unable to look up predistributed keys by extended addresses. For example, in Blom's scheme, the dimensions of the matrix G get too large when using extended addresses. Furthermore, while protocols for autoconfiguring short addresses exist, e.g., 6LoWPAN neighbor discovery (6LoWPAN-ND) [155], such protocols conflict with APKES. This is because such protocols randomly assign short addresses, rendering the assigned short addresses inapplicable to looking up predistributed keys. Preloading short addresses avoids this conflict.

Preloading of cryptographic material: In the second phase, each node is preloaded with cryptographic material. This cryptographic material comprises (i) a seed for generating cryptographic random numbers and (ii) keying material, which is specific to the underlying key predistribution scheme. For example, when using the fully pairwise key scheme, each node is preloaded with $n - 1$ pairwise keys, each of which is shared with the $n - 1$ other nodes in the network, including not-yet-deployed nodes.

Pairwise session key establishment: In the third phase, which is performed at start up, each node establishes pairwise session keys with its neighbors. This happens in the course of a three-way handshake by exchanging newly defined IEEE 802.15.4 command frames, named `HELLO`, `HELLOACK`, and `ACK`. Let us look into APKES' three-way handshake without going into the error handling. Initially, a node A generates a cryptographic random number R_A and broadcasts a `HELLO` containing R_A. A receiver B also generates a cryptographic random number R_B, obtains the predistributed key $K_{B,A}$ shared with A from the underlying key predistribution scheme, and stores A as a tentative neighbor. After a random back-off period, B sends a `HELLOACK` to A containing R_A and R_B. The `HELLOACK` is authenticated by using $K_{A,B}$ as CCM key. Upon reception, A also obtains the predistributed key $K_{A,B} = K_{B,A}$ from the underlying key predistribution scheme and derives the pairwise session key $K'_{A,B}$ as AES-128($K_{A,B}, R_A \| R_B$). Lastly, A stores B as a permanent neighbor and replies with an `ACK`. The `ACK` is authenticated by using $K'_{A,B}$ as CCM key. Upon reception, B also derives the pairwise session key analogously and turns the tentative neighbor A into a permanent one.

Unfortunately, APKES' three-way handshake is not designed with reboots in mind. Specifically, two problems may occur after a reboot. First, as nodes ignore `HELLO`s from senders that are stored as a neighbor already, a deadlock occurs after a reboot, unless all nodes reboot. Second, note that `HELLOACK`s are secured with predistributed keys. Consequently, if a node's frame counter starts over after a reboot, a nonce reuse may occur after a reboot.

Concerning denial-of-sleep attacks, a defense of APKES is to shed `HELLO`s as the number tentative neighbors reaches a threshold. This defense targets *`HELLO` flood attacks*, where an attacker sends `HELLO`s with high transmission powers so as to, e.g., provoke `HELLOACK` transmissions [35, 36]. We will compare APKES' and a newly proposed `HELLO` flood defense in our evaluation of AKES.

4.2.3 EBEAP: Easy Broadcast Encryption and Authentication Protocol

While pairwise session keys can serve for securing unicast frames, they can not directly be used for securing broadcast frames. To this end, Krentz et al. pro-

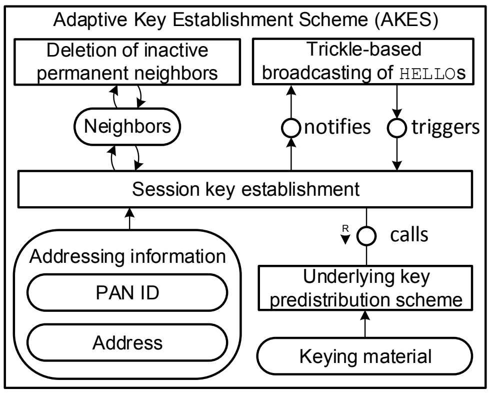

Adaptive Key Establishment Scheme (AKES)
Deletion of inactive permanent neighbors
Trickle-based broadcasting of HELLOs
Neighbors
notifies
triggers
Session key establishment
R
calls
Addressing information
PAN ID
Address
Underlying key predistribution scheme
Keying material

Figure 4.1: Design of AKES

posed the Easy Broadcast Encryption and Authentication Protocol (EBEAP) [10]. The idea of EBEAP is as follows. Suppose a node A wants to securely send a broadcast frame f. Then, A adds a frame counter to f to obtain f' and sends two broadcast frames. The first broadcast frame contains MICs $\text{MIC}_0, \text{MIC}_1, \ldots$ over f' for each of A's neighbors $B_0, B_1, \ldots$, where MIC_i is generated using the pairwise session key between A and B_i. Upon reception, neighbor B_i extracts its corresponding MIC MIC_i and buffers it in a ring buffer. For this, B_i has to have its index i in the neighbor list of A. These indices are exchanged in the course of establishing session keys. The second broadcast is f' itself. Upon reception, B_i generates a MIC over f' using its pairwise session key with A. If the generated MIC is buffered and if the frame counter of f' is fresh, B_i will accept f'. On demand, EBEAP also encrypts the payload of broadcast frames via group session keys, which then have to be established in addition to pairwise session keys. In effect, though a compromised node can decrypt the broadcast frames of its neighbors, he can never impersonate its neighbors, provided that the pairwise session keys are established in an inoculated and opaque manner.

4.3 AKES: Adaptive Key Establishment Scheme

AKES builds on the idea of leaving the underlying key predistribution scheme exchangeable and moves on to address widely open problems, namely the adaptation to topology changes and the mitigation of denial-of-sleep attacks. In this section, we detail AKES' design, which is depicted in Figure 4.1.

4.3.1 Looking up Predistributed Keys

Recall that for looking up the predistributed key shared with a neighboring node, most key predistribution schemes have to be given an identifier of that node. To this end, AKES also reuses IEEE 802.15.4 MAC addresses. That is, AKES requests the predistributed key shared with a neighboring node by passing the node's IEEE 802.15.4 MAC address to the underlying key predistribution scheme. This way of looking up predistributed keys has two advantages. First, since IEEE 802.15.4 addresses are sent along with IEEE 802.15.4 frames as part of the addressing fields anyway, there is no need to include extra node identifiers in the payload of command frames used for establishing session keys. Second, when looking up the predistributed key with a neighboring node by its IEEE 802.15.4 MAC address, it is automatically ensured that the neighboring node owns that IEEE 802.15.4 MAC address in the course of establishing session keys, provided that the underlying key predistribution is inoculated and opaque. Unfortunately, another restriction on this second advantage is that it is only ensured that a neighboring node owns either a certain short or extended address, depending on whether predistributed keys are looked up by short or extended addresses, respectively. For example, if the predistributed key shared with a neighboring node was looked up by its short address, the neighboring node could still use the extended address of another node. To fix this security weakness, AKES presumes that there is a network-wide agreement on whether short or extended addresses are used for both addressing and looking up predistributed keys. In practical terms, when, e.g., agreeing on short addresses, frames containing extended addresses are to be ignored.

4.3.2 Preloading Configuration Settings

Prior to deploying a new node, AKES requires preloading it with configuration settings. The preloaded configuration settings may comprise an IEEE 802.15.4 short address that uniquely identifies the node in its IEEE 802.15.4 network and must comprise keying material, which is specific to the underlying key predistribution scheme. The preloading of short addresses solves two problems at once. First, protocols for autoconfiguring short addresses, such as with 6LoWPAN-ND [155], conflict with AKES. This is because such protocols randomly assign short addresses, rendering the assigned short addresses inapplicable to looking up predistributed keys. Second, if not using TSCH, IEEE 802.15.4 security derives the CCM nonce of a frame from the sender's extended address and the frame's frame counter value. Thus, even if there is a network-wide agreement on using short addresses for both addressing and looking up predistributed keys, extended addresses still have to be negotiated for deriving IEEE 802.15.4-compliant CCM nonces. This is actually done by APKES and ensues a communication and memory overhead. However, if a short address uniquely identifies a node within its IEEE 802.15.4 network, it is equally secure to derive CCM nonces from short addresses.

4.3.3 Establishing Session Keys

At runtime, AKES establishes session keys in the course of a three-way handshake between each pair of neighboring IEEE 802.15.4 nodes, as shown in Figure 4.2. There are three messages involved, namely `HELLO`s, `HELLOACK`s, and `ACK`s. These messages are sent as newly defined command frames. A node A initiates the three-way handshake by broadcasting a `HELLO`. The `HELLO` contains a cryptographic random number R_A and is either authenticated with pairwise or group session keys. However, only receivers that already established session keys with A can distinguish between fresh authentic, and inauthentic or replayed `HELLO`s from A. Any receiver B that wishes to establish session keys with A stores A as a tentative neighbor and replies with a `HELLOACK` after a random back off period $T_{\text{bac}} < M_{\text{bac}}$. Like A's `HELLO`, B's `HELLOACK` also contains a cryptographic

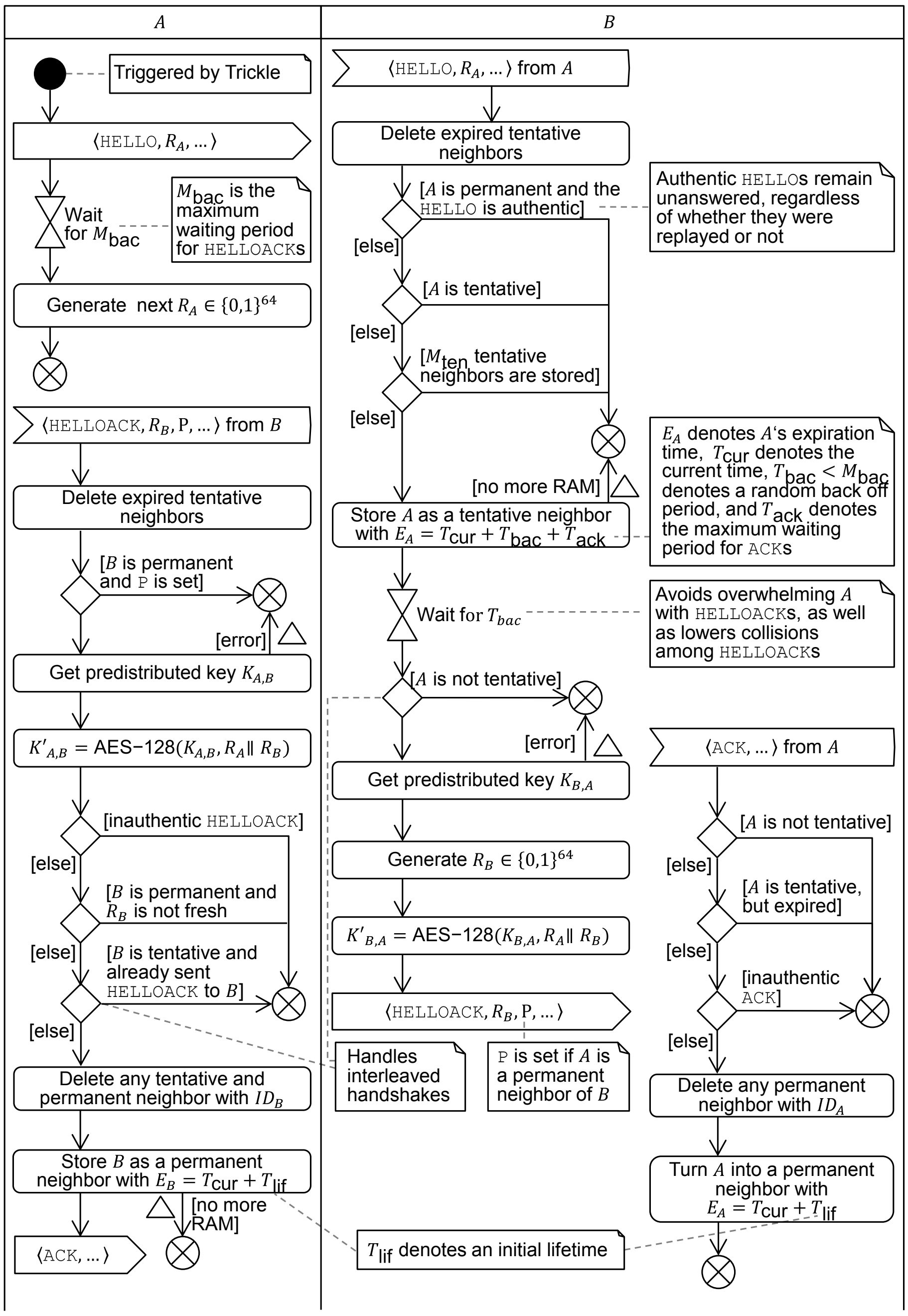

A
B
Triggered by Trickle
⟨HELLO, R_A, ...⟩
Wait for M_{bac}
M_{bac} is the maximum waiting period for HELLOACKs
Generate next $R_A \in \{0,1\}^{64}$
⟨HELLOACK, R_B, P, ...⟩ from B
Delete expired tentative neighbors
[B is permanent and P is set]
[error]
Get predistributed key $K_{A,B}$
$K'_{A,B}$ = AES−128($K_{A,B}$, $R_A \| R_B$)
[inauthentic HELLOACK]
[else]
[B is permanent and R_B is not fresh
[else]
[B is tentative and already sent HELLOACK to B]
[else]
Delete any tentative and permanent neighbor with ID_B
Store B as a permanent neighbor with $E_B = T_{\text{cur}} + T_{\text{lif}}$
[no more RAM]
⟨ACK, ...⟩
⟨HELLO, R_A, ...⟩ from A
Delete expired tentative neighbors
[A is permanent and the HELLO is authentic]
Authentic HELLOs remain unanswered, regardless of whether they were replayed or not
[else]
[A is tentative]
[else]
[M_{ten} tentative neighbors are stored]
[else]
[no more RAM]
Store A as a tentative neighbor with $E_A = T_{\text{cur}} + T_{\text{bac}} + T_{\text{ack}}$
E_A denotes A's expiration time, T_{cur} denotes the current time, $T_{\text{bac}} < M_{\text{bac}}$ denotes a random back off period, and T_{ack} denotes the maximum waiting period for ACKs
Wait for T_{bac}
Avoids overwhelming A with HELLOACKs, as well as lowers collisions among HELLOACKs
[A is not tentative]
[error]
Get predistributed key $K_{B,A}$
Generate $R_B \in \{0,1\}^{64}$
$K'_{B,A}$ = AES−128($K_{B,A}$, $R_A \| R_B$)
⟨HELLOACK, R_B, P, ...⟩
Handles interleaved handshakes
P is set if A is a permanent neighbor of B
⟨ACK, ...⟩ from A
[A is not tentative]
[else]
[A is tentative, but expired]
[else]
[inauthentic ACK]
[else]
Delete any permanent neighbor with ID_A
Turn A into a permanent neighbor with $E_A = T_{\text{cur}} + T_{\text{lif}}$
T_{lif} denotes an initial lifetime

Figure 4.2: Detailed instructions for processing HELLOs, HELLOACKs, and ACKs

random number R_B. Furthermore, the `HELLOACK` is authenticated with a pairwise session key $K'_{B,A}$ that is derived from the predistributed key $K_{B,A}$ shared between A and B, as well as the two cryptographic random numbers R_A and R_B. Upon reception of the `HELLOACK` from B, A checks its authenticity, among others. If successful, A completes the three-way handshake by sending an `ACK` to B. Analogous to B's `HELLOACK`, A's `ACK` is authenticated using the pairwise session key $K'_{A,B}$. After completing this three-way handshake, A and B store each other as permanent neighbors and can start using the established pairwise session key $K'_{B,A} = K'_{A,B}$ as CCM key.

Optionally, A and B may establish group session keys in addition to or in lieu of a pairwise session key. If so, each node generates a group session key using our CSPRNG at start up and generates a new one after each reboot. Furthermore, during the three-way handshake, B piggybacks its group session key $K_{B,*}$ encrypted with $K'_{B,A}$ on the `HELLOACK`. Likewise, the `ACK` carries A's group session key $K_{A,*}$ encrypted with $K'_{A,B}$. Also, A and B may piggyback each other's index in their neighbor lists on the `HELLOACK` and `ACK`, respectively. These indices are, e.g., required to implement EBEAP.

A crucial change to APKES is that `HELLO`s from permanent neighbors are processed in order to survive reboots. Let us, e.g., consider the case that A reboots. Then, a former permanent neighbor B will eventually receive an inauthentic `HELLO` from A. This will cause B to store A as a tentative neighbor, but, at the same time, to keep A as a permanent neighbor. As the `ACK` from A arrives, B will delete the former permanent neighbor A and turn the tentative neighbor A into a permanent one. This effectively starts a new session between A and B. The more common case is that B receives fresh authentic `HELLO`s from A, which are silently discarded. However, a subtlety arises if a node B receives an inauthentic `HELLO` from a permanent neighbor A, although A did not reboot (e.g, due to missing EBEAP's first broadcast frame). In this case, B will also reply with a `HELLOACK`, but with the `P` flag set since A is currently stored as a permanent neighbor by B. Upon reception, A will discard `HELLOACK`s from permanent neighbors with the `P` flag set right away, thereby avoiding to start a new session unnecessarily.

Two further changes to APKES are also crucial for surviving reboots. First, whereas APKES secures `HELLOACK`s with predistributed keys, AKES secures `HELLOACK`s with pairwise session keys. This avoids resuing the same nonce in conjunction with the same key after reboots. Second, whereas APKES detects replayed `HELLOACK`s via frame counters, AKES uses an internal mechanism for this. Specifically, if the sender B of an authentic `HELLOACK` is stored as a permanent neighbor, AKES checks whether the contained challenge R_B differs from its previous one. This can be implemented either by caching challenges or, more efficiently, by checking whether the newly generated pairwise session key would match the current one. Either way, the result is that `HELLOACK`s from a rebooted node will not be considered replayed due to a low frame counter value.

The replay of `ACK`s, on the other hand, is pointless anyway 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 (i) injecting a `HELLO` and (ii) replaying an `ACK`. However, the cryptographic random number within the `HELLOACK` will have changed and hence the derived pairwise session key. Thus, the MIC of the replayed `ACK` will turn out inauthentic, causing the replayed `ACK` to be rejected.

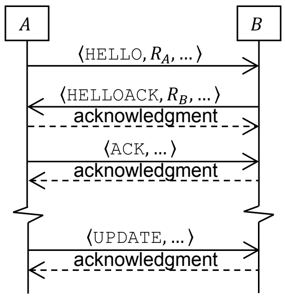


Figure 4.3: Interaction for checking if a permanent neighbor is still in range

4.3.4 Adapting to Topology Changes

Owing to mobile nodes and changing surroundings, nodes can get out of range and new ones can come into range. In the following, we detail how AKES adapts to such topology changes by continuously deleting inactive permanent neighbors and continuously "Trickling" `HELLO`s to discover new neighbors.

4.3.4.1 Deletion of Inactive Permanent Neighbors

When a permanent neighbor expires, AKES checks if that neighbor is still in range via a newly defined command frame, named `UPDATE`. For example, in Figure 4.3, A sends an `UPDATE` to B to check if B is still in range. Upon reception, B replies with an authenticated acknowledgment frame. Finally, as A receives a fresh authentic acknowledgment frame from B in response, A prolongs B's expiration time. Otherwise, if B does not reply with a fresh authentic acknowledgment frame after A retransmitted the `UPDATE` a configurable number of times, A eventually gives up and deletes B. Note that when B receives a fresh authentic `UPDATE` from A, B prolongs A's expiration time, too. Prolonging a permanent neighbor's expiration time is also done implicitly when receiving any fresh authentic frame, thereby reducing the number of explicit `UPDATE`s.

4.3.4.2 Trickle-Based Broadcasting of `HELLO`s

For scheduling the broadcasting of `HELLO`s, AKES adopts the Trickle algorithm [149]. This algorithm takes three parameters:

$I_{\mathbf{min}}$: the minimum interval duration

$I_{\mathbf{max}}$: the maximum interval duration

k: the redundancy constant

and maintains three variables:

c: a counter

I: the current interval duration

t: an instant within the second half of the current interval

Initially, Trickle sets c to 0, I to a random duration within the range $[I_{\min}, I_{\max}]$, and t to a random instant within the range $[\frac{I}{2}, I)$. Up to time t, Trickle increments c upon receiving a consistent broadcast, where the notion of consistent is left application specific. At time t, Trickle broadcasts if $c < k$. What Trickle broadcasts is also left application specific. At the end of the current interval I, Trickle starts a new interval with $c = 0$, $I = \min\{I \times 2, I_{\max}\}$, and a random instant $t \in [\frac{I}{2}, I)$. A new interval also begins immediately if a reset is issued - unless I already is at its minimum $I_{\min}$. If a new interval begins as a result of a reset, Trickle starts over with $c = 0$, $I = I_{\min}$, and a random instant $t \in [\frac{I_{\min}}{2}, I_{\min})$. Altogether, Trickle reduces its broadcast rate exponentially while consistency is achieved and increases its broadcast rate when Trickle is reset. Further energy is saved by suppressing broadcasts if $c < k$.

AKES tailors Trickle to broadcast `HELLO`s as follows. By default, AKES sets $I_{\min} = \max\{30\text{s}, 2 \times M_{\text{bac}} + 1\text{s}\}$, $I_{\max} = I_{\min} \times 2^8$, and $k = 2$. Setting $I_{\min}$ to $\max\{30\text{s}, 2 \times M_{\text{bac}} + 1\text{s}\}$ avoids broadcasting another `HELLO` while still waiting for `HELLOACK`s. AKES resets Trickle if $\max\{\lfloor\frac{n}{4}\rfloor, 1\}$ permanent neighbors were added during the current interval, where n is the current number of permanent neighbors. The motivation behind this reset rule is that new permanent neighbors are a good indicator of far-reaching topology changes. Yet, when establishing a new session key with a permanent neighbor (which, e.g., happens after a permanent neighbor rebooted), AKES does not count this permanent neighbor as new. As consistent broadcasts, AKES considers fresh authentic `HELLO`s unless they originate from a permanent neighbor that already sent a fresh authentic `HELLO` since the last time the receiver broadcasted a `HELLO`. The basic motivation behind this definition of consistent transmissions is that fresh authentic `HELLO`s from permanent neighbors indicate a stable topology. Additionally, this definition factors in that the uncertainty about the consistency of the network topology grows when just hearing fresh authentic `HELLO`s from one permanent neighbor or from a small group of permanent neighbors. In particular, if one permanent neighbor sends fresh authentic `HELLO`s at a high rate, the permanent neighbor may have reset Trickle as a result of far-reaching topology changes, which may also affect a receiver's neighborhood. To speed up joining a network, AKES broadcasts one `HELLO` at start up and resets Trickle thereafter.

4.3.5 Defending against Denial-of-Sleep Attacks

There are two kinds of denial-of-sleep attacks against AKES, as shown in Figure 4.4:

HELLO flood attacks: In a `HELLO` flood attack, on the one hand, an *external attacker*, i.e., an attacker without access to any cryptographic key [156], injects a `HELLO` with a high transmission power, thereby causing each receiver to store a tentative neighbor with the address that is pretended to be the source address of the `HELLO` [36]. More severely, receivers will also send a `HELLOACK` after a random back off period [35]. This consumes a good amount of energy, especially if the external attacker does not acknowledge the reception of `HELLOACK`s because victim nodes will then retransmit `HELLOACK`s multiple times. Moreover, as an *internal attacker*, i.e., an attacker who gained access to one or more cryptographic keys [156], a method for aggravating `HELLO` flood attacks against AKES is to reply

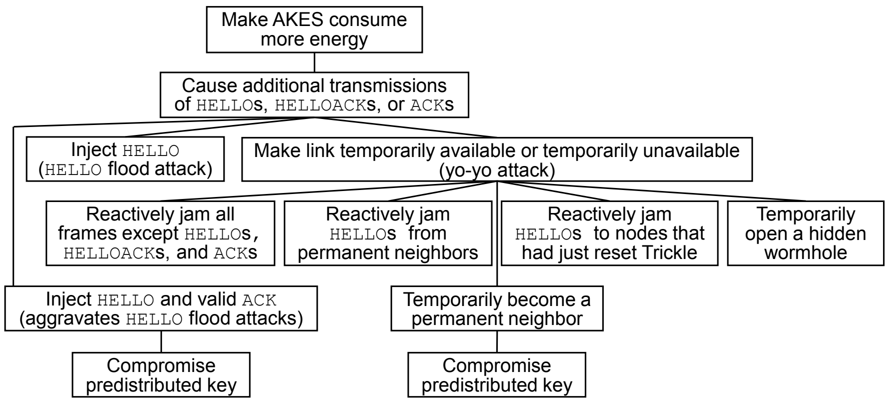

Make AKES consume more energy
Cause additional transmissions of HELLOs, HELLOACKs, or ACKs
Inject HELLO (HELLO flood attack)
Make link temporarily available or temporarily unavailable (yo-yo attack)
Reactively jam all frames except HELLOs, HELLOACKs, and ACKs
Reactively jam HELLOs from permanent neighbors
Reactively jam HELLOs to nodes that had just reset Trickle
Temporarily open a hidden wormhole
Inject HELLO and valid ACK (aggravates HELLO flood attacks)
Temporarily become a permanent neighbor
Compromise predistributed key
Compromise predistributed key

Figure 4.4: Attack tree of denial-of-sleep attacks against AKES

to HELLOACKs with valid ACKs. Normally, if a victim node has reached its maximum number of tentative neighbors, AKES will no longer respond to HELLOs until one of the tentative neighbors expires. Yet, upon reception of a valid ACK, AKES turns a tentative neighbor into a permanent one, allowing internal attackers to launch HELLO flood attacks at a higher rate.

Yo-yo attacks: In a yo-yo attack, on the other hand, an attacker makes links temporarily available or temporarily unavailable so as to provoke more attempts to establish session keys, reestablishments of session keys, or both. As an external attacker there appear to be two avenues for carrying out a yo-yo attack. First, an external attacker can jam certain frames so as to cause bit errors and hence prevent their successful reception, aka reactive jamming [38]. For example, if an attacker jams all frames except HELLOs, HELLOACKs, and ACKs, AKES will delete permanent neighbors and later reestablish session keys. As a result, victim nodes send additional HELLOACKs and ACKs. Moreover, victim nodes potentially reset Trickle due to adding permanent neighbors. Thus, this example yo-yo attack may ensue more HELLO transmissions, too. Another reactive jamming-based yo-yo attack is to prevent a victim node from receiving HELLOs from permanent neighbors, which effectively disables AKES' suppression of HELLOs. A last reactive jamming-based yo-yo attack is to prevent nodes that just reset Trickle from receiving HELLOs so as to delay session key establishment and potentially cause additional Trickle resets. Second, an external attacker can also temporarily open a hidden wormhole so as to trick distant nodes into believing they were neighbors [146], thereby causing AKES to establish session keys between them and potentially reset Trickle, too. Moreover, an internal attacker with access to one or more predistributed keys can become a permanent neighbor of any node by injecting valid ACKs and HELLOACKs. This provokes additional HELLOACK and ACK transmissions, and potentially Trickle resets, as well. Yo-yo attacks by an internal attacker aggravate when the internal attacker does not reply to UPDATEs or other frames after sending its valid HELLOACK or ACK. This is because AKES will delete the seemingly inactive permanent neighbor, thus allowing the internal attacker to become a permanent neighbor thereafter again.

To protect AKES against `HELLO` flood and yo-yo attacks, we propose and evaluate two sets of denial-of-sleep defenses, named intrinsic and LBC-based denial-of-sleep defenses. The approach of the intrinsic denial-of-sleep defenses is to tune AKES' existing parameters to counter `HELLO` flood and yo-yo attacks. Unfortunately, as we argue shortly, the intrinsic denial-of-sleep defenses require a trade-off between denial-of-sleep resilience and the speed at which AKES adapts to topology changes. This observation gives rise to complementary LBC-based denial-of-sleep defenses. Indeed, the LBC-based denial-of-sleep defenses shall turn out to significantly accelerate AKES' reaction to topology changes, without incurring much overhead nor sacrificing on security.

4.3.5.1 Intrinsic Denial-of-Sleep Defenses

In the following, we first explain AKES' intrinsic denial-of-sleep defenses.

Defense against `HELLO` Flood Attacks Recall that AKES has the following parameters, as shown in Figure 4.2:

$M_{\mathbf{ten}}$**:** the maximum number of tentative neighbors

$M_{\mathbf{bac}}$**:** the maximum back off period of `HELLOACK`s

$T_{\mathbf{ack}}$**:** the maximum waiting period for `ACK`s

Hence, by injecting `HELLO`s with random source addresses, external attackers can cause AKES to send `HELLOACK`s at a mean rate of

$$\frac{M_{\text{ten}}}{\frac{1}{2}M_{\text{bac}} + T_{\text{ack}}} \tag{4.1}$$

at most, not counting retransmissions separately. Unfortunately, this rate can not be tuned without affecting the speed at which AKES reacts to topology changes. Specifically, lowering M_{ten} reduces the number of neighbors that AKES can add in parallel. Increasing M_{bac} delays session key establishment. Lastly, increasing T_{ack} entails the following issue. Suppose that a node A broadcasted a `HELLO` and that session key establishment with a neighbor B did not complete due to a missed `HELLOACK` or `ACK`. Now, if A sends another `HELLO` before B may delete A from its list of tentative neighbors, B will ignore A's `HELLO`.

Moreover, internal attackers may be able to cause a victim node to send `HELLOACK`s at a higher rate than in Equation (4.1). This is because, once an internal attacker obtained the predistributed key shared between a victim node and another node, he can (i) establish session keys with the victim node and (ii) reestablish session keys with the victim node over and over, with the only delay being $\frac{1}{2}M_{\text{bac}}$ on average. Concretely, if an internal attacker controls k out of a victim node's $n \geq k$ permanent neighbors, the internal attacker can force the victim node to send `HELLOACK`s at a mean rate of

$$\frac{k}{\frac{1}{2}M_{\text{bac}}} \tag{4.2}$$

at most, not counting retransmissions separately. Thus, to also withstand internal attackers, M_{bac} has to be chosen long, delaying session key establishment.

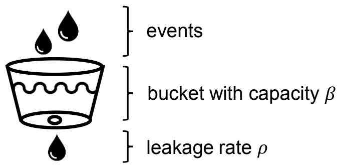


Figure 4.5: Intuition behind leaky bucket counters

Defense against Yo-Yo Attacks Recall that AKES does not delete inactive permanent neighbors immediately, but only after a hysteresis T_{lif}. This rate-limits reestablishments of session keys with the same permanent neighbor. Yet, to counter yo-yo attacks effectively, T_{lif} must be chosen very long. This is because an external attacker may set up multiple hidden wormholes or reactively jam on several links. Hence, victim nodes may not always reestablish session keys with the same permanent neighbor during a yo-yo attack, but with different permanent neighbors over and over. In such occasions, AKES will still issue many Trickle resets, unless T_{lif} is chosen very long so that AKES either abstains from reestablishing session keys with any of a whole set of permanent neighbors for long or cancels reestablishments of session keys because of running out of RAM.

On the other hand, while extending T_{lif} increases AKES' resilience to yo-yo attacks, it also defers the deletion of inactive permanent neighbors and hence the freeing of allocated RAM. Moreover, if much RAM is allocated for storing inactive permanent neighbors, this can deprive AKES of establishing session keys with actual neighbors, decelerating AKES' adaption to topology changes.

4.3.5.2 LBC-based Denial-of-Sleep Defenses

Since the intrinsic denial-of-sleep defenses require a trade-off between denial-of-sleep resilience and the speed at which AKES adapts to topology changes, we propose complementary LBC-based denial-of-sleep defenses below. The intuition behind an LBC is a bucket with a hole in it, as shown in Figure 4.5 [59]. Events drop into the bucket and increase its filling level. It is possible to associate different events with different drop sizes. As the bucket has a hole, its filling level decreases as long as there is water in it. Pressure is neglected so that the filling level of the bucket decreases with a constant rate ρ. In our use case, we want to avoid that the bucket overflows by taking appropriate actions beforehand, i.e., before the filling level exceeds the bucket's capacity β.

Defense against `HELLO` Flood Attacks As a complementary defense against `HELLO` flood attacks, we suggest that each node maintains an LBC $\text{LBC}_{\text{HELLOACK}}$ that is defined as follows:

Capacity: Its capacity β_{HELLOACK} corresponds to the maximum number of `HELLOACK`s that AKES may send in the short run, not counting retransmissions separately.

Leakage rate: Its leakage rate ρ_{HELLOACK} corresponds to the maximum rate at which AKES may send `HELLOACK`s in the long run, not counting retransmissions separately.

Table 4.2: Parameter Sets

ID	I_{min}	I_{max}	M_{ten}	M_{bac}	T_{ack}	T_{lif}
1	30s	128min	5	5s	747.5s	∞
2	601s	160min16s	5	300s	600s	∞
3	30s	128min	5	5s	5s	∞
4	30s	128min	5	5s	747.5s	5min
5	30s	128min	5	5s	747.5s	30min
6	30s	128min	5	5s	5s	5min

ID	with LBCs	β_{HELLO}	ρ_{HELLO}	β_{HELLOACK} $(= \beta_{\text{ACK}})$	ρ_{HELLOACK} $(= \rho_{\text{ACK}})$
1,2,4,5	$\times$	n/a	n/a	n/a	n/a
3,6	✓	10	$\frac{1}{300}$Hz	20	$\frac{1}{150}$Hz

Drop sizes: In the event that a `HELLOACK` is scheduled to be sent, $\text{LBC}_{\text{HELLOACK}}$ is incremented by one. When retransmitting a `HELLOACK`, however, $\text{LBC}_{\text{HELLOACK}}$ is not incremented.

Overflow prevention: AKES shall shed a received `HELLO` if $\text{LBC}_{\text{HELLOACK}}$ overflows otherwise.

An immediate benefit of our LBC-based `HELLO` flood defense is that its parameters are independent from other parameters of AKES. In fact, M_{bac} can now be shortened - but should not be zeroed to avoid overwhelming senders of `HELLO`s with `HELLOACK`s, as well as collisions among `HELLOACK`s [157]. Likewise, $T_{\texttt{ack}}$ can now be minimized according to what is the maximum waiting period between the transmission of a `HELLOACK` and the reception of the corresponding `ACK`. Finally, M_{ten} can now be configured independently from the aimed maximum rate of outgoing `HELLOACK`s under `HELLO` flood attacks.

For example, suppose we aim for a mean rate of $\frac{1}{150}$Hz of outgoing `HELLOACK`s under continuous `HELLO` flood attacks by external attackers, not counting retransmissions separately. According to Equation (4.1), a suitable configuration of the intrinsic defense against `HELLO` flood attacks is $M_{\text{ten}} = 5$, $M_{\text{bac}} = 5\text{s}$, and $T_{\text{ack}} = 747.5\text{s}$. However, in order to also withstand `HELLO` flood attacks by, at least, one attacker-controlled permanent neighbor, choosing $M_{\text{ten}} = 5$, $M_{\text{bac}} = 300\text{s}$, and $T_{\text{ack}} = 600\text{s}$ is necessary according to Equation (4.2). In either case, T_{ack} is quite long, which can lead to delays to session key establishment if frame loss occurs, as described in Section 4.3.5.1. Moreover, raising M_{bac} delays `HELLOACK`s and therefore session key establishment. By contrast, when using the LBC-based defense against `HELLO` flood attacks, a parameter set that provides an equal level of security is $M_{\text{bac}} = 5\text{s}$, $T_{\text{ack}} = 5\text{s}$, $\beta_{\text{HELLOACK}} = 20$, and $\rho_{\text{HELLOACK}} = \frac{1}{150}\text{Hz}$. Apparently, AKES reacts much faster to topology changes with these parameters. Beyond that, the LBC-based defense against `HELLO` flood attacks protects against any number of attacker-controlled permanent neighbors. Table 4.2 lists these parameter sets with IDs for future reference.

Defense against Yo-Yo Attacks Likewise, as a complementary defense against yo-yo attacks, we suggest that each node maintains two additional LBCs $\text{LBC}_{\text{HELLO}}$ and LBC_{ACK}:

Capacity: The capacity β_{HELLO} of $\text{LBC}_{\text{HELLO}}$ corresponds to the maximum number of HELLOs that AKES may broadcast in the short run.

Leakage rate: The leakage rate ρ_{HELLO} of $\text{LBC}_{\text{HELLO}}$ corresponds to the maximum rate at which AKES may broadcast HELLOs in the long run.

Drop sizes: In the event that AKES broadcasts a HELLO, $\text{LBC}_{\text{HELLO}}$ is incremented by one. When retransmitting a HELLO, however, $\text{LBC}_{\text{HELLO}}$ is not incremented.

Overflow prevention: AKES suppresses a HELLO if $\text{LBC}_{\text{HELLO}}$ will overflow otherwise.

Capacity: The capacity β_{ACK} of LBC_{ACK} corresponds to the maximum number of ACKs that AKES may send in the short run, not counting retransmissions separately.

Leakage rate: The leakage rate ρ_{ACK} of LBC_{ACK} corresponds to the maximum rate at which AKES may send ACKs in the long run, not counting retransmissions separately.

Drop sizes: In the event that AKES sends an ACK, LBC_{ACK} is incremented by one. When retransmitting an ACK, however, LBC_{ACK} is not incremented.

Overflow prevention: AKES shall shed an incoming HELLOACK if LBC_{ACK} may overflow otherwise.

An immediate benefit of the LBC-based defense against yo-yo attacks is that T_{lif} may be minimized. Conversely, when using the intrinsic defense against yo-yo attacks, T_{lif} has to be chosen long for mitigating yo-yo attacks effectively, as discussed earlier, and as is also empirically confirmed in Section 4.5.5.

4.4 Implementation

We integrated AKES and IEEE 802.15.4 security into Contiki-NG by introducing a special `MAC` driver named `akes_mac_driver` and a special `FRAMER` named `akes_mac_framer`. The `akes_mac_driver` transparently decorates another `MAC` driver with security features and is compatible with virtually any `MAC` driver that implements an asynchronous MAC protocol, such as Contiki-NG's `csma_driver`, which implements a simplified version of the carrier sense multiple access with collision avoidance (CSMA-CA) MAC protocol of IEEE 802.15.4, or our own `MAC` drivers `contikimac_driver` and `csl_driver`, which implement ContikiMAC and CSL, respectively. The `akes_mac_framer`, on the other hand, hooks into the assembling and parsing of frames for security-related adjustments.

Our implementation can be tailored to satisfy different requirements in three ways. First, it offers several configuration parameters, most importantly configurable security levels as defined in Table 2.2. Second, it leaves the underlying key predistribution scheme exchangeable. Third, it encapsulates selected functions behind an interface called `akes_mac_strategy` so as to enable different flavors of IEEE 802.15.4 security. For example, we implemented (i) an `akes_mac_strategy` called `noncoresec_strategy` that secures both broadcast and unicast frames using group session keys, (ii) an `akes_mac_strategy` called `coresec_strategy` that secures broadcast frames using EBEAP and unicast frames using pairwise session keys, and (iii) an `akes_mac_strategy` called `unicast_strategy` that secures unicast frames using pairwise session keys, authenticates `HELLO`s by including one CCM MIC for each permanent neighbor (each of which is generated using the respective pairwise session key), and sends other broadcast frames as unicast frames to each permanent neighbor, one after another.

As for LBCs, we added an abstract data type for the creation and management of LBCs to Contiki-NG. For implementing the leaking of LBCs, we decided to lazily update the filling levels of LBCs upon enquiries. This not only reduces the processing overhead, but also avoids depriving an IoT device from sleeping just for updating filling levels. As a result, our implementation of LBCs is lightweight in terms of energy consumption. Also, our implementation of LBCs is lightweight in terms of RAM consumption. In fact, the RAM consumption per LBC is merely 12 bytes when using the CC2538 SoC as target platform.

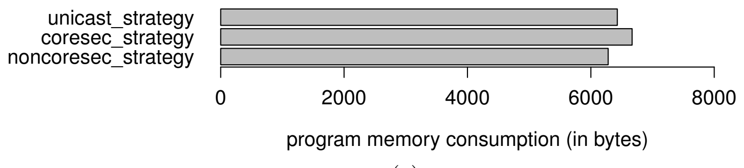
4.5 Evaluation

In the following, we first determine the memory and energy consumption of our implementation in various configurations. Then, we showcase the efficacy of AKES' Trickle-based scheduling of `HELLO`s. Finally, we empirically compare the intrinsic and the LBC-based denial-of-sleep defenses concerning protection and the resultant speed at which AKES adapts to topology changes.

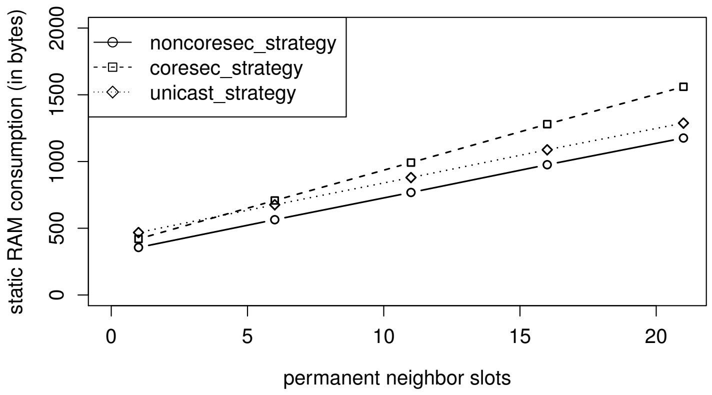
4.5.1 Memory Consumption

The program memory and RAM consumption of our implementation was determined as follows. At first, as a baseline for comparison, the program memory and RAM consumption of ContikiMAC without MAC layer security was measured using the tool `arm-none-eabi-size` and the CC2538 SoC as target platform. Then, the program memory and RAM overhead due to AKES, the `noncoresec_strategy`, as well as dependent modules such as our CSPRNG, was determined. Next, these measurements were repeated with the `coresec_strategy`, the `unicast_strategy`, as well as with different numbers of permanent neighbor slots. Throughout, short addresses, Security Level 6 (see Table 2.2), and Parameter Set 6 were used (see Table 4.2).

As shown in Figure 4.6, the program memory and RAM overhead due to AKES is significant. Only a little amount of program memory can be saved by using the least secure `akes_mac_strategy`, namely the `noncoresec_strategy`. Likewise, the basic RAM consumption differs only marginally between the different `akes_mac_strategy`s. That said, practitioners can save RAM by reducing the number of permanent neighbor slots. When using the `coresec_strategy`, the RAM consumption per permanent neighbor slot is higher since EBEAP necessitates storing both a pairwise session key and a group session key per permanent neighbor if the Security Level is ≥ 5.



(a)



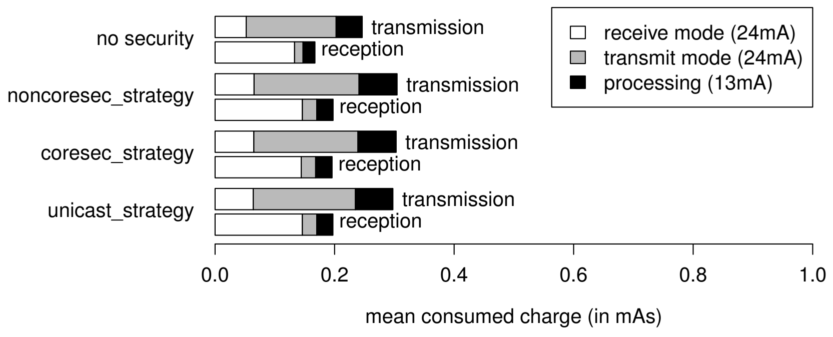
(b)

Figure 4.6: Memory footprint of AKES on CC2538 SoCs

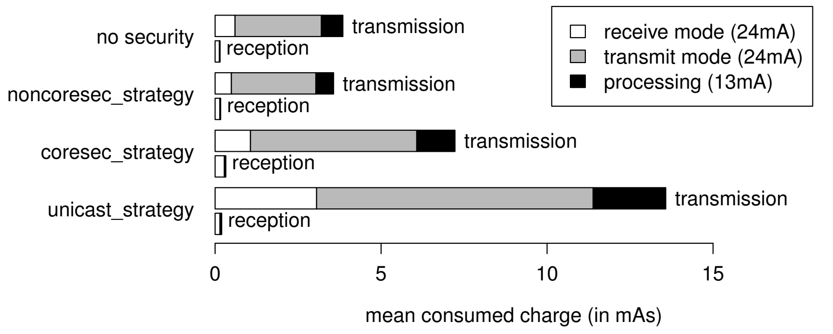
4.5.2 Energy Consumption

To measure the energy overhead during receptions and transmissions due to securing frames, a network of 10 nearby OpenMotes was set up. The OpenMotes ran our implementation of ContikiMAC and, after learning the wake-up time of each of the 9 other OpenMotes, began to send unicast frames containing 64 bytes of payload to each other with a random delay of 0 - 4.5min in between. In the background, each OpenMote ran Contiki-NG's tool "Energest" to trace the energy consumed for transmitting and receiving these unicast frames [158]. Initially, this experiment was conducted without IEEE 802.15.4 security enabled and subsequently with each of our three `akes_mac_strategy`s. Also, this experiment was repeated with broadcast frames instead of unicast frames. Throughout, ContikiMAC's wake-up interval was set to 125ms and all upper-layer protocols were disabled. Furthermore, like in the previous experiment, short addresses, Security Level 6, and Parameter Set 6 were used.

The results are shown in Figure 4.7. As for unicast receptions, the energy consumption increases slightly when enabling any of our `akes_mac_strategy`s. This is mostly because receiving secured unicast frames and transmitting secured acknowledgment frames takes longer. Likewise, as for unicast transmissions, there is also a general increase in energy consumption due to the increased security-related per-frame overhead. As for broadcast receptions, the energy consumption increases also slightly when using the `noncoresec_strategy` or the `unicast_strategy`. However, when using the `coresec_strategy` the energy consumption per broadcast reception approximately doubles because receivers need to receive EBEAP's additional broadcast frame. Similarly, as for broadcast transmissions, using the `coresec_strategy` causes the energy consumption to approximately double as compared to using no security since senders need to



(a)



(b)

Figure 4.7: Energy consumption per transmission and reception of (a) a unicast frame and (b) a broadcast frame containing 64 bytes of payload

send EBEAP's additional broadcast frame. Moreover, when using the `unicast_strategy`, senders need to send broadcast frames as multiple unicast frames. This consumes much energy despite of ContikiMAC's phase-lock optimization. Surprisingly, when using the `noncoresec_strategy`, the energy consumption per broadcast transmission decreases as compared to using no security. This comes down to a peculiarity of ContikiMAC, which repeatedly transmits broadcast frames for an entire wake-up interval plus once to cover corner cases. Thus, it may happen that longer broadcast frames are strobed less often than shorter ones, which then leads to a lower energy consumption. Altogether, in terms of energy consumption, the `noncoresec_strategy` is the best choice.

4.5.3 Communication Overhead

Asynchronous MAC protocols generally consume a considerable amount of energy during broadcast transmissions. Therefore, we put special emphasis on reducing the number of outgoing `HELLO`s in the design of AKES. To showcase the efficacy of the Trickle-based scheduling of `HELLO`s, Contiki-NG's network simulator Cooja was used [159]. The simulated network is shown in Figure 4.8. It comprised 25 nodes, each of which booted at a random point in time during the first 30 virtual minutes. At runtime, each node logged its number of outgoing `HELLO`s. Throughout, Parameter Set 6 was used.

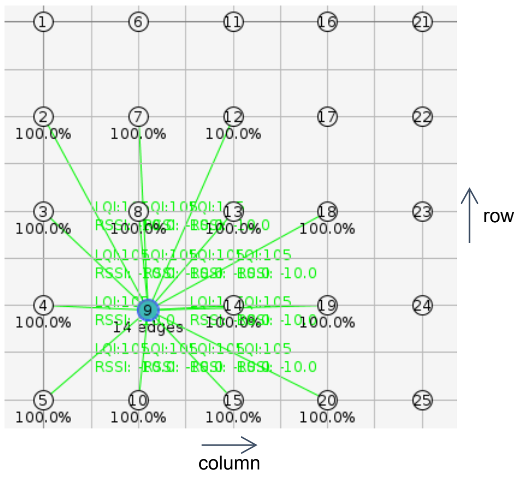


Figure 4.8: Network topology of most of our Cooja simulations in this chapter

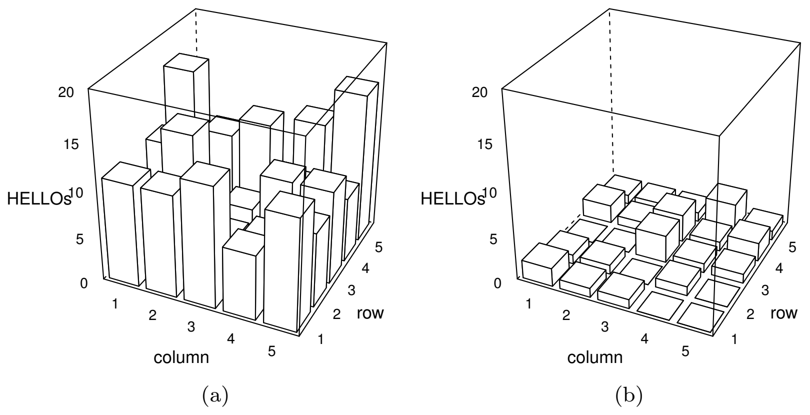


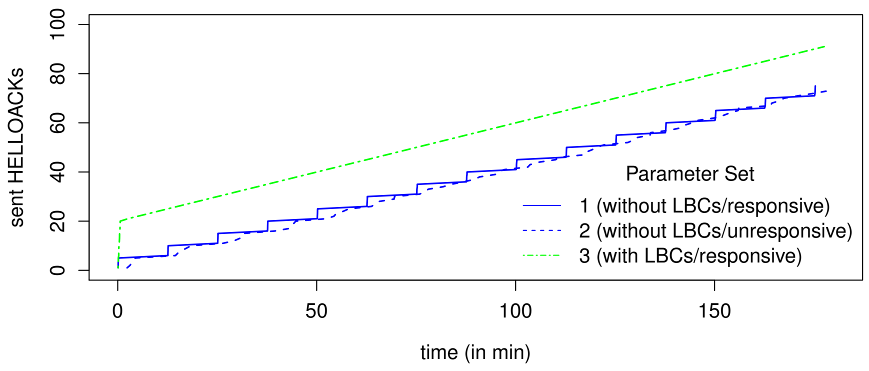
Figure 4.9: Number of sent HELLOs between (a) hour 0 and 6 (b) hour 6 and 12

Figure 4.9a shows the number of outgoing HELLOs after 6 virtual hours. Afterwards, each node only sent 0-3 HELLOs within 6 virtual hours, as shown in Figure 4.9b. Of course, even more energy-efficient configurations are possible, but at the cost of a slower adaptation to topology changes.

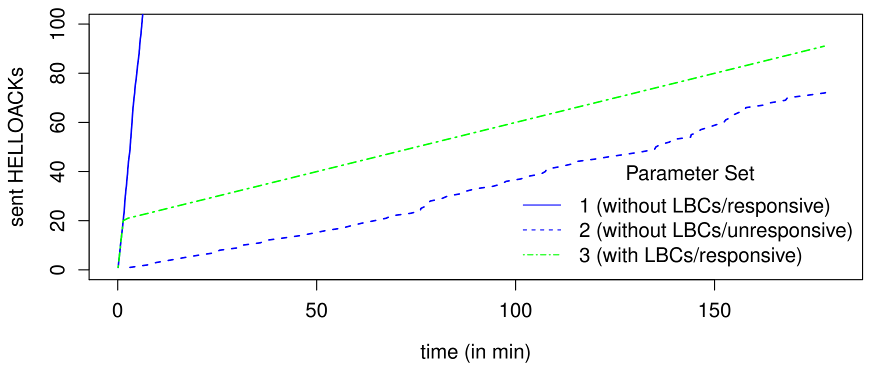
4.5.4 Protection against HELLO Flood Attacks

4.5.4.1 HELLO Flood Attacks by External Attackers

To compare the protection against external attackers of the intrinsic and the LBC-based HELLO flood defense, the following experiment was conducted. A Cooja simulation with two nodes was run for three virtual hours. The first node acted as an external attacker who continuously injected HELLOs with random source addresses at the rate of 1Hz. The second node acted as a victim and initially ran AKES with Parameter Set 1 shown in Table 4.2. This simulation



(a)



(b)

Figure 4.10: Sent HELLOACKs under continuous HELLO flood attacks by (a) an external attacker and (b) an internal attacker controlling one node

was then rerun using Parameter Set 2 and 3. In each run, the victim node logged its number of sent HELLOACKs. Throughout, the frame loss was 0%.

Figure 4.10a shows the results. At the beginning, the victim node answers five HELLOs in a row when using the intrinsic HELLO flood defense. This is because the intrinsic HELLO flood defense only comes into play as the number of tentative neighbors reaches the threshold $M_{\text{ten}} = 5$. Similarly, when using the LBC-based HELLO flood defense, the bucket initially is empty, thus causing the victim node to answer the first $\beta_{\text{HELLOACK}} = 20$ HELLOs. Then, the victim node gradually answers further HELLOs since the bucket leaks at the rate of $\rho_{\text{HELLOACK}} = \frac{1}{150}$Hz. When using the intrinsic HELLO flood defense, the victim node answers further HELLOs in bursts since all initially stored tentative neighbors expire closely after one another. As previously conjectured, both HELLO flood defenses restrict the rate of outgoing HELLOACKs to $\frac{1}{150}$Hz in the long run.

4.5.4.2 HELLO Flood Attacks by Internal Attackers

To compare the protection against one attacker-controlled permanent neighbor of the intrinsic and the LBC-based HELLO flood defense, the following experiment was conducted. Again, a Cooja simulation was run for three virtual hours in which one node acted as an attacker-controlled node and another node acted as a victim. In three successive run, the victim node used (i) Parameter Set 1, (ii) Parameter Set 2, and (iii) Parameter Set 3. During all runs, the victim node logged its number of sent `HELLOACK`s, the frame loss was 0%, and the attacker-controlled node operated as follows. At first, the attacker-controlled node established session keys with the victim node. Then, the attacker-controlled node broadcasted inauthentic `HELLO`s at the rate of 1Hz. In the case that the victim node replied with a `HELLOACK`, the attacker-controlled node completed the three-way handshake by sending a valid `ACK` in response.

As shown in Figure 4.10b, the results differ from the previous experiment as far as the intrinsic `HELLO` flood defense is concerned. Particularly, when using Parameter Set 1, the victim node ends up with 2993 sent `HELLOACK`s after three virtual hours. This disastrous result arises because the attacker-controlled node reestablishes session keys with the victim node again and again with a short waiting period in between. This waiting period takes $\frac{1}{2}M_{\text{bac}} =$ 2.5s on average. Only when tuning M_{bac} appropriately, the intrinsic `HELLO` flood defense approaches the aimed maximum rate of one outgoing `HELLOACK` per 150s. However, if more than one attacker-controlled permanent neighbors shows up, M_{bac} has to be extended further. On the other hand, the LBC-based `HELLO` flood defense protects against any number of attacker-controlled permanent neighbors, even without configuration changes.

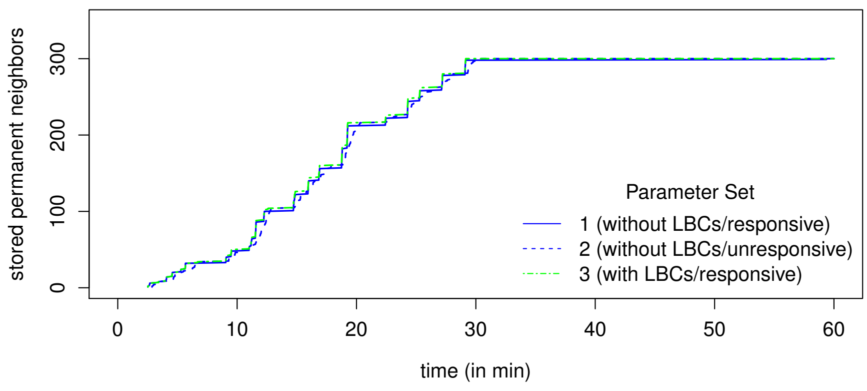
4.5.4.3 Speed of Adapting to Topology Changes

To compare AKES' speed of establishing session keys when using either the intrinsic or the LBC-based `HELLO` flood defense, another set of Cooja simulations was run. Throughout, the topology shown in Figure 4.8 was used, where every of the 25 nodes logged its number of permanent neighbors and booted at a pseudo-random point in time during the first 30 virtual minutes. In three successive runs over one virtual hour, all nodes were configured to use (i) Parameter Set 1, (ii) Parameter Set 2, and (iii) Parameter Set 3. These three runs were also repeated with (i) a frame loss of 10% (instead of 0%) without enabling retransmissions and (ii) a frame loss of 10% with three retransmissions at most.

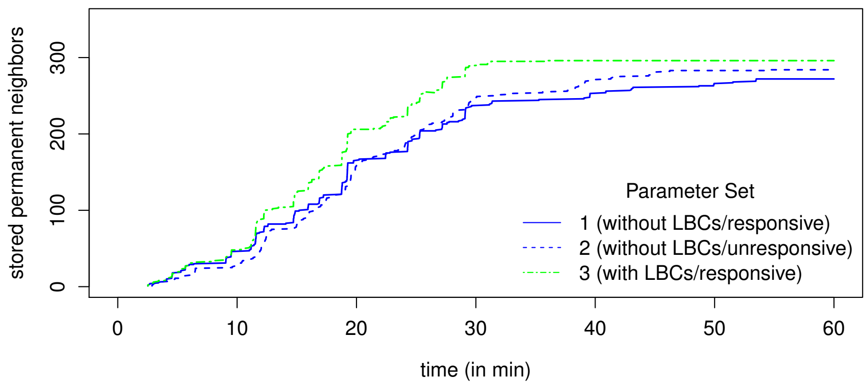
Figure 4.11a shows the results with frame loss disabled. Overall, the speed of adding permanent neighbors does not differ greatly when using either the intrinsic or the LBC-based `HELLO` flood defense. Yet, the intrinsic `HELLO` flood defense with Parameter Set 2 lags behind since it delays `HELLOACK`s by $\frac{1}{2}M_{\text{bac}} =$ 150s on average. Moreover, in the case of a frame loss of 10%, the intrinsic `HELLO` flood defense becomes noticeably slower, as shown in Figure 4.11b. This is because of an issue we mentioned earlier - if an `ACK` or `HELLOACK` is missed, tentative neighbors are kept for long if T_{ack} is long, thus causing incoming `HELLO`s from tentative neighbors to be ignored for a long period of time. A remedy to this issue is to retransmit frames, as shown in Figure 4.11c. Despite this, the intrinsic `HELLO` flood defense with Parameter Set 2 remains slower at adding permanent neighbors because it delays `HELLOACK`s.

4.5.4.4 Discussion

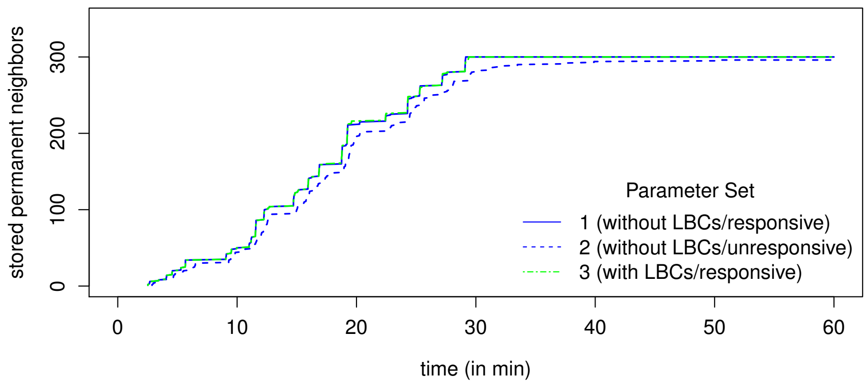
The intrinsic `HELLO` flood defense comes at the cost of extending both T_{ack} and M_{bac}. Extending T_{ack}, on the one hand, may be acceptable if retransmissions are enabled, as shown in Figure 4.11c. Extending M_{bac}, on the other hand, severely



(a)



(b)



(c)

Figure 4.11: Speed of adding permanent neighbors with (a) 0% frame loss, (b) 10% frame loss and without retransmissions, and (c) 10% frame loss and with three retransmissions at most

deteriorates the user experience because session key establishment becomes very slow as a result. For comparison, the LBC-based `HELLO` flood defense also protects against `HELLO` flood attacks, without requiring to set T_{ack} and M_{bac} to long durations. Beyond that, the LBC-based `HELLO` flood defense protects against any number of attacker-controlled nodes and is easier to configure.

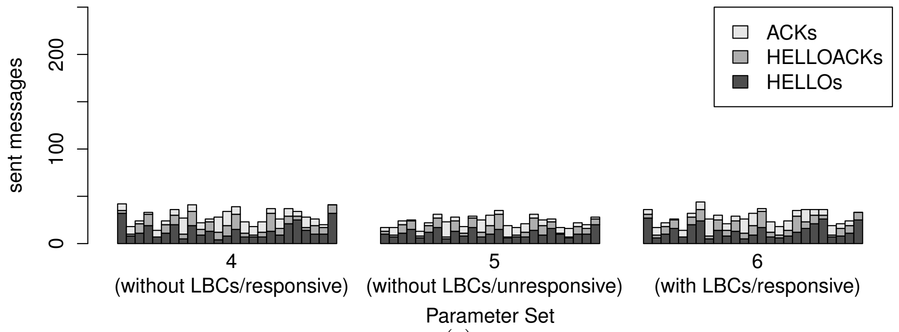
4.5.5 Protection against Yo-Yo Attacks

4.5.5.1 Yo-Yo Attacks by External Attackers

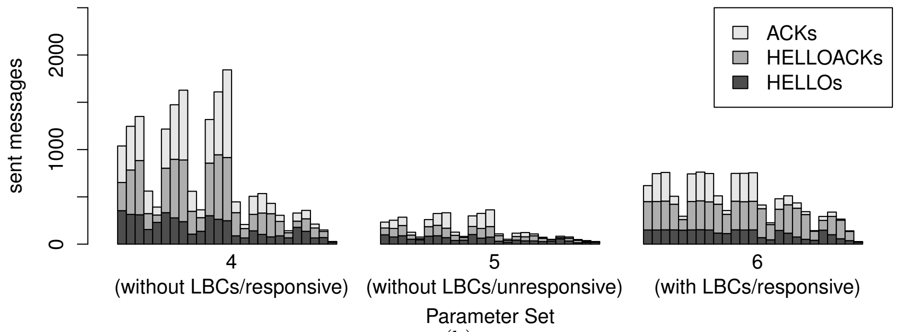
To compare the protection against reactive jamming-based yo-yo attacks of the intrinsic and the LBC-based defense against yo-yo attacks, a Cooja simulation was set up. In each run, the network shown in Figure 4.8 was simulated for 12 virtual hours, where all 25 nodes booted at a random point in time during the first 30 virtual minutes, and the frame loss was 0%. During each run, every node logged its number of sent `HELLO`s, `HELLOACK`s, and `ACK`s. In the first run, no attack was launched as a baseline for comparison. In the second run, the nodes $\{1, 2, 3, 6, 7, 8, 11, 12, 13\}$ solely received `HELLO`s, `HELLOACK`s, and `ACK`s, thus simulating reactive jamming attacks against other kinds of frames. In the third run, the same set of nodes did, in addition, not receive `HELLO`s from their permanent neighbors. This reactive jamming attack effectively disables AKES' suppression of redundant `HELLO`s. In the forth run, if any of the nodes $\{1, 2, 3, 6, 7, 8, 11, 12, 13\}$ had just reset Trickle, i.e., if $I = I_{\text{min}}$, it did not receive any `HELLO`s. The idea behind this reactive jamming attack is to delay session key establishment in order to cause multiple Trickle resets instead of just one. Initially, all nodes used Parameter Set 4 and then all four runs were repeated with Parameter Set 5 and 6.[1]

Figure 4.12 shows the results. Let us first look at the number of sent `HELLO`s. If no reactive jamming attack is launched, extremely few `HELLO`s are sent, regardless of the employed parameter set. This is because Trickle suspects that the network topology is stable and therefore reduces the rate of `HELLO`s. A lot more `HELLO`s are sent when jamming all frames except `HELLO`s, `HELLOACK`s, and `ACK`s as this causes AKES to delete permanent neighbors, reestablish session keys, and potentially reset Trickle. Expectably, the intrinsic defense against yo-yo attacks greatly mitigates such reactive jamming attacks if $T_{\text{lif}} = 30\text{min}$. This is due to the fact that AKES then deletes inactive permanent neighbors only after $T_{\text{lif}} = 30\text{min}$. Conversely, when configuring AKES more responsively by setting $T_{\text{lif}} = 5\text{min}$, the intrinsic defense against yo-yo attacks protects much worse. For comparison, although the LBC-based defense against yo-yo attacks also uses $T_{\text{lif}} = 5\text{min}$, it limits the rate of sent `HELLO`s to $\rho_{\text{HELLO}} = \frac{1}{300}\text{Hz}$. Another attack strategy is to jam `HELLO`s to permanent neighbors so as to abstain victim nodes from suppressing `HELLO`s. Indeed, the number of sent `HELLO`s increases under such reactive jamming attacks, at least when using the intrinsic defense against yo-yo attacks, as shown in Figure 4.12c. By contrast, the LBC-based defense against yo-yo attacks successfully limits the rate of sent `HELLO`s

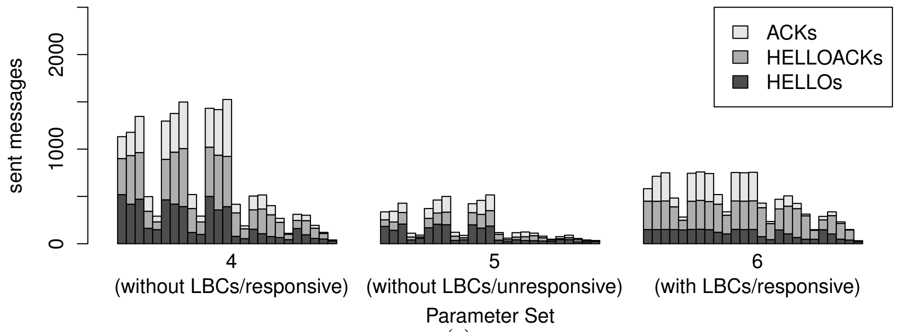
[1]Though it seems difficult for an attacker to prevent some nodes from receiving a `HELLO` while letting it pass to others, this is possible, e.g., if the victim network uses ContikiMAC. This is because ContikiMAC transmits broadcasts as a strobe of frames, selected ones of which may be jammed exactly when a certain node wakes up. Alternatively, an attacker may install multiple jammers and locally jam `HELLO`s. In Section 4.6, we discuss the feasibility of reactive jamming and hidden wormhole attacks in more detail.



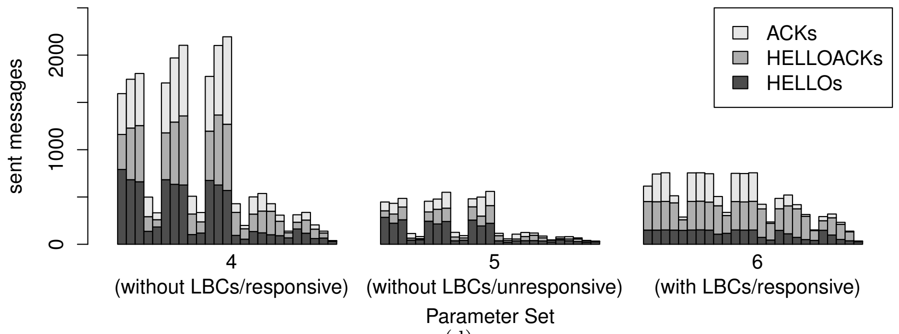
(a)



(b)



(c)



(d)

Figure 4.12: Sent `HELLO`s, `HELLOACK`s, and `ACK`s per node after 12 virtual hours (a) without attacks, (b) when jamming all frames except `HELLO`s, `HELLOACK`s, and `ACK`s, (c) when jamming `HELLO`s to permanent neighbors in addition, and (d) when also jamming `HELLO`s to nodes that had just reset Trickle

to $\frac{1}{300}$Hz. Another level of aggravation is to jam HELLOs to nodes that had just reset Trickle so as to cause multiple Trickle resets instead of just one. Again, the LBC-based defense against yo-yo attacks limits the rate of sent HELLOs under such kind of reactive jamming attacks to $\frac{1}{300}$Hz, as shown in Figure 4.12d. A secondary repercussion of yo-yo attacks is additional HELLOACK and ACK transmissions and receptions. In this regard, the LBC-based restriction on the rate of HELLOACKs also takes effect in this context. The intrinsic HELLO flood defense, by contrast, fails to limit the rate of HELLOACKs to $\frac{1}{150}$Hz. For example, in Figure 4.12b, the rate of answered HELLOs of node 7 between hour 1 and 12 is $\frac{1}{70.09}$Hz when using Parameter Set 4, whereas it is $\frac{1}{150.00}$Hz when using Parameter Set 6. In sum, the intrinsic defense against yo-yo attacks only protects against reactive jamming-based yo-yo attacks when extending T_{lif} to undesirable durations.

Next, to compare the protection against hidden wormhole-based yo-yo attacks of the intrinsic and the LBC-based defense against yo-yo attacks, the above experiment was repeated with a modified topology, which is shown in Figure 4.13. Therein, the nodes $\{3, 4, 5, 9, 10, 15\}$ and $\{11, 16, 17, 21, 22, 23\}$ could communicate with each other. However, the hidden wormhole exclusively tunneled HELLOs, HELLOACKs, and ACKs and not other kinds of frames. In the first run, no attacks other than the hidden wormhole attack was launched. In the second run, this attack was exacerbated by combining it with reactive jamming-based yo-yo attacks against the nodes $\{1, 2, 3, 6, 7, 8, 11, 12, 13\}$. Specifically, like in the previous experiment, each of these nodes only received (i) ACKs, (ii) HELLOACKs, and (iii) HELLOs from non-permanent neighbors if the node had not just reset Trickle.

Figure 4.14 shows the results. Interestingly, the hidden wormhole turned out to be relatively benign. Furthermore, the results suggest that hidden wormhole-based yo-yo attacks only cause the nodes that are connected via the hidden wormhole to send more HELLOs, HELLOACKs, and ACKs, whereas reactive jamming-based yo-yo attacks also cause non-attacked nodes of the victim network to send more HELLOs, HELLOACKs, and ACKs, as shown in Figure 4.12b through 4.12d. That said, Figure 4.14b demonstrates that combining hidden wormhole with reactive jamming attacks exacerbates both of these attacks.

4.5.5.2 Yo-Yo Attacks by Internal Attackers

Lastly, the protection against yo-yo attacks by internal attackers of the intrinsic and the LBC-based defense against yo-yo attacks was assessed as follows. Once more, the topology shown in Figure 4.8 was used, where all 25 nodes booted at a random point in time during the first 30 virtual minutes, and the frame loss was 0%. However, this time, the nodes $\{13, 14, 18, 19\}$ acted maliciously by (i) setting $I_{\text{min}} = I_{\text{max}} = 30s$, (ii) choosing $M_{\text{bac}} = T_{\text{ack}} = 5$s, (iii) not replying to UPDATEs, (iv) deleting any new permanent neighbor right after session key establishment, (v) not rate-limiting their number of outgoing HELLOs, HELLOACKs, and ACKs, and (vi) not communicating among themselves. In effect, these nodes frequently sent valid HELLOs and completed three-way handshakes whenever possible by sending valid HELLOACKs and ACKs. In three successive runs over 12 virtual hours, the legitimate nodes used Parameter Set 4 through 6 and logged their number of sent HELLOs, HELLOACKs, and ACKs.

Figure 4.15 shows the results. Unsurprisingly, the LBC-based defense against yo-yo attacks protects against yo-yo attacks by internal attackers equally well

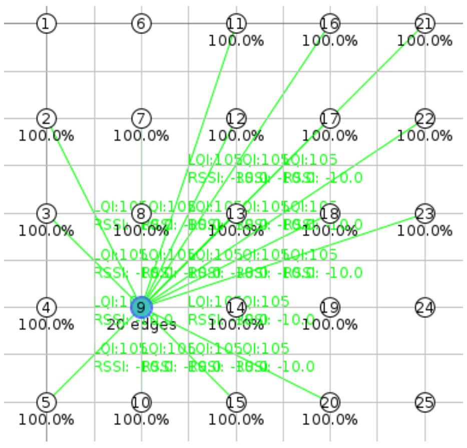
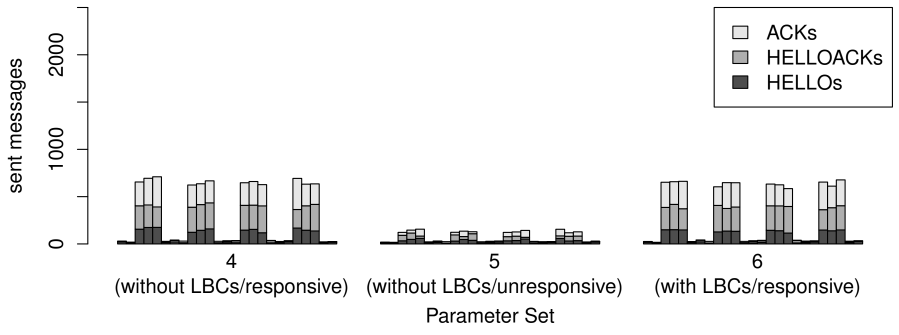
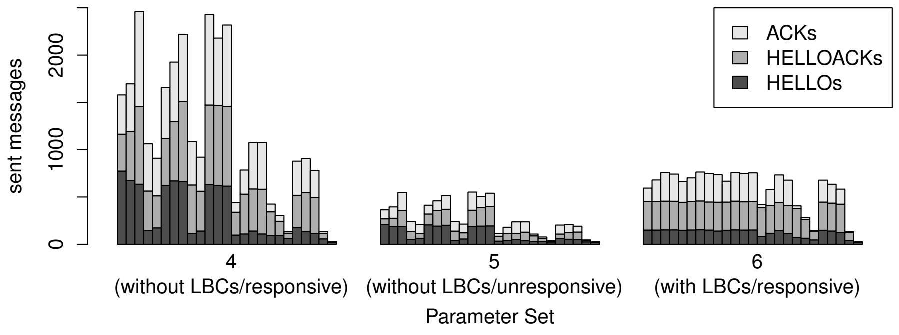


Figure 4.13: Network topology with a hidden wormhole that ralays the traffic between the nodes $\{3, 4, 5, 9, 10, 15\}$ and the nodes $\{11, 16, 17, 21, 22, 23\}$



(a)



(b)

Figure 4.14: Sent `HELLO`s, `HELLOACK`s, and `ACK`s per node after 12 virtual hours (a) in the presence of a hidden wormhole and (b) if launching reactive jamming attacks in parallel

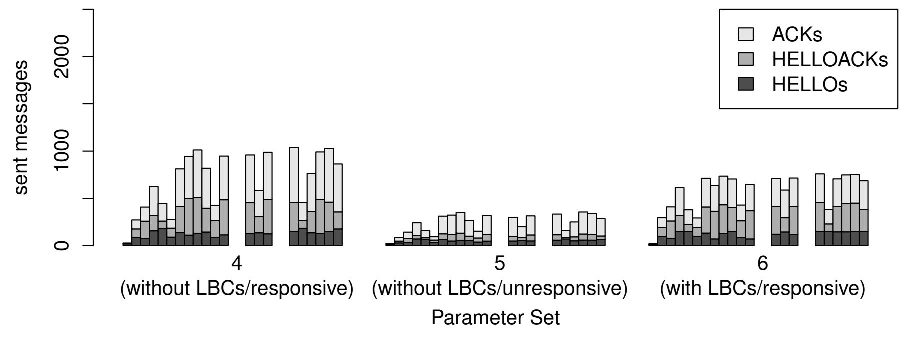


Figure 4.15: Sent HELLOs, HELLOACKs, and ACKs per node after 12 virtual hours in the face of four attacker-controlled nodes

like against yo-yo attacks by external attackers. Yet, surprisingly, when using the intrinsic defense against yo-yo attacks, the results indicate that yo-yo attacks by internal attackers are less severe than yo-yo attacks by external attackers. This is in contrast to HELLO flood attacks by internal attackers, which are more severe than HELLO flood attacks by external attackers, at least when using the intrinsic HELLO flood defense.

4.5.5.3 Discussion

The intrinsic defense against yo-yo attacks comes at the cost of extending T_{lif}. As discussed earlier, extending T_{lif} may deprive AKES from establishing session keys with actual neighbors in situations where a lot of RAM is allocated for storing inactive permanent neighbors. Moreover, an open question is how to choose T_{lif} so as to achieve a certain level of protection. Conversely, the LBC-based defense against yo-yo attacks enables freeing allocated RAM quickly and can directly be configured to meet any required level of protection.

4.6 Related Work

Below, we sum up related work on alternatives to session keys, session key establishment, HELLO flood attacks, reactive jamming, and hidden wormholes.

4.6.1 Alternatives to Session Keys

Establishing session keys solves both conflicts between the frame counter-based replay protection of IEEE 802.15.4 security and key predistribution. A different approach to avoid the swapping of frame counters, in particular, was proposed by Luk et al. [147]. They suggested storing anti-replay data in Bloom filters, which consume a fixed amount of RAM and hence obviate swapping frame counters [147]. Unfortunately, Bloom filters sometimes falsely report that a frame was replayed and, disapprovingly, cause a high energy consumption [160]. After all, Bloom filters do not help survive reboots, whereas session keys help both to avoid swapping and to survive reboots.

An entirely different approach to solve both conflicts between the frame counter-based replay protection of IEEE 802.15.4 security and key predistribution appeared as part of the TSCH MAC protocol [29]. There, the approach is to use timeslot indices in lieu of frame counters. However, TSCH's mechanisms for time synchronization are vulnerable to both internal and external attackers. In fact, Yang et al. pointed out that attacker-controlled nodes can launch various attacks against TSCH's mechanisms for time synchronization [161]. Moreover, TSCH's mechanisms for time synchronization do not even withstand external attackers. For example, jamming beacon frames constitutes a devastating denial-of-sleep attack against unsynchronized TSCH nodes since this holds unsynchronized TSCH nodes in the energy-consuming receive mode.

4.6.2 Session Key Establishment

Current protocols for establishing session keys among neighboring IEEE 802.15.4 nodes can be clustered into PKC-based [20, 30, 31, 32, 33], KDC-based [16], and key predistribution-based protocols [10, 34, 35]. Unfortunately, all current PKC- and KDC-based protocols are susceptible to denial-of-sleep attacks since, by injecting requests for session key establishment, attackers can either trigger energy-consuming processing or communication [17]. Moreover, another limitation of most of the current protocols for establishing session keys among neighboring IEEE 802.15.4 nodes is that they do not discover new neighbors at runtime. The only exception to this seems to be Ilia et al.'s protocol, where session key establishment is triggered if the upper layer sends a unicast frame to a neighbor with whom no session keys were established, yet [33]. However, they do not specify what happens if the upper layer sends a broadcast frame, leaving their solution incomplete. A possible solution we considered is to use upper-layer broadcast frames as `HELLO`s. That is, if the upper layer sends a broadcast frame, the MAC layer could piggyback a cryptographic random number on it. Receivers that have not yet established session keys with the sender of a broadcast frame could extract the contained cryptographic random number and reply with a `HELLOACK` as per AKES' three way handshake. This solution would obviate Trickling `HELLO`s and may even result in a faster adaption to topology changes. On the other hand, this solution would break compatibility with the `unicast_strategy`, where broadcast frames are sent as unicast frames to each permanent neighbor one after another. Furthermore, since the `unicast_strategy` comes in very handy when it comes to implementing channel hopping in Chapter 7, we did not follow up on that solution.

4.6.3 HELLO Flood Attacks

Karlof et al. identified `HELLO` flood attacks as a general attack against wireless mesh networks [36]. Actually, many protocols for wireless mesh networks use some kind of `HELLO` messages to discover neighboring nodes. Thus, a general attack against such protocols is to inject or replay their `HELLO` messages with high transmission powers. Such `HELLO` flood attacks usually mislead receivers into thinking that they are in direct communication range of the node that is pretended to be the sender of the injected or replayed `HELLO` message [36]. In the context of session key establishment, a basic countermeasure is to perform a three-way handshake so as to raise the confidence of being in direct communication range of each other [34]. But, this countermeasure entails the denial-of-sleep vulnerability that injected `HELLO`s trigger replies. To avoid replying to injected `HELLO`s, Lim suggested authenticating `HELLO`s using hash chains [35]. However, when using his approach, attackers can force victim nodes to perform many hash computations by injecting `HELLO`s with late deployment intervals, which constitutes a denial-of-sleep vulnerability, too. Alternatively, APKES sheds `HELLO`s if they arrive in bursts. Our LBC-based defense against `HELLO` flood attacks advances the shedding of `HELLO`s by reducing its incurred delays to session key establishment. In Chapter 6, we further expand on the shedding `HELLO`s by performing the shedding of `HELLO`s already during reception.

4.6.4 Reactive Jamming

The feasibility of reactively jamming IEEE 802.15.4 frames was shown by different groups [72, 162, 163]. Wood et al., for instance, configured an off-the-shelf IEEE 802.15.4 transceiver to issue interrupts upon detecting the SHR of a frame [72]. Then, if an SHR interrupt is issued, the IEEE 802.15.4 transceiver was instructed to jam for a short period of time. Alternatively, Wilhelm et al. implemented reactive jamming based on software-defined radio, which minimizes the time between the analysis of incoming radio traffic and the decision to jam [163]. Either method is applicable to carry out reactive jamming against AKES.

4.6.5 Hidden Wormholes

The feasibility of setting up a hidden wormhole was, e.g., demonstrated by Francillon et al. [164]. They achieved delays of the order of nanoseconds by using cut-through forwarding, i.e., by tunnelling incoming frames already during reception. Such hidden wormholes are hard to detect by measuring delays. Hence, more sophisticated methods for detecting hidden wormholes were devised. In [9], we, e.g., propose a channel reciprocity-based scheme for detecting and subsequently avoiding hidden wormholes, called Secure Channel REciprocity-based WormholE Detection (SCREWED). Unfortunately, SCREWED comes at a high communication overhead and can be bypassed. In this chapter, we hence resorted to an alternative way of dealing with hidden wormholes, namely by mitigating their repercussions.

4.7 Summary

In this section, we have tackled two widely open problems regarding the establishment of session keys among neighboring IEEE 802.15.4 nodes. The first of them is the adaptation to topology changes. To this end, we have proposed a mechanism that deletes inactive permanent neighbors, as well as a Trickle-based mechanism for initiating the discovery of new neighbors. Though Trickle actually is an algorithm for disseminating information in wireless sensor networks, we have shown that Trickle is general enough to be applied to the problem of scheduling the broadcasting of `HELLO`s, too. The advantages of doing so are twofold. On the one hand, Trickle is well-known, well-studied, and even standardized [149]. On the other hand, Trickle helps trade off energy efficiency against the speed of adapting to topology changes. The second open problem is protecting protocols for establishing session keys among neighboring IEEE 802.15.4 nodes against denial-of-sleep attacks. We have identified two kinds of denial-of-sleep attacks against such protocols. In `HELLO` flood attacks, on the one hand, an attacker injects requests for session key establishment so as to trigger energy-consuming processing or communication. In yo-yo attacks, on the other hand, an attacker makes links temporarily available or temporarily unavailable so as to provoke more attempts to establish session keys, reestablishments of session keys, or both. To mitigate `HELLO` flood attacks, we have proposed to abstain from using KDCs and PKC. Besides, to defend against both `HELLO` flood and yo-yo attacks, we have considered adapting intrinsic parameters of AKES, as well as adding LBC-based defenses that rate-limit outgoing session key establishment-related messages. Both sets of defenses have turned out to protect against `HELLO` flood and yo-yo attacks. However, in contrast to the intrinsic denial-of-sleep defenses, the LBC-based ones incur no delays to session key establishment at times when no denial-of-sleep attacks are launched and can be configured more easily.

Chapter 5

Denial-of-Sleep-Resilient Medium Access Control: Extended Problem Statement

This chapter elaborates on the problem of denial-of-sleep-resilient medium access control. We begin with analyzing the susceptibility of ContikiMAC, CSL, TSCH, and LWB to denial-of-sleep attacks. Then, we sum up prior work on denial-of-sleep-resilient medium access control and highlight persisting research gaps. Eventually, we conclude that ContikiMAC and CSL are most promising to be developed into a denial-of-sleep-resilient MAC protocol, yet that several research gaps have to be filled in this endeavor.

5.1 Review of Prevalent MAC Protocols

ContikiMAC, CSL, TSCH, and LWB come without integrated denial-of-sleep defenses. This raises the question whether it is possible to retrofit denial-of-sleep resilience into them at all. To approach this question, this section identifies potential denial-of-sleep attacks against ContikiMAC, CSL, TSCH, and LWB.

5.1.1 Terminology

Let us first clarify some of our terminology. We subdivide denial-of-sleep attacks against MAC protocols into *reception-* and *transmission-oriented* ones. While reception-oriented denial-of-sleep attacks mainly cause victim nodes to stay longer in the energy-consuming receive mode, transmission-oriented denial-of-sleep attacks mainly cause victim nodes to stay longer in the energy-consuming transmit mode. An orthogonal distinction we sometimes use is between *external* and *internal* denial-of-sleep attacks. External denial-of-sleep attacks, on the one hand, can be carried out without access to any cryptographic key. Internal denial-of-sleep attacks, on the other hand, can only be launched by an attacker with access to one or more cryptographic keys. Hence, internal denial-of-sleep

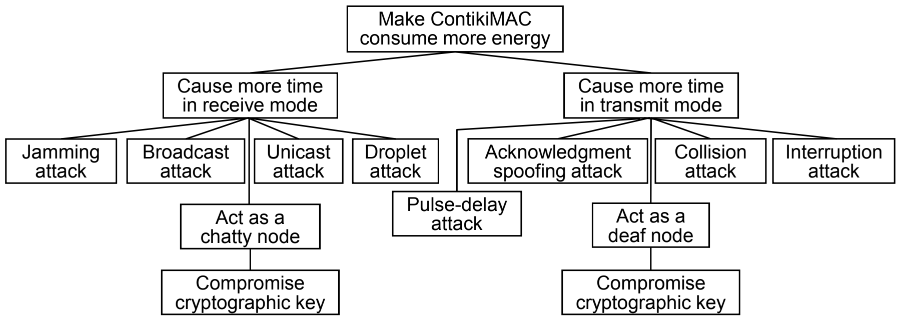

Make ContikiMAC consume more energy
Cause more time in receive mode
Cause more time in transmit mode
Jamming attack
Broadcast attack
Unicast attack
Droplet attack
Acknowledgment spoofing attack
Collision attack
Interruption attack
Act as a chatty node
Pulse-delay attack
Act as a deaf node
Compromise cryptographic key
Compromise cryptographic key

Figure 5.1: Attack tree of denial-of-sleep attacks against ContikiMAC

attacks involve an additional attack step wherein the attacker obtains certain cryptographic keys, such as through side-channel attacks or physical tampering [165, 166]. Correspondingly, we refer to an attacker without access to any cryptographic key as an *external attacker*, and to an attacker with access to one or more cryptographic keys as an *internal attacker* [156].

5.1.2 ContikiMAC

Figure 5.1 shows an attack tree of our identified denial-of-sleep attacks against ContikiMAC. In the following, we will first go through reception-oriented denial-of-sleep attacks against ContikiMAC and then through transmission-oriented denial-of-sleep attacks against ContikiMAC.

5.1.2.1 Reception-Oriented Denial-of-Sleep Attacks

Recall that ContikiMAC regularly performs two CCAs for detecting incoming transmissions. This approach for detecting incoming transmissions results in a low base energy consumption, but incurs two issues. First, it is crucial to set the CCA threshold appropriately [81, 167, 168]. If the CCA threshold is too low, receivers will often wake up unnecessarily, thereby wasting much energy. Conversely, if the CCA threshold is too high, transmissions may go undetected. Second, if an attacker constantly or intermittently jams the channel, ContikiMAC will stay in receive mode until ContikiMAC's fast-sleep optimization eventually disables the received mode. However, despite ContikiMAC's fast-sleep optimization eventually intervenes, ContikiMAC's energy consumption increases a lot under jamming attacks, as we demonstrate in Section 6.2.4. Sha et al. addressed the first issue [81]. They showed that setting a static CCA threshold is insufficient to cope with false wake ups and undetected transmissions. Rather, the CCA threshold should be adapted dynamically at runtime. Accordingly, they proposed the Adaptive Energy Detection Protocol (AEDP), which adjusts the CCA threshold to trade off false wake ups against undetected transmissions.

At first glance, AEDP might also reduce the incurred energy consumption of jamming attacks since AEDP increases the CCA threshold as false wake ups occur. However, AEDP caps CCA thresholds to fit between the minimum RSSI among all incoming links and the floor noise, as shown in Figure 5.2. Thus, as long as attackers jam with a strong enough transmission power, receivers still

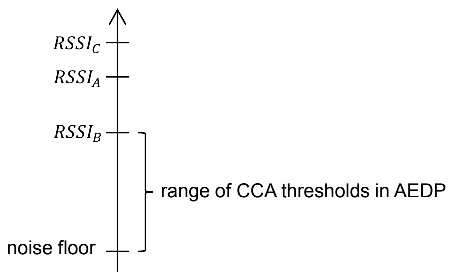


Figure 5.2: Range of CCA thresholds in AEDP

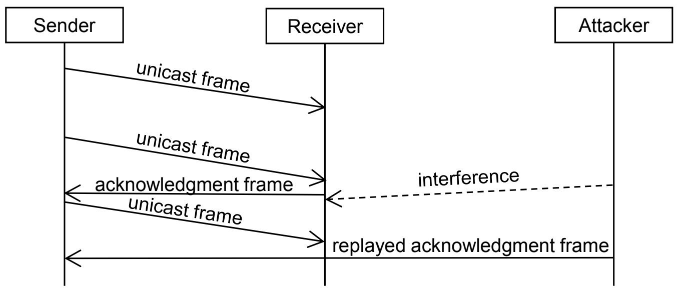
wake up. On the other hand, if a jamming attack causes the floor noise to increase beyond the RSSI of an incoming link, AEDP's behavior is undefined. In such cases, it is reasonable to set the CCA threshold slightly above the level of the floor noise. This is because (i) communication on links with lower RSSIs than the floor noise is error-prone anyway and (ii) it requires attackers to launch jamming attacks with higher and higher transmission powers as otherwise victim nodes will not wake up, thus complicating jamming attacks at least.

After all, to cause ContikiMAC to spend more time in receive mode, attackers can launch *broadcast* [22], *unicast*, or *droplet attacks* [169], as well. In a broadcast attack, an attacker injects or replays a broadcast frame in the same way as a legitimate sender. As a result, every receiver in range will expend energy for receiving and processing the frame. Of course, IEEE 802.15.4 security and AKES eventually reject injected and replayed frames, but only after having consumed energy for their reception and processing already. Likewise, in what we call a unicast attack, an attacker injects or replays a unicast frame like a legitimate sender. In effect, a receiver may not only receive and process the frame, but also send an acknowledgment frame since IEEE 802.15.4 security permits receivers to acknowledge prior to checking the authenticity and freshness of the unicast frame that is being acknowledged. Finally, in order to save his own energy during unicast and broadcast attacks, an attacker can send droplets instead of whole frames. A droplet is just the beginning of a frame without the rest of it being transmitted. Nevertheless, receivers will detect a droplet and demodulate the noise that follows, just to conclude that the checksum is erroneous.

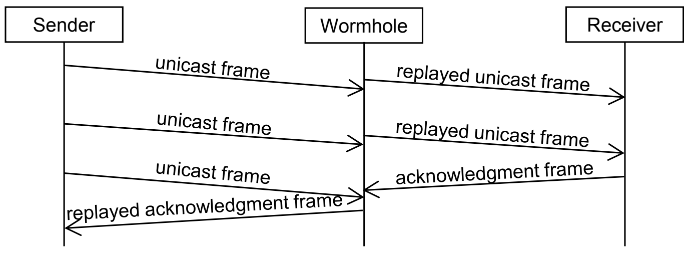
Moreover, a reception-oriented internal denial-of-sleep attack against ContikiMAC is to inject what we call a *chatty node* into a victim network. A chatty node sends fresh authentic frames to its neighbors, thus causing its neighbors to spend more time in receive mode and to further process these frames.

5.1.2.2 Transmission-Oriented Denial-of-Sleep Attacks

The denial-of-sleep attacks discussed so far primarily aim to mislead victim nodes into staying longer in receive mode. Sometimes these reception-oriented denial-of-sleep attacks also cause victim nodes to stay longer in transmit mode. This, e.g., happens if a victim node sends an acknowledgment frame in response to a unicast attack or if retransmissions happen due to the attacker using the channel. Next, we look at denial-of-sleep attacks that primarily aim to mislead victim nodes into staying longer in transmit mode, namely *pulse-delay* [156],



(a)



(b)

Figure 5.3: Pulse-delay attack via (a) interference and (b) a hidden wormhole

acknowledgment spoofing [36], *collision* [38], and *interruption attacks.*

Pulse-delay attacks against ContikiMAC come in two flavors. First, an attacker can jam during the transmission of an acknowledgment frame and replay the jammed acknowledgment frame later on, as shown in Figure 5.3a. Second, an attacker can set up a hidden wormhole and deliberately delay acknowledgment frames, as shown in Figure 5.3b. In either case, senders strobe more often. Moreover, since ContikiMAC's phase-lock optimization learns wake-up times based on when acknowledgment frames are received, ContikiMAC's phase-lock optimization learns wrong wake-up times as a result. Thus, subsequent transmissions to affected neighbors are likely to fail and hence to ensue retransmissions. In the worst case, ContikiMAC's phase-lock optimizations resorts to relearn the wake-up time of an affected neighbor. If so, the next unicast frame to the affected neighbor is sent "unsynchronized" for an entire wake-up interval plus once, which consumes much energy.

An acknowledgment spoofing attack against ContikiMAC is shown in Figure 5.4. The attack comprises two phases. In the first phase, an attacker captures a fresh authentic acknowledgment frame sent from a node B to a node A. It is crucial that the captured acknowledgment frame was not received by A, e.g, due to having been jammed. Jamming is, however, not the only possible method for capturing such an acknowledgment frame. Alternatively, an attacker may inject or replay a unicast frame to B under the name of A. Since IEEE 802.15.4 security permits receivers to send acknowledgment frames prior to checking the authenticity and freshness of the unicast frame that is being acknowledged, B may reply with a fresh authentic acknowledgment frame destined to A like in Figure 5.4. In the second phase, when A sends a unicast frame to B, the attacker replays the captured acknowledgment frame so as to trick A into believing that its unicast frame was successfully received, though it might have

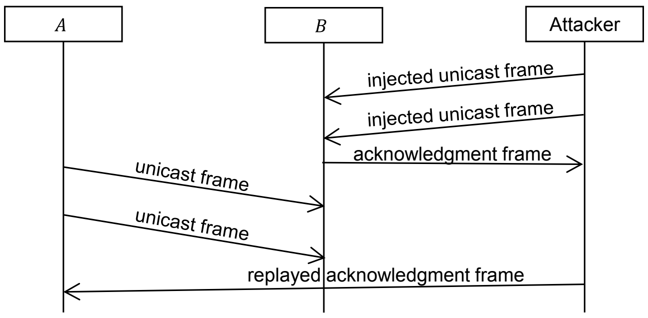


Figure 5.4: Acknowledgment spoofing attack against ContikiMAC

been jammed for example. This attack works out if A has not seen a greater frame counter than that of the captured acknowledgment frame from B in the meantime. Echoing sequence numbers (introduced in Section 2.1.1.3) only complicates acknowledgment spoofing attacks since sequence numbers reoccur every $2^8 = 256$ increments, thus rendering coincidences likely. After all, attackers can easily predict a node's next sequence number via eavesdropping and capture a fitting acknowledgment frame in the first phase. In the context of ContikiMAC, acknowledgment spoofing attacks entail three repercussions. First, a sender is misled into believing that its unicast frame was successfully received. This can result in the loss of important messages or the continuation to use bad links [36]. Second, ContikiMAC's phase-lock optimization learns a wrong wake-up time, likely causing subsequent transmissions to the affected neighbor to fail. Third, ContikiMAC's phase-lock optimization may ultimately decide to relearn the wake-up time of the affected neighbor, which then consumes much energy.

A collision attack is a special reactive jamming attack, where the attacker jams unicast or acknowledgment frames with the goal of provoking retransmissions. In the context of ContikiMAC, a collision attack causes a victim node to strobe for long since no acknowledgment frame comes back. The incurred energy consumption increases further if the attacker interferes with subsequent retransmissions, too. To some extent, ContikiMAC's phase-lock optimization mitigates collision attacks by shortening the maximum duration of a strobe of unicast frames once a receiver's wake-up time is known. However, recall that for handling clock drift, ContikiMAC's phase-lock optimization relearns the wake-up time of a receiver if unicast transmissions to the receiver tend to fail. Attackers can hence aggravate collision attacks by repeating them.

Similarly, we identified a variation of collision attacks that we call interruption attacks. Interruption attacks exploit that ContikiMAC checks whether the channel is still free between each consecutively strobed frame. If the channel is no longer free, ContikiMAC starts over with strobing the frame after a random back off period, unless a maximum number of retransmissions is reached already. Accordingly, in an interruption attack, an attacker jams the channel between consecutively strobed frames so as to provoke retransmissions.

Moreover, a transmission-oriented internal denial-of-sleep attack against ContikiMAC is to inject what we call a *deaf node* into a victim network. A deaf node intentionally acknowledges transmissions only after a couple of retransmissions or not at all.

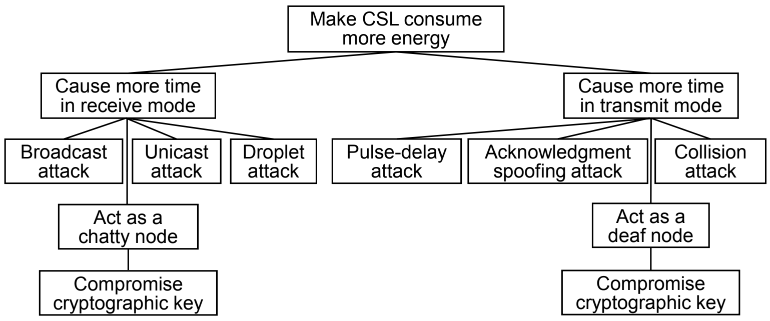

Make CSL consume more energy
Cause more time in receive mode
Cause more time in transmit mode
Broadcast attack
Unicast attack
Droplet attack
Pulse-delay attack
Acknowledgment spoofing attack
Collision attack
Act as a chatty node
Act as a deaf node
Compromise cryptographic key
Compromise cryptographic key

Figure 5.5: Attack tree of denial-of-sleep attacks against CSL

5.1.3 CSL

Figure 5.5 shows an attack tree of our identified denial-of-sleep attacks against CSL.

5.1.3.1 Reception-Oriented Denial-of-Sleep Attacks

Recall that CSL detects incoming transmissions by shortly listening for a wake-up frame. More specifically, when waking up, CSL shortly enables the receive mode to listen for an SHR. If an SHR is detected and the subsequent frame turns out to be a valid wake-up frame, CSL sleeps until shortly before the announced rendezvous time. Then, CSL re-enables the receive mode to listen for another SHR. If an SHR is detected, CSL proceeds to receive the subsequent payload frame. Accordingly, pure jamming attacks do not cause CSL to stay longer in receive mode. However, an attacker can mislead CSL into staying longer in receive mode by launching broadcast, unicast, or droplet attacks. To do so, an attacker injects or replays a sequence of wake-up frames followed by a payload frame that is either a broadcast frame, unicast frame, or droplet. In effect, CSL will receive, process, and potentially acknowledge the injected or replayed payload frame. Moreover, a method for aggravating broadcast, unicast, and droplet attacks against CSL is to set the Frame Pending bit of the injected or replayed payload frame. If a receiver does not check the authenticity and freshness of a payload frame right after its reception, the receiver will stay in receive mode since he expects another payload frame. That payload frame may, again, have the Frame Pending bit set and so on.

Besides, attackers may use captured cryptographic keys to inject chatty nodes into a victim network, which is not different from ContikiMAC.

5.1.3.2 Transmission-Oriented Denial-of-Sleep Attacks

CSL is also vulnerable to collision, pulse-delay, and acknowledgment spoofing attacks. A collision attack against CSL is depicted in Figure 5.6. To some extent, CSL's optimization of shortening wake-up sequences mitigates collision attacks. But, collision attacks can also deprive CSL of updating stored wake-up times for long, thus causing the uncertainty about the affected neighbor's wake-up time to increase and wake-up sequences to him to stretch. Pulse-delay attacks may be launched against any frame that piggybacks a CSL phase,

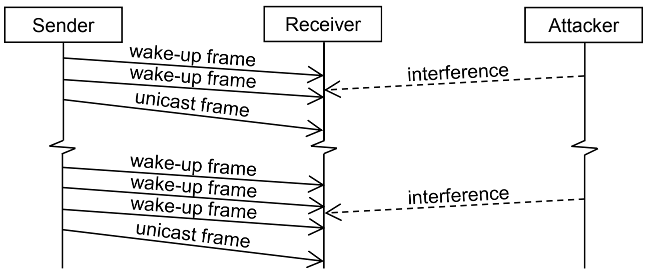


Figure 5.6: Collision attack against CSL. Interfering with unicast transmissions ensues energy-consuming retransmissions, as well as the uncertainty about the affected neighbor's wake-up time to extend.

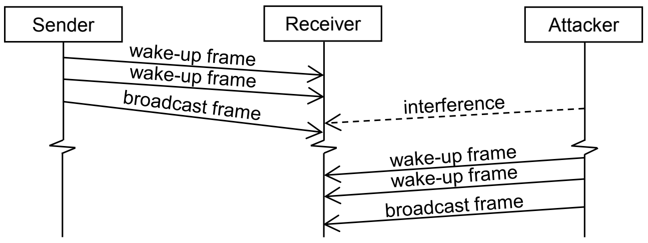


Figure 5.7: Pulse-delay attack against CSL. Jamming frames that piggyback a CSL phase and replaying these jammed frames delayed misleads victim nodes into learning wrong wake-up times, likely causing subsequent transmissions to the affected neighbor to fail.

as exemplified in Figure 5.7. A delayed frame passes IEEE 802.15.4 security since IEEE 802.15.4 security considers frames fresh as long as they contain a greater frame counter than the last authentic frame with that frame counter. Further, since the piggybacked CSL phase of a delayed frame is relative to its time of transmission, CSL learns wrong wake-up times as a result. Ultimately, subsequent unicast transmissions to the affected neighbor are likely to fail, thus ensuing retransmissions, too. Again, these repercussions exacerbate over time as the uncertainty about the affected neighbor's wake-up time extends. Finally, acknowledgment spoofing attacks can be launched against CSL in the same manner as against ContikiMAC.

Besides, internal attackers may inject deaf nodes into a victim network, too.

5.1.4 TSCH

Figure 5.8 shows an attack tree of our identified denial-of-sleep attacks against TSCH.

5.1.4.1 Reception-Oriented Denial-of-Sleep Attacks

TSCH is also vulnerable to broadcast, unicast, and droplet attacks. Moreover, an attacker can schedule the injection or replay of a frame or droplet so that its SHR arrives at the end of the reception period of duration `RxWait`. In effect,

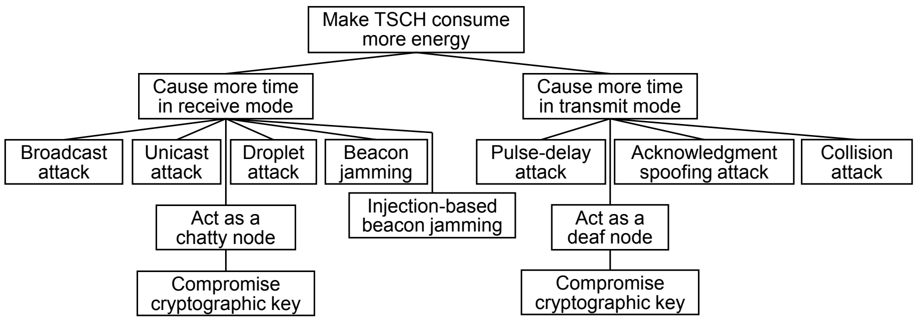

Make TSCH consume more energy
Cause more time in receive mode
Cause more time in transmit mode
Broadcast attack
Unicast attack
Droplet attack
Beacon jamming
Injection-based beacon jamming
Act as a chatty node
Compromise cryptographic key
Pulse-delay attack
Acknowledgment spoofing attack
Collision attack
Act as a deaf node
Compromise cryptographic key

Figure 5.8: Attack tree of denial-of-sleep attacks against TSCH

TSCH not only stays in receive mode for `RxWait`, but also for the time it takes to receive the bulk of the injected or replayed frame or droplet.

Apart from broadcast, unicast, and droplet attacks, we also identified one other external denial-of-sleep attack that causes TSCH to stay longer in receive mode. We call this attack *beacon jamming*. Recall that unsynchronized TSCH nodes repeatedly enter the receive mode until they overhear an appropriate beacon frame. Thus, jamming beacon frames causes an unsynchronized TSCH node to stay in receive mode almost continuously, which is extremely energy consuming. Similarly, a variation of this attack is to continuously inject frames so as to keep unsynchronized TSCH nodes busy with demodulating unwanted frames instead of beacon frames. Beacon jamming can not only be performed at deployment time when TSCH nodes are unsynchronized anyway, but potentially at any time by desynchronizing victim nodes beforehand. For this, an attacker can, e.g., reactively jam resynchronization-related frames or inject unwanted frames, too. Additional methods for desynchronizing TSCH nodes are pulse-delay and acknowledgment spoofing attacks, both of which we discuss shortly.

Besides, internal attackers can not only desynchronize TSCH nodes in various ways [161], but also inject chatty nodes into a victim network.

5.1.4.2 Transmission-Oriented Denial-of-Sleep Attacks

Figure 5.9 depicts a pulse-delay attack against TSCH's frame-based resynchronization [161]. The general idea is to jam a frame that is used for frame- or acknowledgment-based resynchronization and to replay the jammed frame later on. As a result, victim nodes are tricked into "moving to the future". However, delaying frames during a frame-based resynchronization can not cause the time offset `Offset` to grow larger than `RxWait` as delayed frames will either be missed or rejected otherwise. This is because receivers only stay in receive mode for `RxWait` and replaying a frame within a later timeslot causes receivers to reject the frame since they will use a different CCM nonce to check its MIC. Similarly, senders of unicast frames only accept acknowledgment frames `AckDelay` after transmitting for `AckWait`, as shown in Figure 2.7. Thus, `Offset` remains smaller or equal than the minimum over $\{\texttt{RxWait}, \texttt{AckWait}\}$ when an attacker delays unicast frames during an acknowledgment-based resynchronization.

As for acknowledgment spoofing attacks, these are more difficult against TSCH, but seem feasible if receivers send acknowledgment frames prior to check-

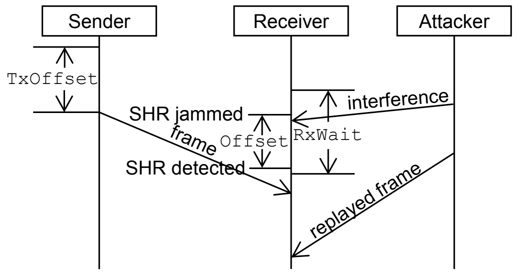


Figure 5.9: Pulse-delay attack against TSCH's frame-based resynchronization

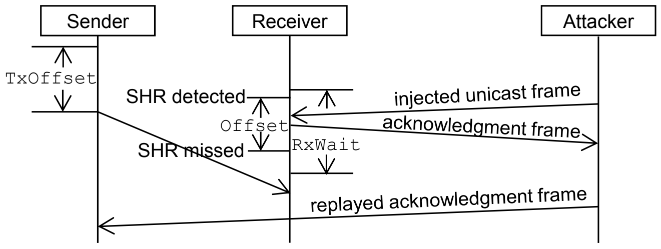


Figure 5.10: Acknowledgment spoofing attack against TSCH's acknowledgment-based resynchronization

ing the authenticity and freshness of the unicast frame that is being acknowledged, as shown in Figure 5.10. In the first phase of the attack, an attacker captures a fresh authentic acknowledgment frame by injecting or replaying a unicast frame to a node B under the name of another node A. In the second phase of the attack, within the same timeslot, when A sends a frame to B, the attacker replays the captured acknowledgment frame so as to trick A into believing that its frame was successfully received. As both phases have to happen within one timeslot, the attacker must be quite fast. This may still be feasible when injecting or replaying a very short frame right when B wakes up, as shown in Figure 5.10. The short-term repercussions of an acknowledgment spoofing attack against TSCH's acknowledgment-based resynchronization are (i) that the victim node "moves to the past" by `RxWait` at most and (ii) that the victim node is tricked into believing that its unicast frame was successfully received.

Although `RxWait` and `AckWait` impose a limit on the time offset due to pulse-delay and acknowledgment spoofing attacks, attackers can cause a larger time offset in two ways. First, as for multi-hop TSCH networks, attackers can launch such attacks at multiple hops so that time offsets propagate and hence accumulate. Second, as for TSCH networks that use Chang et al.'s adaptive resynchronization [91], false time offsets lead to wrong estimates of clock drifts, which also aggravates pulse-delay and acknowledgment spoofing attacks.

The long-term repercussions of pulse-delay and acknowledgment spoofing attacks are fourfold. First, victim nodes and their timing children desynchronize and may become unable to communicate with properly synchronized nodes. Thus, a whole subtree of nodes may waste energy for retransmissions of unicast frames. Second, in order to synchronize again, unsynchronized TSCH nodes have to listen for a beacon frame and hence become vulnerable to beacon jam-

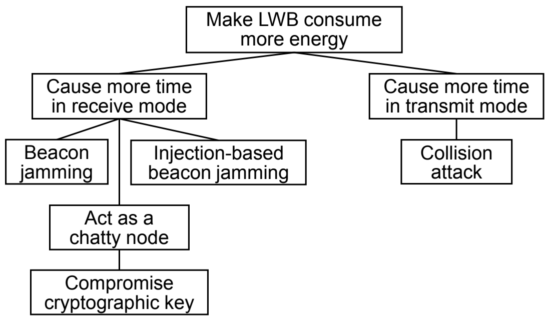

Make LWB consume more energy
Cause more time in receive mode
Cause more time in transmit mode
Beacon jamming
Injection-based beacon jamming
Collision attack
Act as a chatty node
Compromise cryptographic key

Figure 5.11: Attack tree of denial-of-sleep attacks against LWB

ming. Third, if TSCH's time synchronization is also used for application-specific tasks, victim nodes may, e.g., report sensor readings with wrong timestamps or actuate at wrong times. Forth, once victim nodes are properly synchronized again, they may experience a timing loop. If so, CCM nonces may reoccur and thus confidential data may leak [170]. Also, replay attacks may become possible.

Another method for misleading TSCH into staying longer in transmit mode is a collision attack. Collision attacks against TSCH work in the same manner as discussed in the context of ContikiMAC and CSL. In the context of TSCH, collision attacks are, however, much less severe since TSCH does not need to wake up the receiver side via strobing or wake-up sequences.

Additionally, internal attackers can inject deaf nodes into a victim network.

5.1.5 LWB

Figure 5.11 shows an attack tree of our identified denial-of-sleep attacks against LWB.

5.1.5.1 Reception-Oriented Denial-of-Sleep Attacks

In LWB, the equivalent of TSCH's beacon frames are schedule floods. Schedule floods occur at the beginning and end of each round and serve two purposes. First, they assign communication slots to LWB nodes. Second, schedule floods allow unsynchronized LWB nodes to synchronize with the time of the host, as well as synchronized LWB nodes to resynchronize with the time of the host. Yet, when an attacker repeatedly prevents the reception of schedule floods, e.g., by injecting unwanted frames or jamming reactively, victim nodes desynchronize and eventually fall back on listening for schedule floods. Then, as long as an attacker continues to prevent the reception of schedule floods, victim nodes stay in receive mode 100% of their time, which is highly energy consuming.

As for chatty nodes, they may cause LWB to schedule extra communication slots and then incur an increased energy consumption on every node.

5.1.5.2 Transmission-Oriented Denial-of-Sleep Attacks

While LWB does not define retransmission mechanisms, upper layers may notice packet loss and hence initiate end-to-end retransmissions. Thus, collision attacks may still be launched against LWB in order to cause victim nodes to stay longer in transmit mode. Moreover, if end-to-end retransmissions are provoked, LWB potentially schedules additional communication slots, thereby causing all nodes in the network to stay longer in receive mode, too.

Similarly, malicious nodes may decide to omit acknowledgments at upper layers, but we do not count such attacks as attacks against LWB.

5.1.6 Discussion

ContikiMAC, CSL, TSCH, as well as LWB are vulnerable to various denial-of-sleep attacks. The most severe denial-of-sleep attack we identified is beacon jamming. Beacon jamming is particularly severe because it can be launched by external attackers and because it holds victim nodes in receive mode indefinitely. To counter beacon jamming, we considered three approaches. First, a straightforward approach is to send unsynchronized nodes back to sleep for a while when they do not detect an appropriate beacon frame (resp. schedule flood) for some time span. This way, one could limit the severity of beacon jamming, but one would decelerate the recovery from desynchronizations at the same time, resulting in a worse user experience. Second, an approach that would overcome this drawback is a hybrid MAC protocol. More specifically, unsynchronized TSCH nodes could sleep most of their time if they used an asynchronous MAC protocol for receiving beacon frames (resp. schedule floods) and if synchronized nodes used the asynchronous MAC protocol for sending beacon frames (resp. schedule floods). However, the implementation complexity and runtime overhead of such a hybrid MAC protocol would be immense. Third, unsynchronized TSCH nodes could detect beacon jamming and react to it. Yet, while detecting jamming attacks is feasible to some extent [162, 171], it is virtually impossible to discriminate between frames from co-located IEEE 802.15.4 networks and frames injected by an attacker so as to keep unsynchronized TSCH nodes busy with demodulating unwanted frames instead of beacon frames (resp. schedule floods). Altogether, none of our three considered approaches to countering beacon jamming is promising.

Another hard-to-solve issue with both TSCH and LWB is their susceptibility to internal attackers. As for TSCH, Yang et al. showed that internal attackers can launch various attacks against TSCH's mechanisms for time synchronization [161]. Fixing these vulnerabilities, appears to be a major undertaking in its own right. As for LWB, it seems even more difficult to defend against internal attackers because leveraging constructive interference necessitates either the use of network-wide keys, which do not provide any protection against internal attackers, or the use of digital signatures, which are energy consuming [21].

Both CSL and ContikiMAC are not prone to these issues with TSCH and LWB. In fact, both CSL and ContikiMAC are (i) immune to beacon jamming, (ii) immune to time synchronization attacks except for the more tractable pulse-delay and acknowledgment spoofing attacks, and (iii) compatible with pairwise session keys. Thus, our review of prevalent MAC protocols retains ContikiMAC and CSL as promising candidates for further development.

5.2 Related Work

Below, we sum up prior work on protecting medium access control against denial-of-sleep attacks. We will first go into existing defenses against reception-oriented denial-of-sleep attacks and then move on to existing defenses against transmission-oriented denial-of-sleep attacks. Lastly, we highlight research gaps.

5.2.1 Reception-Oriented Denial-of-Sleep Attacks

Hsueh et al. proposed a defense against unicast attacks for two asynchronous MAC protocols for IEEE 802.15.4 networks [40], namely X-MAC and A-MAC [172, 173]. In X-MAC, receivers wake up periodically and scan the channel for a wake-up frame like in CSL. However, in X-MAC, wake-up frames do not contain a rendezvous time and are spaced by short silence periods. Upon reception of a wake-up frame, the receiver replies with an acknowledgment frame, whereupon the sender transmits the payload frame. In A-MAC, each node regularly transmits probe frames. Senders wait until they receive a probe frame from the intended receiver and thereupon transmit an acknowledgment frame followed by the payload frame. To secure X-MAC and A-MAC against unicast attacks, Hsueh et al. suggest exchanging random numbers during the wake-up frame/acknowledgment frame and probe frame/acknowledgment frame negotiation, respectively. These random numbers are used to derive a one-time password (OTP), which is then prefixed to the payload frame. While receiving the payload frame, receivers check if the prefixed OTP is valid and otherwise disable the receive mode immediately. Unfortunately, Hsueh et al.'s method is neither applicable to ContikiMAC nor CSL since, in ContikiMAC and CSL, frames are sent without prior negotiation. Also, a drawback of X-MAC compared to both ContikiMAC and CSL is the higher energy consumption per wake up [60]. Besides, a basic drawback of A-MAC compared to both ContikiMAC and CSL is that probe frames may interfere with the incoming transmissions of neighboring nodes. Performing a CCA before transmitting a probe frame mitigates this issue, yet not entirely due to the hidden node problem.

He et al. considered adopting a technique called frame masking for preventing droplet attacks [72, 169]. As per IEEE 802.15.4, O-QPSK SHRs are actually fixed. Nevertheless, CC2420 transceivers support changing an SHR's final two bytes to a custom value [138]. Accordingly, the idea of frame masking is to replace an SHR's final two bytes with a pseudo-random value that is derived from a pairwise key between the sender and the receiver, as well as the index of the current timeslot. In effect, receivers only detect IEEE 802.15.4 frames with valid "one-time SHRs". Yet, owing to deriving one-time SHRs from pairwise keys, frame masking lacks support for broadcast frames.

Similar concepts appeared in the context of wake-up receivers [42, 43, 44]. Falk et al. designed a wake-up receiver, which only wakes up upon receiving an OTP that is contained in a list of acceptable OTPs [42]. Yet, in Falk et al.'s design, nodes have to run an energy-consuming initialization protocol and senders and receivers can get out of sync [43]. Aljareh et al. solved both problems via time-synchronized OTPs [43]. Capossele et al. expanded on Falk et al.'s and Aljareh et al.'s work by (i) adding support for broadcast frames and (ii) obviating the need for time synchronization [44].

Raymond et al. devised a reactive defense against broadcast attacks, entitled Clustered Adaptive Rate Limiting (CARL) [174]. CARL detects broadcast attacks by caching whether recent wake ups led to receiving an inauthentic or replayed broadcast frame. In response to broadcast attacks, CARL lowers the wake-up interval. Furthermore, Raymond et al. addressed the problem that lowering the wake-up interval should be done in a coordinated manner as senders need to be aware of the wake-up interval of receivers in order to convey frames reliably. To this end, they suggest that nodes do not lower their wake-up interval immediately when they detect a broadcast attack. Instead, nodes that detected a broadcast attack and have a frame to send, send this frame and only thereafter lower their wake-up interval. Other nodes that also detected a broadcast attack wait until they receive a frame and thereupon lower their wake-up intervals. However, if a node neither receives nor sends a frame, the node eventually lowers its wake-up interval after a timeout. Yet, this way of coordinating the response to broadcast attacks (i) defers the reaction to broadcast attacks and (ii) does not exclude the possibility that some nodes temporarily use different wake-up intervals than other nodes, potentially leading to retransmissions of unicast frames, as well as to an increased loss of broadcast frames.

Raymond et al. also devised a reactive defense against jamming attacks [39]. In response to jamming attacks, Raymond et al. suggest switching to an LPM for some time span. Yet, their reactive defense against jamming attacks does not coordinate the response to jamming attacks. Hence, neighbors of victim nodes may retransmit pointlessly in the face of jamming attacks.

Chatty nodes are a generalization of PDoS attacks. In a PDoS attack, a malicious node sends packets to distant destinations, thereby causing each node along the path to expend energy for receiving, processing, and forwarding [108]. To counter PDoS attacks, many so-called en-route filtering schemes were proposed [108, 175, 176]. Their common idea is to add extra authentication tags to packets so that intermediary nodes can filter out packets en-route [108, 175, 176]. However, current en-route filtering schemes incur a high communication overhead and do not protect against chatty nodes that send fresh authentic frames to one-hop neighbors. Moreover, current en-route filtering schemes only filter out unwanted packets after having consumed energy for their reception already. Hence, we conjecture that a more lightweight, more holistic, as well as more effective defense would be to rate-limit the number of frames that a node is willing to receive from a particular neighbor. Furthermore, if a neighbor exceeded this rate, its frames could be rejected early during reception.

5.2.2 Transmission-Oriented Denial-of-Sleep Attacks

Ren et al. put forward a reactive defense against collision attacks [41]. They use an intricate threshold rule to detect collision attacks, as well as many other denial-of-sleep attacks. If a node detects any denial-of-sleep attack, they suggest switching to an LPM for some time span, similar to Raymond et al.'s reactive defense against jamming attacks. However, Ren et al. also neglected the problem that transmissions fail if a receiver is currently reacting to denial-of-sleep attacks. Moreover, as the neighbor of a victim node may retransmit pointlessly, the neighbor may suspect a collision attack and enter an LPM, too. Thus, a single denial-of-sleep attack may propagate through a whole network.

Song et al. defined delay attacks as attacks where an "attacker deliberately delays some of the time messages [...] so as to fail the time synchronization

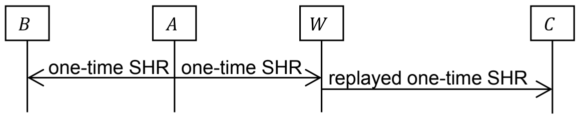


Figure 5.12: Susceptibility of one-time SHRs to hidden wormholes

process" [177]. Especially internal attackers can launch delay attacks since compromised nodes can send time messages at will [161]. Moreover, Ganeriwal et al. found that external attackers can launch pulse-delay attacks. In order to counter pulse-delay attacks, Ganeriwal et al. proposed the Secure Pairwise Synchronization Protocol (SPS) [156]. SPS does not actually prevent pulse-delay attacks, but confines the maximum time offset due to pulse-delay attacks.

5.2.3 Research Gaps

Synergistic integration of OTPs: An effective pattern for defending against broadcast, unicast, as well as droplet attacks emerged to be the embedding of OTPs in synchronization or frame headers. The general idea is to validate the embedded OTPs during reception and, if an OTP turns out invalid, to discard the incoming transmission without receiving the rest of it. For generating these OTPs, it is common to pass a key derivation function a symmetric key and a varying portion, such as a frame counter [42, 44], a timeslot index [43, 72], or a random number [40].

However, an open issue is the integration of session key establishment protocols, such as AKES, with an OTP-based defense against broadcast, unicast, and droplet attacks. More precisely, Hsueh et al. and Capossele et al. made proposals for such an integration [40, 44], but their proposals leave nodes vulnerable to broadcast, unicast, and droplet attacks during session key establishment. Moreover, they suggest establishing extra keys for the purpose of generating OTPs, rather than looking for synergies with protocols that establish session keys for basic wireless security anyway.

Practical implementation of OTPs: Another open research question is how to practically implement an OTP-based defense against broadcast, unicast, and droplet attacks. Indeed, the effectiveness of such defenses was shown in simulation studies [40, 44], but no practical implementation is available, yet. An exception is frame masking for which a prototype was implemented using CC2420 transceivers [72]. However, newer transceivers do not support changing SHRs anymore [14]. Even CC2420 transceivers have fairly limited support for changing SHRs - these transceivers can only scan for single SHR at a time. This restriction raises three vulnerabilities when trying to add support for broadcast frames to frame masking:

1. First, assume that the receivers of a broadcast frame are indefinite, e.g., during neighbor discovery. Then, the one-time SHR of the broadcast frame should be derived from a network-wide key. Furthermore, all nodes that currently listen for indefinite broadcast frames should be ready to accept that SHR. Attackers can exploit this by

replaying the one-time SHR of the broadcast frame in the same timeslot elsewhere. For example, in Figure 5.12, the one-time SHR sent from A to B is relayed by the hidden wormhole W to wake up C.

2. Second, if a broadcast frame is only destined to known neighbors $B_1, \dots, B_n$, one can derive its one-time SHR from a group key so as to limit the scope of the one-time SHR. However, $B_1, \dots, B_n$ may also wish to accept broadcast frames from their neighbors in the same timeslot and so on. In this case, one-time SHRs of targeted broadcast frames are to be derived from a network-wide key, too. This can, again, be exploited via a hidden wormhole.
3. Third, if a node accepts both unicast and broadcast frames in a single timeslot, unicast frames have to have the same one-time SHRs as broadcast frames. Consequently, in Figure 5.12, an eavesdropped one-time SHR of a unicast frame from A to B can also be misused to wake up C if both C and B accept broadcast frames in this timeslot.

Holistic defense against chatty nodes: Current en-route filtering schemes incur a high communication overhead, do not fully protect against chatty nodes, and only filter out unwanted packets after having consumed energy for their reception already. Hence, a more lightweight, more holistic, and more effective defense has to be developed.

Effective defense against collision attacks: Since Ren et al.'s defense against collision attacks may even aggravate collision attacks [41], an effective defense against collision attacks is yet to be found.

Lightweight defense against pulse-delay attacks: While Ganeriwal et al.'s SPS prevents pulse-delay attacks [156], it incurs a high communication overhead since it requires sending cryptographic random numbers in addition. Thus, a lightweight defense against pulse-delay attacks is needed.

First defenses against certain denial-of-sleep attacks: Thus far, there exist - to the best of our knowledge - no defenses against beacon jamming, interruption attacks, acknowledgment spoofing attacks, and deaf nodes.

5.3 Summary

In this chapter, we have reviewed the four - according to our impression - most prevalent MAC protocols for IEEE 802.15.4 networks regarding their denial-of-sleep resilience, namely ContikiMAC, CSL, TSCH, and LWB. A particularly severe denial-of-sleep attack we have identified is beacon jamming against LWB and TSCH. Moreover, securing the time synchronization of LWB and TSCH against internal attackers appears to be a hard-to-solve issue in its own right. Thus, our review has retained ContikiMAC and CSL as the most promising starting points for developing a denial-of-sleep-resilient MAC protocol.

Unfortunately, little can be drawn from prior work in the endeavor of securing ContikiMAC and CSL against denial-of-sleep attacks. On the one hand, there is the promising OTP-based approach to defending against broadcast, unicast, and droplet attacks. But, the practical implementation of OTPs, as well as their synergistic integration with basic wireless security constitute research gaps. On the other hand, Ganeriwal et al.'s SPS prevents pulse-delay attacks, but incurs a high communication overhead. As for all other denial-of-sleep attacks against ContikiMAC and CSL, it is necessary to develop denial-of-sleep defenses practically from scratch.

Chapter 6

Denial-of-Sleep Defenses for ContikiMAC

This chapter reports on our development steps toward a denial-of-sleep-protected version of ContikiMAC. For the scope of this chapter, we restrict ourselves to external attackers and also assume the use of group session keys. Later, in Chapter 7, we will widen our scope to internal attackers.

6.1 Overview

In sum, this chapter reports on our following development steps:

- For mitigating broadcast, unicast, as well as droplet attacks against ContikiMAC, we propose Practical On-the-fly Rejection (POTR). POTR adopts the emergent pattern of embedding OTPs, yet features three main novelties. First, rather than depending on special hardware, such as wake-up or CC2420 transceivers, POTR leverages the common capability of IEEE 802.15.4 transceivers of parsing incoming frames during reception. Second, POTR integrates with AKES so as to mitigate broadcast, unicast, and droplet attacks during session key establishment, too. Third, POTR uses the same keys as IEEE 802.15.4 security, thereby obviating the establishment of extra keys. Such a reuse of keys in different contexts often enables related-key attacks, but POTR avoids this potential vulnerability by staying within the framework of CCM.[1]
- To mitigate jamming attacks, as well as to further mitigate broadcast, unicast, and droplet attacks, we propose a tweak to ContikiMAC named

[1]We will present a revised version of POTR, which, unlike originally defined [6], integrates with CCM and uses LBCs to protect against broadcast, unicast, and droplet attacks during session key establishment. Originally, POTR did not integrate with CCM, but used an unproven method for preventing related-key attacks. Moreover, instead of LBCs, POTR originally used special OTPs for protecting against broadcast, unicast, and droplet attacks during session key establishment, which ensued two problems. First, since the insertion of PAN IDs would have delayed the validation of OTPs of `HELLO`s, `HELLO`s from co-located IEEE 802.15.4 networks were processed and answered with `HELLOACK`s. Second, since the special OTPs of `HELLOACK`s and `ACK`s did not change in each transmission and retransmission, it could happen that retransmitted `HELLOACK`s and `ACK`s were considered replayed and not acknowledged.

dozing optimization. The dozing optimization not only helps in mitigating jamming, broadcast, unicast, and droplet attacks, but also reduces ContikiMAC's energy consumption for receiving legitimate frames.

- To protect ContikiMAC against acknowledgment spoofing, pulse-delay, and collision attacks, we propose the Secure Phase-Lock Optimization (SPLO). To prevent pulse-delay attacks, in particular, SPLO secures acknowledgment frames much like Ganeriwal et al.'s SPS [156]. However, as opposed to SPS, SPLO prevents pulse-delay attacks without any communication overhead and is closer to practice.[2]

- We integrate the last bits (LB) optimization into IEEE 802.15.4 security, AKES, POTR, and SPLO [147, 178]. While the LB optimization is intended to reduce the per-frame overhead of a frame counter-based replay protection, we demonstrate that the LB optimization also helps in accelerating POTR's on-the-fly rejection of unwanted frames and hence in mitigating broadcast, unicast, as well as droplet attacks.[3]

- Next, we propose an intra-layer optimization between ContikiMAC and IEEE 802.15.4 security, named Intra-Layer Optimization for IEEE 802.15.4 Security (ILOS). Essentially, ILOS replaces frame counters with what we term wake-up counters. This brings several advantages. Most notably, ILOS achieves strong freshness, thus overcoming a basic issue with the frame counter-based replay protection of IEEE 802.15.4 security.

Besides, to prevent interruption attacks against ContikiMAC, we propose to not perform CCAs between successively strobed frames. While this increases the probability of inter-network collisions, intra-network collisions are still avoided by the two CCAs that precede each strobe. The existing open-source implementation of ContikiMAC already provides the possibility to not perform CCAs between successively strobed frames. Henceforth, we assume this configuration.

6.2 POTR: Practical On-the-Fly Rejection

In this section, we propose and evaluate POTR.

6.2.1 Design

POTR introduces a special frame format, as well as corresponding procedures for validating incoming frames during reception.

[2]We will present a revised version of SPLO, fixing two flaws within the initialization of wake-up times [5]. First, whereas, in the original version of SPLO, `ACK`s are acknowledged before checking their authenticity, in our revised version of SPLO the authenticity of `ACK`s is checked beforehand. This change is necessary to prevent acknowledgment spoofing attacks against `ACK`s. Second, whereas, in the original version of SPLO, the random number Q is generated by the `HELLOACK` receiver, in our revised version of SPLO the `HELLOACK` sender generates Q. This change is necessary to fully prevent pulse-delay attacks against `HELLOACK`s. Besides, in our revised version of SPLO, wake-up times are initialized earlier during AKES' three-way handshake. Hence, as opposed to the original version of SPLO, our revised version of SPLO also mitigates collision attacks against transmissions of `HELLOACK`s and `ACK`s.

[3]We will present a streamlined integration of the LB optimization. Whereas, for resynchronizing frame counters, we originally required receivers to respond to `UPDATE`s with `UPDATEACK`s [6, 7], here we piggyback the receiver's response on acknowledgment frames.

6.2.1.1 Frame Format

The intention behind POTR's special frame format is to reject unwanted IEEE 802.15.4 frames as early as possible during reception. The special frame format of POTR is shown in Figure 6.1. In comparison to the IEEE 802.15.4-compliant frame format shown in Figure 2.2b, POTR makes seven changes:

1. First, POTR uses the IEEE 802.15.4-compliant customization mechanism of extended frame types. Furthermore, in conjunction, the Extended Frame Type and Subtype fields imply which fields follows and whether an acknowledgment frame is expected, thereby obviating the Frame Control field. Overall, POTR supports eight kinds of frames, namely `HELLO`, `HELLOACK`, `ACK`, unicast data, unicast command, broadcast data, broadcast command, and acknowledgment frames. The difference between data and command frames is that data frames are passed to the upper layer, whereas command frames are processed at the MAC layer.

2. Second, POTR always elides the Destination Address and PAN ID fields in frames with OTPs. Instead, a frame with an OTP is assumed to be intended for a receiver if the receiver finds the OTP valid. Besides, in line with AKES, two simplifications of POTR are to assume (i) a network-wide agreement on using either short or extended addresses and (ii) that short addresses are unique within an IEEE 802.15.4 network.

3. Third, POTR reduces the Auxiliary Security Header field to the Frame Counter field, thus cutting off the Security Control and Key Identifier fields. Instead of sending the Security Control field, POTR assumes a preloaded security level. Alternatively, a security level may be negotiated during session key establishment, as well [179]. Instead of sending the Key Identifier field, POTR looks up keys by the sender's address and the frame's type. Finally, a crucial change to the Auxiliary Security Header field is that POTR increments the frame counter in retransmissions, too.

4. Forth, POTR adds the l-bit OTP field for embedding OTPs. The generation and validation of OTPs will be described shortly.

5. Fifth, POTR moves the Sequence Number field right behind the OTP field. This is because the validation of OTPs should happen as soon as possible, whereas the detection of duplicates has lower priority. Note that broadcast frames do not include a sequence number. This is because ContikiMAC never retransmits broadcast frames with our defense against interruption attacks anyway. Adding sequence numbers to `HELLOACK`s and `ACK`s is also unnecessary since AKES considers duplicated `HELLOACK`s and `ACK`s replayed and hence rejects them anyway.

6. Fifth, to avoid that a frame can fall between ContikiMAC's two regular CCAs, POTR adds a sufficient number of padding bytes to each frame.

7. Sixth, POTR removes the FCS field as all frames are authenticated via CCM MICs and thus integrity protected anyway. The securing of acknowledgment frames, however, is deferred to Section 6.4.

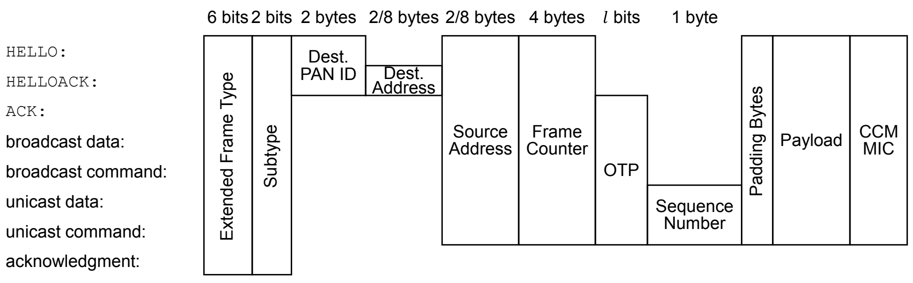


Figure 6.1: Frame format of POTR

Table 6.1: CCM Inputs as per POTR. C_A denotes A's frame counter, ID_B denotes B's MAC address, and ID_* denotes the broadcast MAC address

Occasion	CCM key	CCM nonce
OTP of a broadcast data or command frame from A	A's group session key $K_{A,*}$	$ID_*||C_A||$0xFF
OTP of a unicast data or command frame from A to B	A's group session key $K_{A,*}$	$ID_B||C_A||$0xFF
OTP of an ACK from A to B	Pairwise session key $K'_{A,B}$	$ID_B||C_A||$0xFF
Authentication and encryption of a HELLO, broadcast data, or broadcast command frame from A	A's group session key $K_{A,*}$	$ID_*||C_A||$0xFE
Authentication and encryption of a unicast data or command frame from A to B	A's group session key $K_{A,*}$	$ID_B||C_A||$0xFE
Authentication and encryption of a HELLOACK or ACK from A to B	Pairwise session key $K'_{A,B}$	$ID_B||C_A||$0xFE

6.2.1.2 On-the-Fly Rejection

When ContikiMAC detects the SHR of a frame, POTR takes over and performs the following checks during reception. Throughout, if any check fails, POTR disables the receive mode immediately, thereby stopping further energy loss.

Data and command frames: Upon reception of a data or command frame, POTR first ensures that the sender is stored as a permanent neighbor by inspecting the Source Address field. If so, POTR proceeds with generating the expected OTP and checks if it matches the one in the incoming frame. The OTP of a data or command frame is generated as the CCM MIC over the length of the data or command frame with the sender's group session key as CCM key and a special CCM nonce shown in Table 6.1. Finally, POTR makes sure that the frame is fresh and, if so, updates the anti-replay data about the sender.

ACKs: Analogously, upon reception of an ACK, POTR checks that the sender is stored as a tentative neighbor by inspecting the Source Address field. If true, POTR proceeds with generating the expected OTP and checks if it

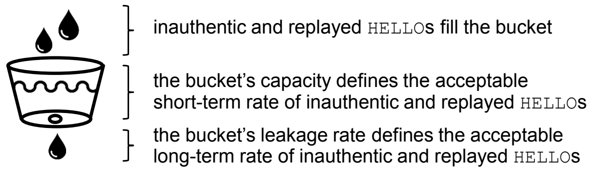

inauthentic and replayed HELLOs fill the bucket
the bucket's capacity defines the acceptable short-term rate of inauthentic and replayed HELLOs
the bucket's leakage rate defines the acceptable long-term rate of inauthentic and replayed HELLOs

Figure 6.2: Mitigation of broadcast and droplet attacks with `HELLO`s

matches the one in the incoming frame. The OTP of an `ACK` is generated as the CCM MIC over the length of the `ACK` with the pairwise session key as CCM key and a special CCM nonce shown in Table 6.1 (In spite of establishing group session keys, AKES temporarily generates a pairwise session key we can use at this point.). Finally, POTR makes sure that the frame is fresh and, if so, updates the anti-replay data about the sender.

HELLOs: Upon reception of a `HELLO`, POTR first ensures that the `HELLO` is not destined to a co-located IEEE 802.15.4 network by inspecting the Destination PAN ID field. Next, POTR checks that the length of the `HELLO` complies with POTR's frame format. Then, if the sender of the `HELLO` is not a permanent neighbor, POTR ascertains that the sender is not a tentative neighbor already, as well as that AKES can reply with a `HELLOACK` at all. In fact, AKES may be unable to reply with a `HELLOACK` because AKES may run out of memory and because AKES restricts itself to sending a maximum rate of `HELLOACK`s. Note, however, that AKES is interested in receiving `HELLO`s from permanent neighbors since AKES uses such `HELLO`s to trigger a reestablishment of session keys after reboots, as well as to suppress redundant `HELLO`s. Lastly, POTR makes sure that receiving the `HELLO` does not result in exceeding a maximum rate of incoming `HELLO`s. POTR implements this rate limitation via an LBC, as illustrated in Figure 6.2. Yet, to avoid penalizing fresh authentic `HELLO`s, the LBC is decremented again if a `HELLO` turns out authentic and fresh.

HELLOACKs: Upon reception of a `HELLOACK`, POTR ascertains that the `HELLOACK` is not destined to another node and that its length complies with POTR's frame format. Also, POTR ensures that AKES recently sent a `HELLO` at all and that AKES' LBC for rate-limiting outgoing `ACK`s is not full at the moment. Finally, POTR makes sure that receiving the `HELLOACK` does not ensue exceeding a maximum rate of incoming `HELLOACK`s. Again, POTR implements this rate limitation via another LBC and decrements that LBC if a `HELLOACK` turns out authentic and fresh.

Acknowledgment frames: Upon reception of an acknowledgment frame, POTR just ensures that its length is compliant with POTR's frame format.

6.2.2 Security Analysis

Below, we analyze the applicability of various attacks against POTR.

6.2.2.1 Replay Attacks

POTR takes over the task of filtering out replayed data, command, and `ACK` frames from IEEE 802.15.4 security and fulfills this task already during reception. Hence, a first aspect that is worth analyzing is the applicability of replay attacks. Assume that a node A receives a replayed data, command, or `ACK` frame that originated from a node B. Only if B is still stored as a permanent or tentative neighbor and the CCM key from which the replayed OTP was derived has not changed in the meantime, A may find the OTP of the replayed frame valid. Yet, A will eventually reject the replayed frame since the contained frame counter will not be greater than that of the last accepted frame from B. On the other hand, if A missed the original frame and any subsequent frame from B, A may find the replayed OTP valid and fresh, but only once. Also, note that if an attacker replays an OTP as part of a frame that is longer than the frame wherein the OTP was originally embedded, the OTP will become invalid.

6.2.2.2 Guessing Attacks

The probability of guessing an OTP right is 2^{-l}. A repercussion of a correctly guessed OTP is not only that the attacker's frame is fully received, but also that the respective anti-replay data gets corrupted. Thus, after a successful guessing attack, legitimate data, command, and `ACK` frames may be considered replayed. In the worst case, the affected nodes need to reestablish session keys so as to recover from such a situation. If a reestablishment of session keys becomes necessary, AKES will automatically do so since AKES deletes inactive permanent neighbors and continuously discovers new neighbors. We suggest setting $l = 24$ to trade off the repercussions of successful guessing attacks against the per-frame overhead of longer OTPs. Shorter values of l become reasonable when deriving OTPs from wake-up counters, as we will discuss in Section 6.6.

6.2.2.3 Broadcast, Unicast, and Droplet Attacks

Data, command, and `ACK` frames: Let us now consider a broadcast, unicast, or droplet attack with a data, command, or `ACK` frame. If POTR finds the OTP of such a frame invalid or replayed, POTR will discard the frame early during reception, thereby stopping further energy loss and preventing ContikiMAC from sending an acknowledgment frame, too. POTR does, however, not prevent ContikiMAC from staying in receive mode during the time span between a negative CCA and the detection of a frame's SHR. During this time span, ContikiMAC still only leaves the receive mode if the fast-sleep optimization decides so. This is a basic limitation of POTR, which we analyze and address in Section 6.3.

HELLOs and HELLOACKs: As for broadcast, unicast, and droplet attacks with `HELLO`s or `HELLOACK`s, the protection provided by POTR is weaker. This is because POTR accepts such frames up to a configurable rate via LBCs. Nevertheless, even when configuring POTR's LBCs to accept `HELLO`s and `HELLOACK`s at moderate rates, the level of protection provided by POTR is relatively strong. For illustration, let us compare the time that ContikiMAC stays in receive mode anyway for doing two CCAs every $t_w = 125$ms with the time it takes to receive 64-byte frames at the rate of

$\frac{1}{15}$Hz. When using the 2.4-GHz O-QPSK PHY, the time in receive mode per CCA is 0.128ms as per IEEE 802.15.4. Thus, on a percentage basis, ContikiMAC's base time in receive mode is $100\% \times \frac{0.128\text{ms}\times 2}{t_w} = 0.205\%$. On the other hand, receiving a 64-byte frame plus the 6-byte O-QPSK PHY header takes $\frac{70\text{byte}\times 8\frac{\text{bit}}{\text{byte}}}{250\frac{\text{kbit}}{\text{s}}} = 2.24\text{ms}$. Altogether, permitting ContikiMAC to receive 64-byte frames at a rate of $\frac{1}{15}$Hz only adds 0.016% to the base time in receive mode.

Acknowledgment frames: As for acknowledgment frames, POTR protects against an increased energy consumption due to over-long acknowledgment frames by validating the length of acknowledgment frames.

6.2.2.4 Hidden Wormholes

An idea to bypass POTR's checks is to set up a hidden wormhole. Indeed, an attacker can, e.g., relay `HELLO`s, `HELLOACK`s, and `ACK`s, thereby causing nodes at the other end of the hidden wormhole to receive these frames. Moreover, if an attacker subsequently also tunnels data and command frames, victim nodes need to receive such frames, too. Fortunately, since AKES limits the number of outgoing `HELLO`s, `HELLOACK`s, and `ACK`s, the number of additionally received `HELLO`s, `HELLOACK`s, and `ACK`s under hidden wormhole attacks is limited, as well. Furthermore, only `UPDATE`s are currently sent as command frames and at a low rate. The rate of incoming data frames, on the other hand, depends on the upper layers, which may include their own defenses against hidden wormholes. For example, RPL uses a hysteresis to mitigate hidden wormholes [180].

6.2.3 Implementation

We first integrated POTR with the existing open-source implementation of ContikiMAC [60]. However, we encountered two severe issues with this implementation of ContikiMAC. First, it blocks all other Protothreads while sending frames. Second, as also reported by Uwase et al. [181], it is subject to timing issues. In view of these two issues, we eventually opted for reimplementing ContikiMAC entirely. Unlike the existing open-source implementation of ContikiMAC, our reimplementation allows other Protothreads to progress while sending and receiving frames. For this, we make use of an enriched `RADIO` driver, which exposes `SFD`, `FIFOP`, and `TXDONE` interrupts of CC2538 SoCs. `SFD` interrupts issue when an SHR was detected. `FIFOP` interrupts issue as soon as `FIFOP_THR` bytes can be parsed or when a frame was completely received (The `FIFOP` interrupt has a different behavior when enabling the in-built frame filtering of the CC2538 transceiver. However, since the in-built frame filtering of CC2538 transceivers validates against the original IEEE 802.15.4 frame format and not POTR's frame format, our enriched `RADIO` driver disables the in-built frame filtering of CC2538 transceivers). `TXDONE` interrupts issue when a frame was sent [14]. While other hardware platforms may not offer `SFD`, `FIFOP`, and `TXDONE` interrupts, one may still be able to use our ContikiMAC implementation unmodified by emulating these interrupts in software. Besides, we use Contiki-NG's `rtimer` module for scheduling wake ups, transmissions, and other events.

Within the `FIFOP` ISR, POTR performs its on-the-fly rejection. For example, if it comes to validating an OTP, POTR (i) generates the expected OTP, (ii)

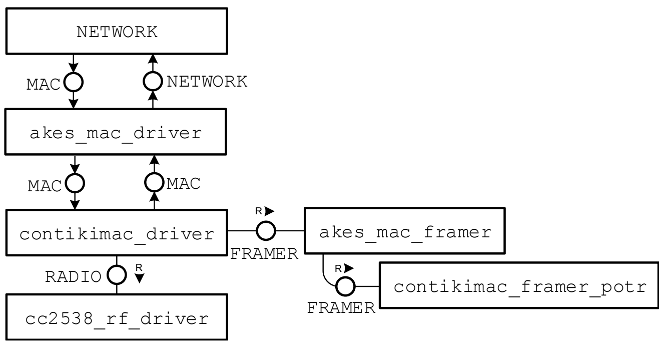


Figure 6.3: Integration of our reimplementation of ContikiMAC into Contiki-NG's network stack, as well as into our implementation of AKES

busy-waits for the OTP field to arrive, and (iii) checks if both OTPs match. Thus, the validation and reception of an OTP happens in parallel. Yet, a major difficulty with implementing the on-the-fly rejection of POTR was concurrency. This is because the bulk of Contiki-NG's code can not be called from within an ISR. To resolve this problem, we added locks so as to protect code regions that should not be interrupted by POTR. Besides, we adapted Contiki-NG's `packetbuf` module to support parsing incoming frames within an ISR.

Figure 6.3 depicts how our reimplementation of ContikiMAC integrates into Contiki-NG's network stack and our implementation of AKES. At the `RADIO` layer, we use our enriched `cc2538_rf_driver`. At the `MAC` layer, the `akes_mac_driver` decorates our reimplementation of ContikiMAC, which is contained in the `contikimac_driver`. For assembling and parsing frames, the `contikimac_driver` calls the `akes_mac_framer`, which, in turn, calls the `contikimac_framer_potr`. Optionally, POTR can be disabled, in which case an IEEE 802.15.4-compliant `FRAMER` is called instead of the `contikimac_framer_potr`. Also, it is possible to run our reimplementation of ContikiMAC without AKES, in which case the `contikimac_driver` and the IEEE 802.15.4-compliant `FRAMER` are called directly, i.e., without passing through AKES' modules.

6.2.4 Evaluation

Using our implementation, we now demonstrate that, in comparison to using no denial-of-sleep defense, POTR reduces the time that ContikiMAC spends in receive mode in the face of unicast attacks with unicast data frames significantly. Thereafter, we also measure the actual energy savings under broadcast and unicast attacks with data frames. Lastly, we determine that POTR actually saves program memory as compared to not using POTR.

6.2.4.1 Rejection Speed

As a baseline for comparison, it was first measured how long it takes to receive a maximum-length frame of 127 bytes. Concretely, an OpenMote A sent maximum-length frames to another OpenMote B. During 100 receptions, B logged the time between when a frame's SHR was detected, which is signaled by `SFD` interrupts, and when a frame was completely received, which is signaled

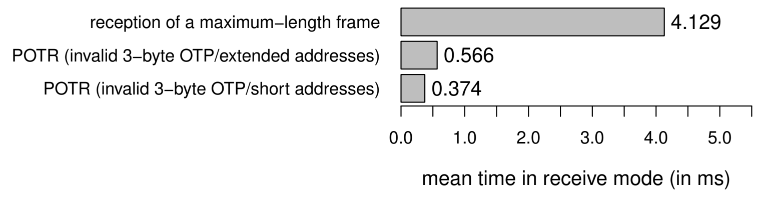


Figure 6.4: Rejection speed of POTR

by `FIFOP` interrupts. The mean time between these two instants turned out to be 4.129ms. This result accords with our expectations since the 2.4-GHz O-QPSK PHY has a data rate of 250kbit/s. Furthermore, the 1-byte Frame Length field of the O-QPSK PHY header has to be received in addition to the 127 bytes and hence the time between the detection of a frame's SHR and its complete reception should theoretically be $\frac{(1\text{byte}+127\text{byte})\times 8\frac{\text{bit}}{\text{byte}}}{250\frac{\text{kbit}}{\text{s}}} = 4.096\text{ms}$.

Next, to measure the time until POTR rejects a frame due to an invalid OTP, an OpenMote A injected maximum-length unicast data frames, each of which contained the source address of a permanent neighbor of another OpenMote B, as well as an invalid 3-byte OTP. The OpenMote B, which had POTR enabled, received 100 of these frames and logged the time between `SFD` interrupts and the moment when the frame reception aborts, which is signaled by `RXABO` interrupts. This experiment was repeated for both short and extended addresses.

Figure 6.4 shows how quick B rejects the injected 127-byte frames. The employed addressing mode has a noticeable effect on the rejection speed. This is because POTR starts validating OTPs earlier or later, depending on the employed addressing mode. Thus, it is advisable to use short addresses.

6.2.4.2 Energy Savings

To determine the energy consumption of ContikiMAC under broadcast attacks with versus without POTR, an OpenMote A sent maximum-length broadcast data frames to another OpenMote B, which was placed 50cm apart from A. B rejected these frames due to an invalid 3-byte OTP or an inauthentic CCM MIC when using POTR or not, respectively. During 200 receptions, the energy consumed by B was recorded using the experimental setup described in Section 3.6.2. To avoid bias, A sent at randomized times. This experiment was repeated with short and extended addresses, as well as with maximum-length unicast data frames instead of maximum-length broadcast data frames. Lastly, as a baseline for comparison, B's energy consumption when performing ContikiMAC's two CCAs and directly going back to sleep was measured, too.

Figure 6.5 shows the results. When not using POTR, the energy consumption under broadcast attacks with data frames increases a lot compared to ContikiMAC's baseline energy consumption. This is because, when not using POTR, injected and replayed broadcast frames are fully received before being rejected. Using short instead of extended addresses does not mitigate broadcast attacks since we always padded the injected broadcast frames to the maximum frame length like an attacker would do, as well. Moreover, under a unicast attack, a victim node may not only fully receive the injected or replayed unicast frame, but additionally send an acknowledgment frame. This actually happened

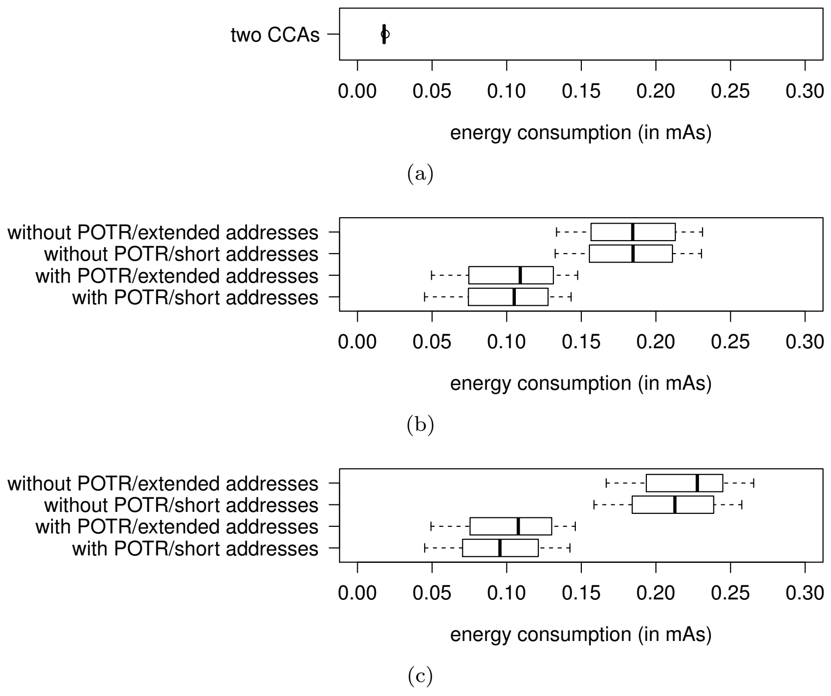


Figure 6.5: Energy consumption per wake up (a) if both CCAs return positive, (b) in the face of a broadcast attack, and (c) in the face of a unicast attack

in this experiment. Also, as acknowledgment frames extend when using extended addresses, the incurred energy consumption of unicast attacks increases in this case. When using POTR, the energy consumption under broadcast and unicast attacks is much lower since POTR rejects the injected broadcast and unicast data frames early during reception. Hence, POTR also avoids sending acknowledgment frames in response to the injected unicast frames. When using short addresses instead of extended addresses, the effectiveness of POTR increases because POTR then rejects frames with invalid OTPs earlier.

6.2.4.3 Processing Overhead

To showcase the processing overhead of POTR, the following experiment measures the time spent in the `FIFOP` ISR when successfully validating an OTP. Concretely, an OpenMote A sent maximum-length unicast data frames to another OpenMote B. A and B were permanent neighbors of each other and the frames contained valid and fresh OTPs. During each reception, B logged the time spent in the `FIFOP` ISR. This experiment was repeated with both short and extended addresses, as well as with different OTP lengths $l \in \{8, 16, 24, 32, 40\}$.

Figure 6.6 shows the results. The workload of the `FIFOP` ISR increases as l exceeds 24. This is because, at this point, the reception of OTPs takes longer than their validation, thus necessitating busy-waiting within the `FIFOP` ISR. One can minimize busy-waiting by configuring the `FIFOP` interrupt to issue later. Currently, our implementation is optimized for $l = 24$. Another observation is that short addresses slightly reduce the workload of the `FIFOP` ISR, which can

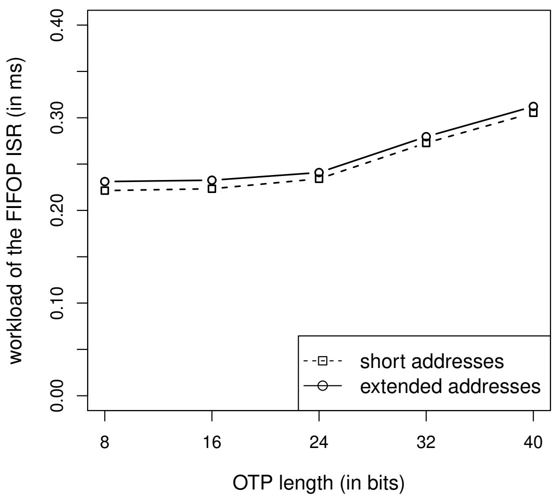


Figure 6.6: Workload of the `FIFOP` ISR when successfully validating OTPs

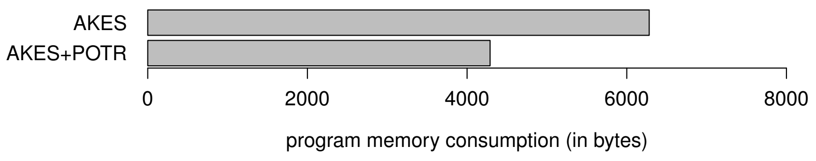


Figure 6.7: Program memory overhead due to POTR

be attributed to the fact that less bytes are being processed.

6.2.4.4 Program Memory Overhead

To measure the overhead in program memory due to POTR, it was first measured how much program memory is consumed when using ContikiMAC without MAC layer security using the tool `arm-none-eabi-size`. Then, the overhead in program memory (i) due to enabling AKES, but not POTR and (ii) due to enabling both AKES and POTR was measured. Throughout, short addresses, Security Level 6, and 3-byte OTPs were used.

Figure 6.7 shows the results. It turned out that POTR actually saves program memory, which can, on the one hand, be attributed to the streamlined frame format of POTR, and, on the other hand, to Contiki-NG's currently unoptimized assembling and parsing of IEEE 802.15.4-compliant frames.

6.2.5 Discussion

We have argued and partly demonstrated that POTR greatly mitigates broadcast, unicast, and droplet attacks against ContikiMAC. However, two limitations of POTR have become apparent in this section. First, if an attacker guesses an OTP right, the repercussion is not only that the attacker's frame is fully received, but also that the respective anti-replay data gets corrupted. The corruption of anti-replay data upon successful guessing attacks is not a specific issue to POTR, but regularly arises when deriving OTPs from frame counters [42, 44]. Second, POTR just mitigates broadcast, unicast, and droplet attacks. We work on both these limitations in subsequent development steps.

6.3 The Dozing Optimization

In this section, we propose the dozing optimization. The dozing optimization complements POTR very well since it further mitigates broadcast, unicast, and droplet attacks, as well as adds protection against jamming attacks.

6.3.1 Design

We will denote by:

- t_l the time to transmit a maximum-length IEEE 802.15.4 frame. When using the 2.4-GHz O-QPSK PHY, t_l is 4.256ms because (i) the 2.4-GHz O-QPSK PHY has a data rate of 250kbit/s, (ii) the length of the O-QPSK PHY header is 6 bytes and (iii) the maximum frame length that O-QPSK supports is 127 bytes.

- t_r the duration of one CCA. When using the 2.4-GHz O-QPSK PHY, t_r is 0.128ms since a CCA shall take 8 symbol periods as per IEEE 802.15.4.

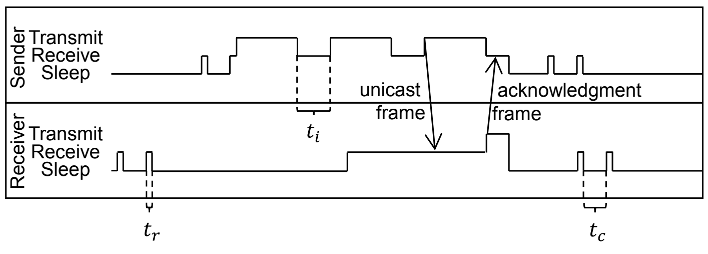
- t_c the configurable time span between ContikiMAC's two regular CCAs, as depicted in Figure 6.8a.

- t_i ContikiMAC's inter-frame period, as shown in Figure 6.8a. Though t_i is configurable, t_i can not be set arbitrarily short to allow for sending and detecting acknowledgment frames between successively strobed unicast frames. Neither can t_i be set arbitrarily long as otherwise ContikiMAC's two regular CCAs may fall between successively strobed frames.

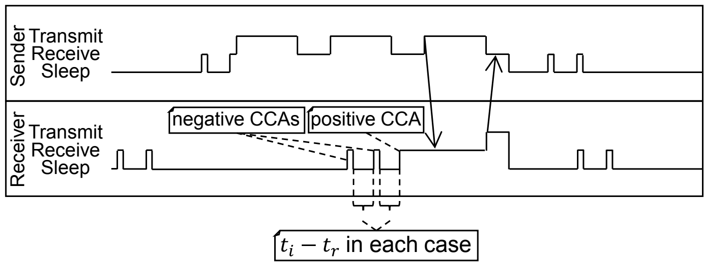
- t_d the time to detect an SHR. When using 2.4-GHz O-QPSK, t_d is 0.16ms.

- t_p the time that POTR needs to decide if a frame is to be rejected once ContikiMAC detected its SHR. The actual value of t_p depends on the addressing mode, the length of OTPs, as well as the PHY. As previously demonstrated, t_p can get as low as 0.374ms.

Recall that ContikiMAC's fast-sleep optimization lets receivers go back to sleep (i) after t_l if the radio noise lasts longer than t_l, (ii) after $t_l + t_i$ if the silence period lasts longer than t_i, and (iii) after $t_l+t_i+t_d$ if no SHR is detected. Attackers can misuse this behavior by launching jamming, broadcast, unicast, and droplet attacks. POTR mitigates broadcast, unicast, and droplet attacks by rejecting an unwanted frame t_p after the detection of its SHR. Yet, in the worst case, an attacker can still cause ContikiMAC to stay in receive mode for $t_r + t_r + t_l + t_i + t_d + t_p = w_{\text{original}}(t_i)$, where "$t_r$+" accounts for the occasion when the first of ContikiMAC's two regular CCAs returns positive and "+t_r+" allows for a minimal overlap between the ending of the second CCA and the



(a)



(b)

Figure 6.8: Operation of a unicast transmission in ContikiMAC (a) without the dozing optimization and (b) with the dozing optimization

beginning of a frame. Analogously, the maximum time that ContikiMAC stays in receive mode for receiving a legitimate frame is $w_{\text{original}}(t_i) - t_d - t_p + t_l$.

The operation of the dozing optimization is exemplified in Figure 6.8b. As opposed to the original design of ContikiMAC, if a CCA returns negative, the dozing optimization goes back to sleep and schedules another CCA after $t_i - t_r$. If this CCA also returns negative, the dozing optimization goes back to sleep again and performs another CCA after $t_i - t_r$ and so on. In accordance with the fast-sleep optimization, if the radio noise lasts longer than t_l, the dozing optimization schedules no further CCAs. If, however, a subsequent CCA returns positive, the dozing optimization stays in receive mode as this indicates that a silence period between two successively strobed frames is found. Also, in accordance with the fast-sleep optimization, a receiver goes back to sleep if no radio noise is detected after $t_l + t_i$ and if no SHR is detected after $t_l + t_i + t_d$. Essentially, the dozing optimization thus mimics the fast-sleep optimization except for dozing after a negative CCA when searching a silence period.

Observe that, if t_i is long, the dozing optimization dozes longer and hence performs less CCAs, while, if t_i is short, the dozing optimization needs to spend less time in receive mode once a CCA returns positive. Concretely, under attack, the dozing optimization performs up to $2 + \lceil \frac{t_l}{t_i} \rceil + 1$ CCAs and stays at most $t_i + t_d + t_p$ in receive mode. Again, "2+" accounts for a positive first CCA and a negative second CCA. Then, up to $\lceil \frac{t_l}{t_i} \rceil$ negative CCAs and one positive CCA may follow. Next, the dozing optimization stays up to $t_i + t_d + t_p$ in receive mode. Altogether, the worst-case time in receive mode in the adversarial case is $(3 + \lceil \frac{t_l}{t_i} \rceil) \times t_r + t_i + t_d + t_p = w_{\text{dozing}}(t_i)$. Similarly, upon receiving a legitimate frame, the dozing optimization stays in receive mode for at most $w_{\text{dozing}}(t_i) - t_d - t_p + t_l$. As shown in Figure 6.9a, $w_{\text{dozing}}(t_i)$ has a minimum at

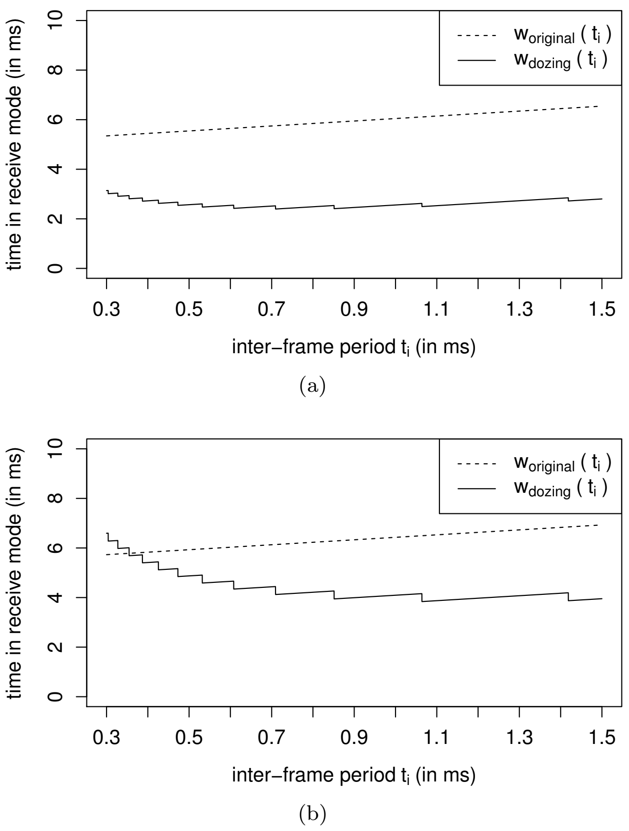


Figure 6.9: Worst-case duration that ContikiMAC stays in receive mode in the face of broadcast, unicast, and droplet attacks when using POTR alone ($w_{\text{original}}(t_i)$) and when using the dozing optimization in addition ($w_{\text{dozing}}(t_i)$) (a) if $t_l = 4.256$ms, $t_r = 0.128$ms, $t_d = 0.16$ms, and $t_p = 0.374$ms and (b) $t_l = 4.256$ms, $t_r = 0.32$ms, $t_d = 0.16$ms, and $t_p = 0.374$ms

$t_i = 0.7094$ when using the 2.4-GHz O-QPSK PHY. Figure 6.9a also shows that the dozing optimization greatly reduces the time spent in receive mode in the face of broadcast, unicast, and droplet attacks compared to using POTR alone.

6.3.2 Implementation

We integrated the dozing optimization into our implementation of ContikiMAC for CC2538 SoCs. However, unlike standardized in IEEE 802.15.4, the CC2538 needs 0.32ms per CCA. This is because the CC2538 stays 0.192ms in an energy-consuming intermediate state before it actually starts with a CCA. This affects the functions $w_{\text{original}}(t_i)$ and $w_{\text{dozing}}(t_i)$ in Figure 6.9a. Figure 6.9b shows the updated functions. Now, $w_{\text{dozing}}(t_i)$ has a minimum at 1.064ms. Thus, 1.064ms constitutes the optimal value for t_i in terms of worst-case duration in receive mode. However, owing to the limited precision of Contiki-NG's `rtimer` module, our implementation rounds up to $t_i = 1.068$ms.

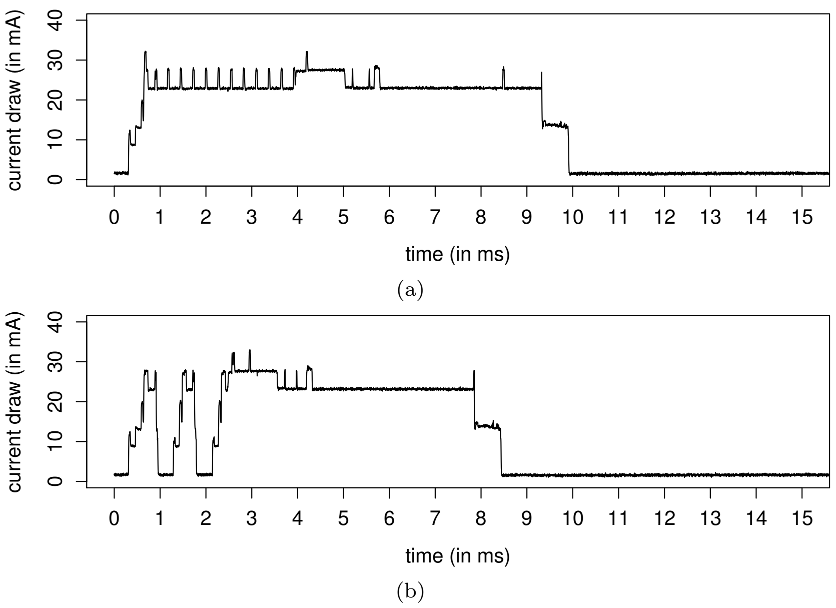


Figure 6.10: Current draw while receiving a maximum-length broadcast data frame when the dozing optimization is (a) off and (b) on

6.3.3 Evaluation

Using our implementation, we now demonstrate that the dozing optimization (i) noticeably reduces the energy consumption for receiving legitimate frames, (ii) significantly reduces the energy consumption in the face of jamming attacks, as well as in the face broadcast and unicast attacks with data frames, and (iii) practically incurs no overhead in program memory.

6.3.3.1 Energy Consumption of Frame Receptions

To compare the energy consumption for receiving legitimate frames with versus without the dozing optimization, an OpenMote A sent fresh authentic maximum-length broadcast data frames to another OpenMote B. B was placed 50cm away from A. During 200 receptions, the current draw of B was recorded like described in Section 3.6.2. This experiment was conducted two times for the cases of (i) disabling the dozing optimization and (ii) enabling the dozing optimization. Throughout, POTR was disabled, short addresses were used, A sent at randomized times to avoid bias, and Security Level 6 was configured.

Figure 6.10a exemplifies a frame reception, where the dozing optimization is off. Here, the first of ContikiMAC's two regular CCAs returns negative, causing B to stay in receive mode and to periodically perform CCAs so as to find a silence period. As B finds a silence period at time 4ms, B stops performing CCAs and enables the SHR search of the CC2538. It would also be possible to enable the SHR search of the CC2538 right from the beginning, but then chances are that an SHR is found within radio noise or the previously transmitted frame [81]. Then, B stays in receive mode to receive the approaching maximum-length broadcast data frame. Lastly, B processes the frame and enters an LPM.

Figure 6.10b shows a frame reception, where the dozing optimization is on. Since the first of the two regular CCAs returns negative, B starts dozing. The second CCA also returns negative, which is why B continues to doze. Eventually, the third CCA returns positive, causing B to stay in receive mode. At this point, B also enables the SHR search of the CC2538. Again, the SHR search could be enabled in the first place. Despite this, we disable the SHR search during CCAs since this improves the delivery ratio for unknown reasons. Finally, B processes the received maximum-length broadcast data frame and enters an LPM.

Figure 6.11 shows boxplots of the energy consumption per reception of a fresh authentic maximum-length broadcast data frame. Accordingly, the dozing optimization saves a good amount of energy.

6.3.3.2 Mitigation of Jamming, Broadcast, and Unicast Attacks

To determine the incurred energy consumption of jamming attacks, as well as of broadcast and unicast attacks with data frames, the above experiment was adapted as follows. As for jamming attacks, A continuously emitted radio noise via a special transmit mode of its CC2538. In four subsequent runs, B was programmed to use (i) neither POTR nor the dozing optimization, (ii) not POTR, but the dozing optimization, (iii) POTR, but not the dozing optimization, and (iv) both POTR and the dozing optimization. In each run, 200 traces of B's current draw per wake up were recorded. As for broadcast attacks, the procedure was the same, except that A sent maximum-length broadcast data frames that are rejected by B due to invalid 3-byte OTPs or inauthentic MICs when using POTR or not, respectively. Likewise, as for unicast attacks, A sent maximum-length unicast data frames that are rejected by B due to invalid 3-byte OTPs or inauthentic MICs when using POTR or not, respectively.

Figure 6.12a shows the results for jamming attacks. Without any defense, ContikiMAC's fast sleep optimization leaves the receive mode only after $t_l = 4.256\text{ms}$. This causes a mean energy consumption of 0.103mAs. By contrast, if the dozing optimization is enabled, jamming attacks are much less severe. Expectably, POTR does not help in mitigating jamming attacks since POTR only comes into effect once an SHR is detected.

Figure 6.12b shows the results for broadcast attacks. If using no defense or only the dozing optimization, we get similar results as in Figure 6.11. On the one hand, this is because even frames with inauthentic MICs are fully received and validated before they get rejected. On the other hand, we disabled upper-layer protocols in both experiments. Normally, the processing of fresh authentic data frames is more energy consuming. Enabling POTR mitigates broadcast attacks greatly since POTR rejects frames with invalid OTPs on the fly. Even more energy can be saved by enabling both POTR and the dozing optimization.

Figure 6.12c shows the results for unicast attacks. In contrast to broadcast attacks, unicast attacks may cause victim nodes to send acknowledgment frames. This actually happened in our experiment because, if POTR is off, even injected and replayed unicast frames are being acknowledged if they pass basic validity checks. Hence, unicast attacks can cause a higher energy consumption than broadcast attacks if POTR is off. If POTR is on, the maximum-length unicast data frames are rejected on the fly and thus no acknowledgment frames are triggered. The dozing optimization further mitigates unicast attacks.

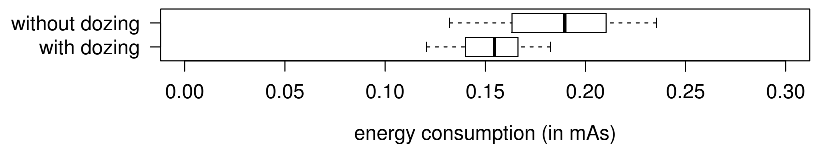


Figure 6.11: Energy consumption per reception of a fresh authentic maximum-length broadcast frame with vs without the dozing optimization

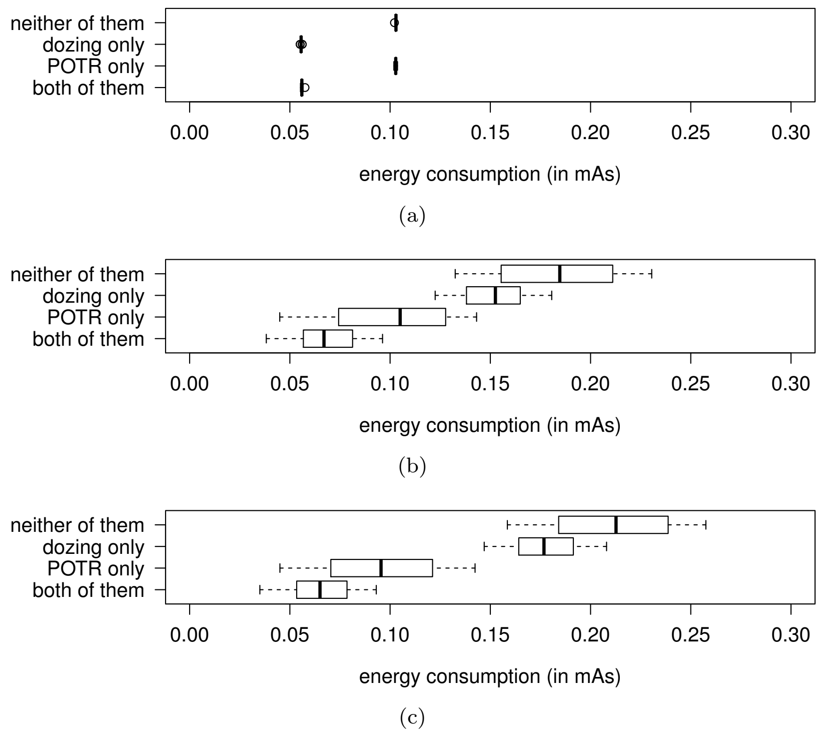


Figure 6.12: Energy consumption per wake up under (a) jamming, (b) broadcast, and (c) unicast attacks with vs without the dozing optimization

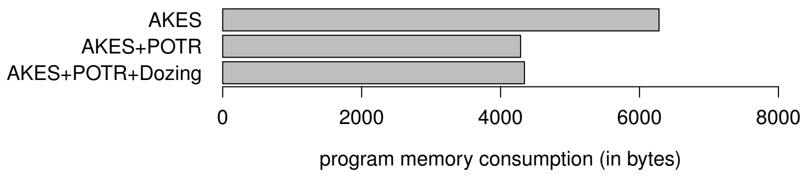


Figure 6.13: Program memory overhead due to the dozing optimization

6.3.3.3 Program Memory Overhead

To measure the overhead in program memory due to the dozing optimization, it was first measured how much program memory is consumed when using ContikiMAC without MAC layer security. Subsequently, the overhead in program memory due to enabling AKES, POTR, and the dozing optimization was measured. Short addresses, Security Level 6, and 3-byte OTPs were used.

The result is shown in Figure 6.13 alongside the results obtained for POTR. Maybe surprisingly, the dozing optimization incurs a marginal overhead in program memory of 56 bytes. The reason for this is that the dozing optimization only requires few modifications to the original version of ContikiMAC.

6.3.4 Discussion

To further mitigate broadcast, unicast, and droplet attacks, as well as to mitigate jamming attacks, we have proposed the dozing optimization. Both in theory and in practice, the dozing optimization has turned out to be very effective in mitigating these attacks. Beyond that, we have shown the dozing optimization to also reduce the energy consumption for receiving legitimate frames. However, even when combining POTR and the dozing optimization, the energy consumption of victim nodes still increases under jamming, broadcast, unicast, and droplet attacks, thus still leaving room for improvements.

6.4 SPLO: Secure Phase-Lock Optimization

In this section, we propose SPLO, which makes ContikiMAC resistant to acknowledgment spoofing and pulse-delay attacks, as well as resilient to collision attacks. We will first go into the design of SPLO and then evaluate SPLO.

6.4.1 Design

We split the description of the design of SPLO into three parts. First, we explain how SPLO secures acknowledgment frames that are sent in response to unicast data and command frames. Second, we explain how SPLO mitigates collision attacks. Third, we explain how SPLO securely initializes wake-up times in parallel to AKES' three-way handshake. This involves special means to secure acknowledgment frames that are sent in response to `HELLOACK`s and `ACK`s.

6.4.1.1 Securing Acknowledgment Frames

While IEEE 802.15.4 security can be used "as is" to secure acknowledgment frames, the security provided by IEEE 802.15.4 security is limited in two ways. First, IEEE 802.15.4 security does not achieve strong freshness. Second, IEEE 802.15.4 security does not ensure the "correspondence" between unicast and acknowledgment frames, i.e., whether an acknowledged unicast frame was actually received successfully. As these two limitations enable pulse-delay and acknowledgment spoofing attacks, SPLO takes over the task of securing acknowledgment frames that are sent in response to unicast data and command frames and not only ensures their authenticity and sequential freshness, but also their strong freshness and correspondence.

Table 6.2: CCM Inputs as per SPLO. C_A is A's frame counter, ID_B is B's MAC address, ID_* is the broadcast MAC address, and $\kappa <$ 0xFE is the strobe index.

Occasion	CCM key	CCM nonce
OTP of a broadcast data or command frame from A	A's group session key $K_{A,*}$	$ID_* | C_A |$0xFF
OTP of a unicast data or command frame from A to B	A's group session key $K_{A,*}$	$ID_B | C_A |$0xFF
OTP of an `ACK` from A to B	Pairwise session key $K'_{A,B}$	$ID_B | C_A |$0xFF
Authentication and encryption of a `HELLO`, broadcast data, or broadcast command frame from A	A's group session key $K_{A,*}$	$ID_* | C_A |$0xFE
Authentication and encryption of a unicast data or command frame from A to B	A's group session key $K_{A,*}$	$ID_B \| C_A \| \kappa$
Authentication and encryption of a `HELLOACK` or `ACK` from A to B	Pairwise session key $K'_{A,B}$	$ID_B \| C_A \| \kappa$
Authentication of an acknowledgment frame from B to A	Same as the corresponding unicast frame	$ID_A \| C_A \| \kappa$

For authenticating acknowledgment frames that are sent in response to unicast data and command frames, SPLO appends a MIC to each such acknowledgment frame. These MICs are generated using CCM. As CCM key, SPLO uses the same as was used for securing the unicast data or command frame whose reception is being acknowledged. As CCM nonce, SPLO uses the concatenation of (i) the MAC address of the receiver of the acknowledgment frame, (ii) the frame counter of the unicast data or command frame whose reception is being acknowledged, and (iii) the value of the Strobe Index field of the unicast data or command frame whose reception is being acknowledged, as shown in Table 6.2. The Strobe Index field is a new 1-byte header field, which indicates how often ContikiMAC strobed a unicast frame already. Since the Strobe Index field changes in each consecutively strobed unicast frame, CCM has to be rerun over each consecutively strobed unicast frame. To avoid a nonce reuse in this process, SPLO also includes the strobe index in the CCM nonces of unicast frames.

For ensuring the strong freshness and correspondence of acknowledgment frames that are sent in response to unicast data and command frames, SPLO takes three complementary measures. First, senders of such acknowledgment frames implicitly echo the source address, the frame counter, and the strobe index of the unicast data or command frame whose reception is being acknowledged. By implicitly we mean that these values are not actually included in the payload of such acknowledgment frames, but only included in the CCM nonces of such acknowledgment frames. Second, SPLO checks the authenticity of a received unicast data or command frame *before* replying with an acknowledgment frame. Third, SPLO requires a confined reception window for acknowledgment frames. We denote the duration of this reception window by t_a. The effectiveness of these three measures will become apparent in the course of our security analysis of SPLO.

6.4.1.2 Mitigating Collision Attacks

Recall that ContikiMAC's original phase-lock optimization schedules the start of a strobe of unicast frames right before the intended receiver wakes up. For this, the original phase-lock optimization exploits the fact that if an acknowledgment frame is received, the next to last strobed unicast frame must have been transmitted while the receiver woke up. Specifically, like shown in Figure 6.14, let t_0 denote the time when the transmission of the next to last acknowledged frame to a neighbor began, t_1 denote the time when the transmission of the next to last acknowledged frame to a neighbor ended, and t_w be ContikiMAC's wake-up interval. When another strobe of unicast frames to a neighbor shall be sent, the original phase-lock optimization starts strobing at $t_0 + t_w \times n - t_g$ and only strobes once more after $t_1 + t_w \times n + t_g$, where t_g is a configurable guard time and the integer n is chosen so that $t_0 + t_w \times n - t_g$ is in the future.

While ContikiMAC's original phase-lock optimization uses a static guard time, SPLO uses a guard time with a static and a dynamic portion. The static portion, denoted by t_s, should account for inaccuracies. Additionally, t_s must accommodate pulse-delay attacks and therefore $t_s > t_a$. The dynamic portion, denoted by t_u, can be chosen according to the current uncertainty about the wake-up time of the intended receiver. This uncertainty can be upper bounded as follows. Let θ be the frequency tolerance of the employed clocks and let t be the current time. Then, $t_u = (t - t_0) \times (\theta + \theta)$ [75].

The above consideration also implies when the intended receiver must have woken up, namely before $t_1 + t_w \times n + (t_s + t_u)$. Hence, when starting a strobe of unicast frames at $t_0 + t_w \times n - (t_s + t_u)$, only one additional frame must be strobed after $t_1 + t_w \times n + (t_s + t_u)$. Consequently, unlike in ContikiMAC's original phase-lock optimization, SPLO needs no fallback mechanism when unicast transmissions to a receiver tend to fail.

Note that t_0 and t_1 serve as lower and upper bounds of the true wake-up time τ, as shown in Figure 6.14. To allow for a better estimation of τ, SPLO reports back on $\Delta = t_1 - \tau$ to senders by piggybacking Δ on acknowledgment frames, similar to what was suggested in [182]. With the knowledge of Δ, SPLO starts strobing at $\tau + t_w \times n - (t_s + t'_u)$, where $t'_u = (t - \tau) \times (\theta + \theta)$. Analogously, SPLO only strobes one additional frame after $\tau + t_w \times n + (t_s + t'_u)$. This interval is more narrow than $[t_0 + t_w \times n - (t_s + t_u), t_1 + t_w \times n + (t_s + t_u)]$, thereby saving energy during normal operation and under collision attacks.

SPLO's main defense against collision attacks is to send keep-alive messages in the absence of upper-layer traffic. This keeps t'_u low throughout a session and thus all strobes of unicast data and command frames short. Specifically, SPLO modifies AKES to send `UPDATE`s to permanent neighbors whose wake-up time was not updated for a critical period of time T_{lif}. This incurs additional transmissions of `UPDATE`s as compared to not using SPLO since AKES otherwise also prolongs a permanent neighbor's lifetime upon reception of a fresh authentic broadcast or unicast frame, i.e., not only upon reception of a fresh authentic acknowledgment frame. On the other hand, this change leads to a reduction in RAM consumption since it is no longer necessary to store for each permanent neighbor an expiration time. Instead, an `UPDATE` is sent as $t - \tau$ exceeds T_{lif}, where τ is stored anyway. To also keep strobes of `HELLOACK`s and `ACK`s short, SPLO initializes the wake-up times of new neighbors early on, as we detail below. In the exceptional case that $2 \times (t_s + t'_u) \geq t_w$, SPLO starts strobing unicast

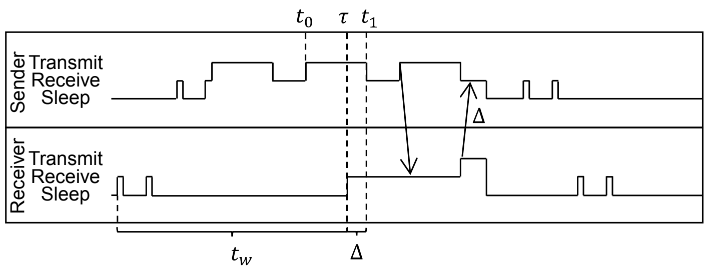


Figure 6.14: Illustration of t_0, t_1, t_w, τ, and Δ

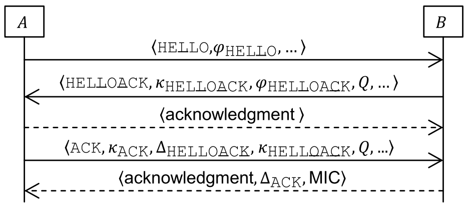

A
B
⟨HELLO, φ_{HELLO}, ...⟩
⟨HELLOACK, κ_{HELLOACK}, $\varphi_{\text{HELLOACK}}$, Q, ...⟩
⟨acknowledgment⟩
⟨ACK, κ_{ACK}, Δ_{HELLOACK}, κ_{HELLOACK}, Q, ...⟩
⟨acknowledgment, Δ_{ACK}, MIC⟩

Figure 6.15: Initialization of wake-up times in parallel to AKES' three-way handshake

frames instantly for at most a whole wake-up interval plus once. Usually, this condition should not take effect since T_{lif} should be chosen so that $2 \times (t_s + t'_u)$ always stays well below t_w.

6.4.1.3 Initializing Wake-Up Times

SPLO initializes wake-up times in parallel to AKES' three-way handshake, as shown in Figure 6.15. Actually, SPLO initializes wake-up times twice during this three-way handshake. The first round of initializations happens when B receives A's HELLO and A receives B's HELLOACK. For this, A embeds the time to its next wake up $\phi_{\texttt{HELLO}}$ in each consecutively strobed HELLO. Likewise, B embeds the time to its next wake up $\phi_{\texttt{HELLOACK}}$ in each consecutively strobed HELLOACK. However, this first round of initializations is vulnerable to attacks. If an attack happens, the firstly initialized wake-up times can be wrong, potentially preventing the successful reception of the subsequent HELLOACK or ACK. The second round of initializations happens when B receives A's ACK and A receives B's acknowledgment frame. For this, A reports back on (i) the cryptographic random number Q that was contained in B's HELLOACK, (ii) the strobe index $\kappa_{\texttt{HELLOACK}}$ that was contained in B's HELLOACK, and (iii) $\Delta_{\texttt{HELLOACK}}$, which is calculated like shown in Figure 6.14 upon receiving B's HELLOACK. Similarly, B piggybacks $\Delta_{\texttt{ACK}}$ on its acknowledgment frame. Acknowledgment frames that are sent in response to ACKs are secured in the same manner as acknowledgment frames that are sent in response to unicast data and command frames.

6.4.2 Security Analysis

In the following, we argue that SPLO makes ContikiMAC resistant to acknowledgment spoofing and pulse-delay attacks, as well as resilient to collision attacks.

6.4.2.1 Resistance to Acknowledgment Spoofing Attacks

To see that acknowledgment spoofing attacks are prevented, let us assume the contrary that a node A receives an authentic acknowledgment frame f in response to a unicast data, unicast command, or `ACK` frame g, though the intended receiver B did not successfully receive g. From the construction of CCM nonces we know that f correctly echoed g's source address, frame counter, and strobe index. Thus, some node must have found a unicast data, unicast command, or `ACK` frame, say h, with g's source address, frame counter, and strobe index authentic. Since f is also authentic, h must have been secured with the same up-to-date session key as f. However, since A's last reboot, the combination of g's frame counter and strobe index is unique, which is why h is g. Yet, any other node than B would have discarded g due to an inauthentic MIC and hence not acknowledged g. Consequently, there is a contradiction.

6.4.2.2 Resistance to Pulse-Delay Attacks

To show that SPLO resists pulse-delay attacks, we will first show the strong freshness of acknowledgment frames:

Let us first assume that a node A receives an acknowledgment frame f delayed by $> t_a$ in response to a unicast data, unicast command, or `ACK` frame g. Note that if f was originally sent in response to a unicast data, unicast command, or `ACK` frame with a different source address, frame counter, or strobe index than g, A will find f inauthentic because all these values are "implicitly echoed" as part of the CCM nonces of acknowledgment frames. Thus, only if f was originally sent in response to a unicast data, unicast command, or `ACK` frame, say h, with g's source address, frame counter, and strobe index, A may find f authentic. Now, there are two cases. First, the sender of h may be A itself. However, h can not be the frame that A just sent because the duration of the reception window for acknowledgment frames is t_a. We can also exclude that A sent h since A's last reboot because A's frame counter increments in each transmission and retransmission. This only retains the possibility that A sent h before the last reboot, in which case A finds f inauthentic since h was then secured using an old session key. Second, the sender of h may be an attacker who impersonated A. In this case, h will not be a fresh authentic frame (unless the session key leaked, but, by definition, pulse-delay attacks are done by external attackers). Thus, no receiver would have replied to h with an acknowledgment frame. Altogether, there is no possibility that A finds f authentic.

Let us now assume that a `HELLOACK` sender B receives a corresponding acknowledgment frame f purportedly from a neighbor A delayed by $> t_a$. Since the duration of the reception window for acknowledgment frames is t_a, f was originally sent in response to a `HELLOACK` that contained a different cryptographic random number Q or a different strobe index $\kappa_{\texttt{HELLOACK}}$. In either case, B will discard A's `ACK`. Thus, B's (second) initialization of A's wake-up time is correct up to t_a. Note that this even holds when acknowledging `HELLOACK`s immediately, without checking their authenticity beforehand. This is crucially important since session keys are not yet in place when receiving a `HELLOACK`.

Consequently, pulse-delay attacks can only be successful if acknowledgment frames are delayed by $\leq t_a$. Yet, since SPLO allows for the undetectable offset t_a when scheduling unicast frames, SPLO resists pulse-delay attacks.

6.4.2.3 Mitigation of Collision Attacks

In the original phase-lock optimization, senders resort to strobe unicast frames for a whole wake-up interval plus once if unicast transmissions to the intended receiver tend to fail, as well as if the wake-up time of the intended receiver was not learned, yet. Thus, the maximum duration of a strobe of unicast frames is $t_w + t_l + t_i + t_l$, where "$+t_l+$" allows for the situation that the transmission of the next to last unicast frame may have begun right before t_w elapsed.

By contrast, in SPLO, the maximum duration of a strobe of unicast data or command frames is $\min\{t_w, 2 \times (t_s + t'_u)\} + t_l + t_i + t_l$. Taking the minimum over $2\times(t_s+t'_u)$ and t_w accounts for the fact that SPLO strobes unicast data and command frames instantly for a full wake-up interval plus once if $2\times(t_s+t'_u)$ becomes greater or equal than t_w. Figure 6.16 shows how this duration compares to that of the original phase-lock optimization. Clearly, if the last acknowledged frame was sent a short time ago, SPLO mitigates collision attacks greatly. For mitigating collision attacks throughout a session, SPLO relies on AKES' `UPDATE`s. In our implementation, we default to send an `UPDATE` to a permanent neighbor if he sent no authentic acknowledgment frame for $T_{\text{lif}} = 5\text{min}$. This reduces the maximum duration of a strobe of unicast data or command frames from 134.58ms to 18.95ms in our implementation. Consequently, collision attacks against unicast data or command frames become much less severe when using SPLO. This even holds true if an attacker jams the `UPDATE`s themselves. This is because if an `UPDATE` is not acknowledged after a configurable number of retransmissions, AKES will delete the seemingly inactive neighbor. Furthermore, no upper-layer traffic can be sent to a deleted neighbor until reestablishing session keys with him. Thus, a victim node may even save energy if an attacker jams `UPDATE`s. However, it remains to be investigated if, by blocking certain links, attackers can also cause an increased energy consumption since this prevents the use of certain routes.

Also, SPLO mitigates collision attacks against `HELLOACK`s and `ACK`s. This is because senders of `HELLOACK`s and `ACK`s already have an estimate of the receiver's wake-up time. Further, since this estimate was only initialized a couple of seconds before sending a `HELLOACK` or `ACK`, strobes of `HELLOACK`s and `ACK`s are very short, thus rendering collision attacks against `HELLOACK`s and `ACK`s benign.

6.4.3 Implementation

A small fix to IEEE 802.15.4 security is to check the authenticity of received unicast data, unicast command, and `ACK` frames before replying with acknowledgment frames. Yet, the implementation of this fix is problematic because ContikiMAC's inter-frame period is pretty short, leaving little time for authenticity checks. We solved this issue by accelerating the transmission of acknowledgment frames through four measures. First, we check the authenticity of a received unicast data, unicast command, or `ACK` frame directly within an interrupt context, rather than waiting for other Protothreads to finish. Second, we leverage the hardware-accelerated CCM implementation of CC2538 SoCs. Third, we prepare acknowledgment frames already during the reception of a unicast frame. Finally, we begin the transmission of acknowledgment frames before the authenticity of a received unicast data, unicast command, or `ACK` frame is checked. This works since we can abort the transmission of an acknowledgment frame early enough if a received unicast data, unicast command, or ACK frame turns out inauthentic. Early enough here means that the abortion happens before the MIC of the acknowledgment frame is being transmitted. These four measures enable us to send acknowledgment frames immediately after receiving a unicast frame.

6.4.4 Evaluation

Using our implementation, we now determine the energy cost of securing acknowledgment frames. Also, we demonstrate that SPLO greatly mitigates collision attacks. Finally, we measure the memory overhead due to SPLO.

6.4.4.1 Energy Cost of Securing Acknowledgment Frames

While SPLO protects ContikiMAC against collision and pulse-delay attacks, SPLO adds the Strobe Index field to each unicast frame and appends a CCM MIC to each acknowledgment frame that is not sent in response to a HELLOACK. To see the increased energy consumption at the sender side, the following experiment was conducted. An OpenMote A sent unicast data frames with 50 bytes of payload to another OpenMote B, which was located 50cm apart from A. During 200 frame transmissions, the energy consumption of A was recorded using the experimental setup described in Section 3.6.2. This experiment was conducted two times, once with SPLO disabled and once with SPLO enabled. Throughout, short addresses, 3-byte OTPs, and Security Level 6 were used.

Figure 6.17 and 6.18a show the results. When not using SPLO, the energy consumption per transmission follows a bimodal distribution. This is because ContikiMAC strobed each frame either two or three times. By contrast, when using SPLO, ContikiMAC always strobed just twice since SPLO predicts wake-up times more precisely than ContikiMAC's original phase-lock optimization. Thus, surprisingly, SPLO may actually save energy at the sender side.

To also measure the energy consumption at the receiver side with vs without SPLO, the above experiment was repeated with the modification that the energy consumption of A instead of B was measured during 200 receptions per run. Throughout, the dozing optimization was used.

The results in Figure 6.18b show that the energy consumption per reception increases when enabling SPLO as expected.

6.4.4.2 Mitigation of Collision Attacks

To put SPLO's mitigation of collision attacks into perspective, the following experiment was conducted. In three successive runs, two OpenMotes ran (i) ContikiMAC without MAC layer security ($t_g = 2$ms), (ii) ContikiMAC with AKES, POTR, and the dozing optimization ($t_g = 2$ms), and (iii) ContikiMAC also with SPLO ($t_s = 0.183$ms, $\theta = 15$ppm, $T_{\text{lif}} = 5$min). In each run, the two OpenMotes sent maximum-length unicast data frames to each other with a random delay of 0 - 4.5min in between. Further, to simulate collision attacks, these unicast data frames were not acknowledged. The energy consumed for transmitting and retransmitting these unicast data frames was traced via Energest [158]. Throughout, ContikiMAC's maximum number of retransmissions was 5, ContikiMAC's wake-up interval t_w was 125ms, and ContikiMAC's inter-frame period t_i was 1.068ms.

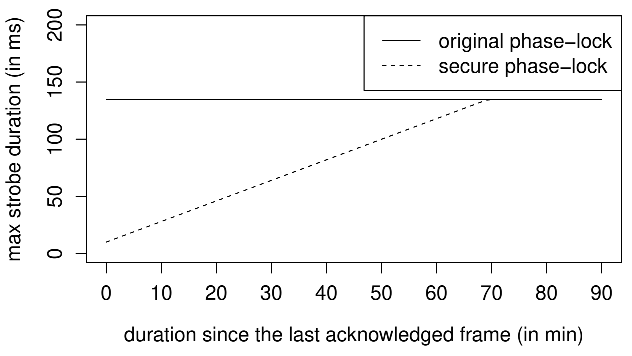


Figure 6.16: Maximum duration of a strobe of unicast frames with vs without SPLO ($t_l = 4.256$ms, $t_i = 1.068$ms, $t_s = 0.183$ms, $\theta = 15$ppm)

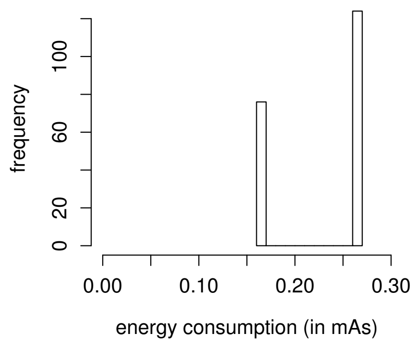


Figure 6.17: Energy consumption per transmission of a unicast data frame with 50 bytes of payload with ContikiMAC's original phase-lock optimization

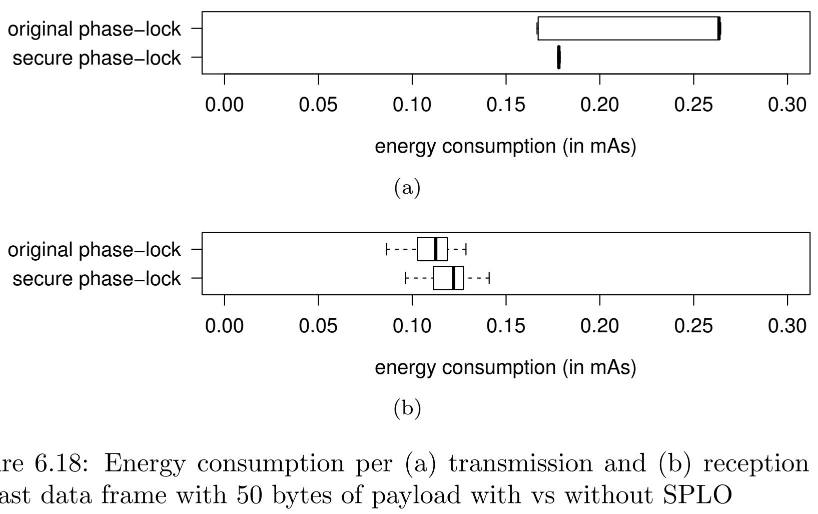


Figure 6.18: Energy consumption per (a) transmission and (b) reception of a unicast data frame with 50 bytes of payload with vs without SPLO

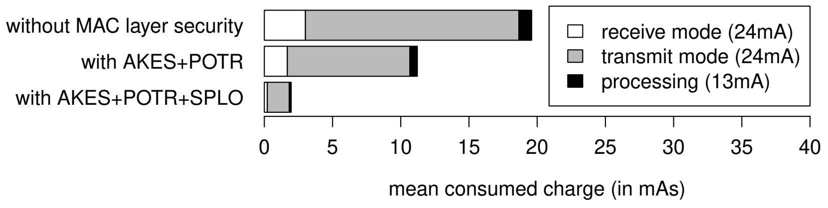


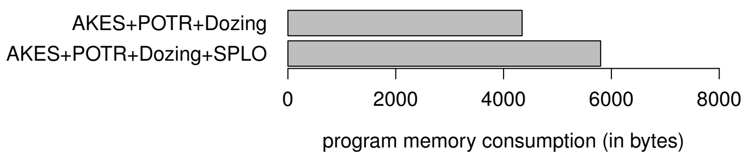
Figure 6.19: Energy consumption per transmission of a maximum-length unicast data frame in the face of collision attacks

As shown in Figure 7.10, when using no MAC layer security, ContikiMAC turned out to expend the most energy. This is because the simulated collision attacks prevent both OpenMotes from learning each other's wake-up time, thus causing ContikiMAC to always strobe for a whole wake-up interval. When enabling POTR, collision attacks become less severe. This positive result comes down to the use of AKES for session key establishment. Specifically, in the course of AKES' three-way handshake, each OpenMote sends one unicast frame to the other OpenMote and hence learns the other OpenMote's wake-up time. Furthermore, AKES sends `UPDATE`s so as to check if a permanent neighbor is still in range. As a side effect, these `UPDATE`s update wake-up times and hence cause ContikiMAC's original phase-lock optimization to strobe only for a limited period of time. Of course, an attacker can also interfere with these `UPDATE`s, but this would usually decrease the energy consumption of victim nodes in the long run. This is because AKES would delete a seemingly inactive neighbor and hence stop upper-layer traffic to the deleted neighbor. Nevertheless, AKES does not reliably mitigate collision attacks because AKES suppresses `UPDATE`s to permanent neighbors that recently sent a fresh authentic broadcast or unicast frame, which, unlike fresh authentic acknowledgment frames, do not update wake-up times as a side effect. Moreover, the effectiveness of these `UPDATE`s is limited since ContikiMAC's original phase-lock optimization relearns the wake-up time of a neighbor if unicast transmissions to the neighbor tend to fail, thus aggravating consecutive collision attacks. By contrast, SPLO never relearns wake-up times and modifies AKES to send `UPDATE`s to permanent neighbors whose wake-up time was not updated for a critical period of time T_{lif}. This explains the much better mitigation of collision attacks when using SPLO.

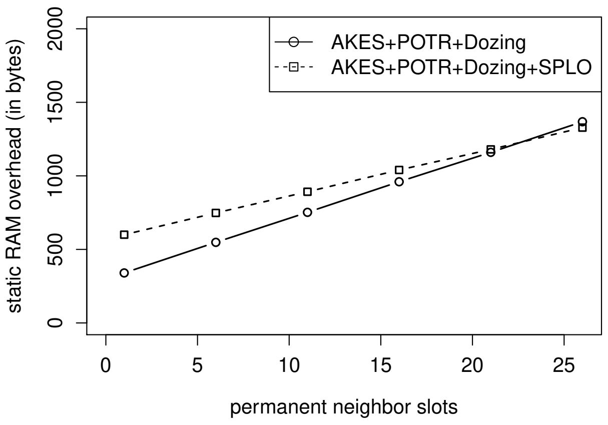
6.4.4.3 Memory Overhead

For measuring the memory overhead due to SPLO, the tool `arm-none-eabi-size` was used. As a baseline for comparison, the memory consumption of using ContikiMAC without MAC layer security was measured. Then, the memory overhead due to enabling AKES, POTR, and the dozing optimization was measured. Next, the memory consumption when enabling also SPLO was measured. These measurements were repeated with different numbers of permanent neighbor slots, whereas the number of tentative neighbor slots was fixed to 5. Throughout, short addresses, Security Level 6, and 3-byte OTPs were used.

The results are shown in Figure 6.20. While the program memory consumption increases due to SPLO, the RAM consumption per permanent neighbor decreases when enabling SPLO. This is because SPLO requires storing only τ



(a)



(b)

Figure 6.20: Memory overhead due to SPLO

per permanent neighbor, whereas ContikiMAC's original phase-lock optimization requires storing t_0 and t_1. Additionally, SPLO frees AKES from storing an expiration time per permanent neighbor.

6.4.5 Discussion

To protect ContikiMAC against acknowledgment spoofing, pulse-delay, and collision attacks, we have proposed replacing ContikiMAC's original phase-lock optimization with SPLO. For protecting against acknowledgment spoofing and pulse-delay attacks, SPLO not only ensures the authenticity of acknowledgment frames, but also their correspondence and strong freshness. Yet, in contrast to Ganeriwal et al.'s SPS, SPLO protects against pulse-delay attacks without any communication overhead [156]. For protecting against collision attacks, SPLO restricts the maximum length of strobes of unicast frames throughout a session. Thus, unlike Ren et al.'s reactive defense against collision attacks [41], SPLO proactively defends against collision attacks.

6.5 The Last Bits Optimization

As became apparent when we compared different addressing modes, the effectiveness of POTR improves when reducing the number of bytes that have to be received prior to validating an OTP. In this regard, the LB optimization may come in handy [147, 178]. The idea of the LB optimization is to just send the L least significant bits of frame counters and to let receivers restore the higher

Table 6.3: CCM Inputs as per the LB Optimization. $C_{A,*}$ is A's broadcast frame counter, $C_{A,B}$ is A's counter for unicast frames to B, ID_B is B's MAC address, ID_* is the broadcast MAC address, and $\kappa <$ 0xFE is the strobe index.

Occasion	CCM key	CCM nonce
OTP of a broadcast data or command frame from A	A's group session key $K_{A,*}$	$ID_*|C_{A,*}|$0xFF
OTP of a unicast data or command frame from A to B	A's group session key $K_{A,*}$	$ID_B|C_{A,B}|$0xFF
OTP of an `ACK` from A to B	Pairwise session key $K'_{A,B}$	$ID_B|C_{A,B}|$0xFF
Authentication and encryption of a `HELLO`, broadcast data, or broadcast command frame from A	A's group session key $K_{A,*}$	$ID_*|C_{A,*}|$0xFE
Authentication and encryption of a unicast data or command frame from A to B	A's group session key $K_{A,*}$	$ID_B\|C_{A,B}\|\kappa$
Authentication and encryption of a `HELLOACK` or `ACK` from A to B	Pairwise session key $K'_{A,B}$	$ID_B\|C_{A,B}\|\kappa$
Authentication of an acknowledgment frame from B to A	Same as the corresponding unicast frame	$ID_A\|C_{A,B}\|\kappa$

order bits by means of their anti-replay data. In this section, we hence integrate the LB optimization into IEEE 802.15.4 security, AKES, POTR, and SPLO.

6.5.1 Design

When using the LB optimization, each node maintains a separate frame counter for outgoing broadcast frames (i.e., `HELLO`, broadcast data, and broadcast command frames) and separate per-neighbor frame counters for outgoing unicast frames (i.e., `HELLOACK`, `ACK`, unicast data, and unicast command frames) [147, 178]. If each node continued to use a single frame counter, the restoration of higher order bits would be unreliable in situations where a node sends many unicast frames to a particular neighbor in a row. By contrast, when using a separate broadcast frame counter, a receiver B can restore a permanent neighbor A's broadcast frame counter unambiguously as long as B missed less than 2^L broadcast frames from A in a row. Likewise, when using per-neighbor unicast frame counters, a permanent neighbor B of A can restore A's unicast frame counter unambiguously as long as B missed less than 2^L unicast frames from A in a row.

Yet, as a result of using separate frame counters, CCM nonces of unicast data and command frames may reoccur together with the same group session key, as becomes apparent in Table 6.3. Therefore, each node A additionally maintains an overall frame counter for outgoing unicast frames, denoted by $C_{A,\circ}$. Its purpose is to initialize new unicast frame counters to a value that is definitely greater than any value that might have been used already.

In order to initially learn the values of a new neighbor's broadcast frame counter, `HELLOACK`s and `ACK`s piggyback this counter in full, as shown in Fig-

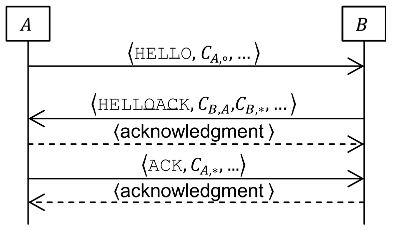

A
B
$\langle$HELLO, $C_{A,\circ}$, ...$\rangle$
$\langle$HELLOACK, $C_{B,A}$,$C_{B,*}$, ...$\rangle$
⟨acknowledgment⟩
$\langle$ACK, $C_{A,*}$, ...$\rangle$
⟨acknowledgment⟩

Figure 6.21: Bootstrapping of the LB optimization in parallel to AKES' three-way handshake

ure 6.21. HELLOs additionally carry the sender's overall unicast frame counter. This is because the OTP of ACKs is derived from a unicast frame counter whose higher order bits would be unknown at time of reception otherwise. By contrast, HELLOACKs contain the sender's unicast frame counter in full, i.e., without compressing the higher order bits, as indicated in Figure 6.21.

While using separate frame counters reduces the risk of getting out of sync, a node A may still lose track of the unicast or broadcast frame counter of a permanent neighbor B. In effect, A no longer gets fresh authentic unicast or broadcast frames from B. To recover from this situation, we apply two tweaks to AKES' detection of inactive permanent. First, we piggyback the sender's broadcast frame counter in full on AKES' UPDATEs, as well as the acknowledgment frames that are sent in response to UPDATEs. This enables the receiver of such a frame to resynchronize with the sender's broadcast frame counter. Second, we modify AKES and SPLO to not only send UPDATEs to permanent neighbors whose wake-up time was not updated for T_{lif}, but also when no fresh authentic broadcast frame, UPDATE, or UPDATE acknowledgment frame was received for T_{lif}. Thus, when a node A loses track of the broadcast frame counter of a permanent neighbor B, A will eventually send an UPDATE to B. Furthermore, the corresponding acknowledgment frame contains B's broadcast frame in full so that A can resynchronize with B's broadcast frame counter. On the other hand, when a node A loses track of the unicast frame counter of a permanent neighbor B, B will eventually delete A from its list of permanent neighbors. This is because A rejects B's unicast data and command frames and therefore sends no acknowledgment frames in response. After B deleted A from its list of permanent neighbors, A no longer receives fresh authentic acknowledgment frames from B. Thus, A will delete B from its list of permanent neighbors, as well. Then, A and B no longer silently discard each other's HELLOs, eventually triggering a reestablishment of session keys between them. While reestablishing session keys, A and B relearn each other's frame counters like explained above.

6.5.2 Evaluation

In the following, we first measure the acceleration of the rejection speed of POTR when using the LB optimization, as well as the resultant energy savings in the face of broadcast and unicast attacks with data frames. Next, we demonstrate that when using the LB optimization, the energy consumption of transmitting and receiving unicast data frames decreases slightly. Finally, we determine the memory overhead due to the LB optimization.

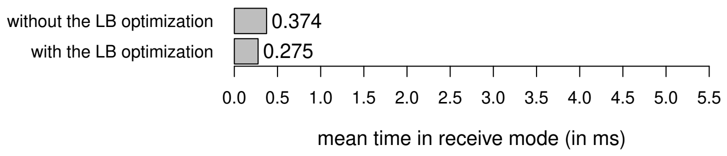


Figure 6.22: POTR's rejection speed with vs without the LB optimization

6.5.2.1 Rejection Speed

To measure the rejection speed, an OpenMote A injected maximum-length unicast data frames, each of which contained the source address of a permanent neighbor of another OpenMote B, as well as an invalid OTP. The OpenMote B, which had POTR, SPLO, and the LB optimization enabled, received 100 of these frames and logged the time between `SFD` interrupts and the moment when the frame reception aborts, which is signaled by `RXABO` interrupts. This experiment was then repeated without the LB optimization. Throughout, short addresses, 3-byte OTPs, and $L = 8$ were used.

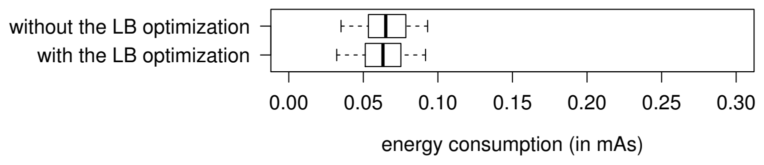
The results are shown in Figure 6.22. As expected, when using the LB optimization, the rejection speed is faster by $\approx \frac{24\text{bit}}{250\frac{\text{kbit}}{\text{s}}} = 96\mu\text{s}$.

6.5.2.2 Energy Savings

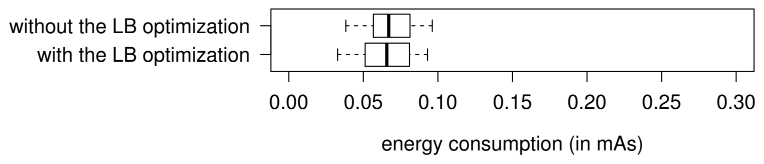
Next, to compare the resultant energy savings under broadcast and unicast attacks with data frames, an OpenMote A was placed 50cm apart from another OpenMote B. Furthermore, the OpenMote A sent maximum-length broadcast data frames, which are rejected by B due to invalid OTPs. During 200 of these broadcast attacks, the energy consumption of B was traced using the experimental setup described in Section 3.6.2. This experiment was first performed without the LB optimization and subsequently with the LB optimization. Then, these two runs were repeated with maximum-length unicast data frames instead of maximum-length broadcast data frames. Throughout, short addresses, 3-byte OTPs, $L = 8$, and the dozing optimization were used.

The results are shown in Figure 6.23. The energy savings when using the LB optimization are marginal, which can be explained as follows. Recall that OpenMotes draw between 20 and 24mA in receive mode, depending on the strength of the input signal. Furthermore, as explained above, the time in receive mode is only reduced by 96μs when using the LB optimization. Hence, the reduction in energy consumption should theoretically range between 0.00192 and 0.00230mAs. In Figure 6.23a and 6.23b, the mean reduction in energy consumption is 0.00282mAs and 0.00289mAs, respectively.

Another beneficial effect of the LB optimization is that the energy consumption for transmitting and receiving legitimate data frames decreases because of the decreased security-related per-frame overhead. To show this effect, an OpenMote A sent unicast data frames with 50 bytes of payload to another OpenMote B, which was located 50cm apart from A. During 200 frame receptions and 200 frame transmissions, the energy consumption of B and A was recorded using the experimental setup described in Section 3.6.2, respectively. This experiment was conducted two times, once with the LB optimization disabled and once with the LB optimization enabled. Throughout, short addresses, 3-byte OTPs, Security

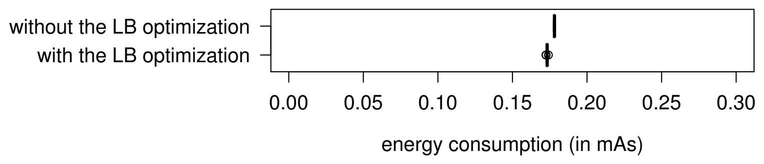


(a)

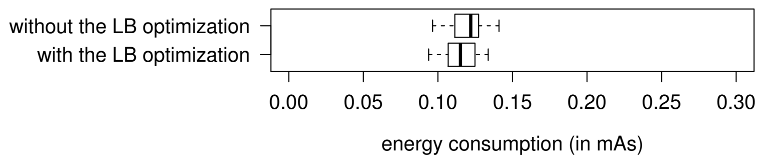


(b)

Figure 6.23: Energy consumption per wake up under (a) unicast and (b) broadcast attacks with vs without the LB optimization



(a)



(b)

Figure 6.24: Energy consumption per (a) transmission and (b) reception of a unicast data frame with 50 bytes of payload with vs without the LB optimization

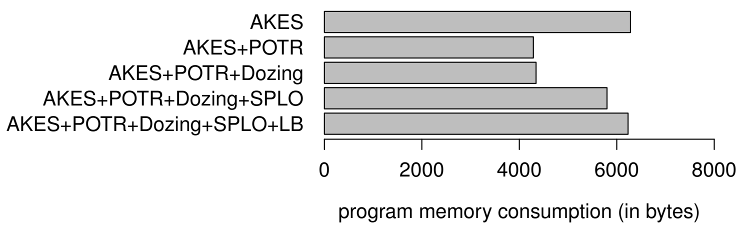
Level 6, $L = 8$, SPLO, and the dozing optimization were used.

The results are shown in Figure 6.24. Expectably, transmissions and receptions consume slightly less energy when the LB optimization is enabled.

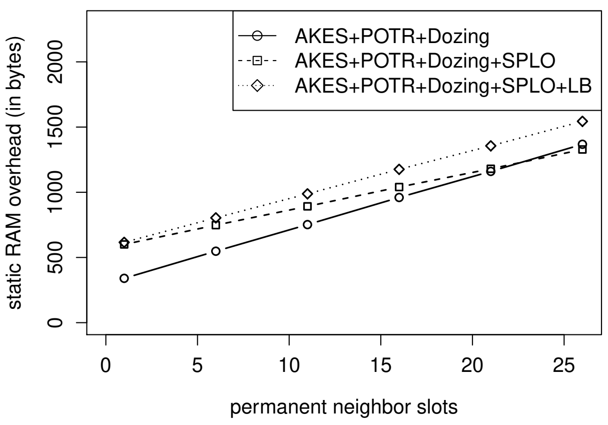
6.5.2.3 Memory Overhead

The memory overhead due to the LB optimization was determined as follows. At first, as a baseline for comparison, the memory consumption of using ContikiMAC without MAC layer security was measured using the tool `arm-none-eabi-size`. Then, the memory overhead due to enabling AKES, POTR, the dozing optimization, SPLO, and the LB optimization was determined. This measurement was repeated with different numbers of permanent neighbor slots, whereas the number tentative neighbor slots was fixed to 5. Throughout, short addresses, Security Level 6, and 3-byte OTPs were used.

The results are shown in Figure 6.25 alongside previously obtained results. While the program memory overhead due to the LB optimization is small, the



(a)



(b)

Figure 6.25: Memory overhead due to the LB optimization

RAM overhead increases considerably when enabling the LB optimization. This is because the LB optimization necessitates storing two frame counters per permanent neighbor, rather than just one. Moreover, whereas SPLO just requires storing the last known wake-up time of a permanent neighbor for scheduling UPDATEs, the LB optimization additionally necessitates storing per permanent neighbor the time when its broadcast frame counter should be updated next.

6.5.3 Discussion

To improve the effectiveness of POTR, we have integrated the LB optimization into IEEE 802.15.4 security, AKES, POTR, and SPLO. Indeed, we have demonstrated that the LB optimization accelerates POTR's rejection speed, leading to a slight reduction of the incurred energy consumption of broadcast, unicast, and droplet attacks with ACK, data, and command frames. Additionally, the LB optimization reduces the security-related per-frame overhead, which saves energy during receptions and transmissions of legitimate data frames. Also, thanks to reducing the security-related per-frame overhead, fragmentation at upper layers may happen less often. On the other hand, the LB optimization requires an energy- and time-consuming resynchronization when losing track of a permanent neighbor's unicast or broadcast frame counter. Moreover, the RAM overhead of the LB optimization is considerable.

6.6 ILOS: Intra-Layer Optimization for IEEE 802.15.4 Security

ILOS overcomes the following three remaining issues with our denial-of-sleep-protected version of ContikiMAC:

- If an attacker guesses an OTP right, the corresponding anti-replay data gets corrupted, potentially ensuing a reestablishment of session keys.
- Strong freshness is only ensured in the case of acknowledgment frames. For all other frames, only sequential freshness is achieved.
- While the LB optimization reduces the security-related per-frame overhead and accelerates POTR's rejection speed, it comes at the cost of an increased RAM consumption and occasional energy- and time-consuming resynchronizations. Moreover, these resynchronizations are not triggered immediately upon desynchronization, but only delayed.

6.6.1 Design

The main idea of ILOS is to replace frame counters with what we term *wake-up counters*. This requires changes to (i) CCM nonces, (ii) AKES, and (iii) replay protection. In the following, we will explain each of these changes.

6.6.1.1 Notations

Let A and B be adjacent nodes. We denote by:

- t_w ContikiMAC's wake-up interval, as shown in Figure 6.15.
- ω_A the wake-up counter of A. A increments ω_A at the rate of t_w in lockstep with ContikiMAC's two regular CCAs. If A skips over doing two CCAs due to sending at that time, A must increment ω_A anyway.
- $\tau_{A,B}$ what A stores as the last wake-up time of B. SPLO initializes $\tau_{A,B}$ in parallel to establishing group session keys and updates $\tau_{A,B}$ upon reception of an authentic acknowledgment frame from B.
- $\omega_{A,B}$ the wake-up counter of B at time $\tau_{A,B}$. Likewise, ILOS initializes $\omega_{A,B}$ in the course of establishing session keys and updates $\tau_{A,B}$ upon reception of an authentic acknowledgment frame from B.
- ID_A A's MAC address.
- ID_* the broadcast MAC address.
- $K_{A,*}$ A's group session key.
- $\kappa <$ 0xFE a strobe index.

Table 6.4: CCM Inputs as per ILOS

Occasion	CCM key	CCM nonce
OTP of a broadcast data or command frame from A	A's group session key $K_{A,*}$	$ID_*\|\omega_A+1\|0\text{xFF}$
OTP of a unicast data or command frame from A to B	B's group session key $K_{B,*}$	$ID_A\|\omega_B\|0\text{xFF}$
OTP of an `ACK` from A to B	pairwise session key $K'_{A,B}$	$ID_B\|\omega_B\|0\text{xFF}$
Authentication and encryption of a `HELLO`, broadcast data, or broadcast command frame from A	A's group session key $K_{A,*}$	$ID_*\|\omega_A+1\|0\text{xFE}$
Authentication and encryption of a unicast data or command frame from A to B	B's group session key $K_{B,*}$	$ID_A\|\omega_B\|\kappa$
Authentication and encryption of a `HELLOACK` or `ACK` from A to B	pairwise session key $K'_{A,B}$	$ID_A\|\omega_B\|\kappa$
Authentication of an acknowledgment frame from B to A	same as the corresponding unicast frame	$\neg ID_A\|\omega_B\|\kappa$

6.6.1.2 Adapting CCM Nonces

In order for CCM nonces to be secure, they must never reoccur in conjunction with the same key. ILOS achieves this via two complementary techniques. On the one hand, ILOS uses a common base format and differing values in the final field to avoid collisions among CCM nonces that are used in different occasions. On the other hand, ILOS uses wake-up counters and MAC addresses to avoid collisions among CCM nonces that are used in same occasions. Concretely, ILOS generates CCM nonces like shown in Table 6.4.

Unicast frames: As for authenticating and encrypting a unicast frame from a node A to a node B, ILOS generates the CCM nonce by concatenating ID_A, ω_B, and κ, where ω_B is B's wake-up counter at time of reception. Thus, A has to predict ω_B. As A is aware of $\tau_{A,B}$ and $\omega_{A,B}$, A can predict ω_B as $\omega_{A,B} + \left\lceil \frac{t_\text{sched} - \tau_{A,B}}{t_w} \right\rceil$, where t_sched is the time when SPLO schedules the transmission of the unicast frame. This prediction is correct if SPLO is configured to keep the uncertainty about the wake-up time of a permanent neighbor below t_w.

Unicast OTPs: As for generating the OTPs of unicast data, unicast command, and `ACK` frames, the CCM nonce is generated in the same manner like for authenticating and encrypting unicast frames, except that the final field is set to 0xFF.

Acknowledgment frames: As for authenticating acknowledgment frames, the same CCM nonce is used like for authenticating and encrypting unicast frames, except that the MAC address is bitwise negated.

Broadcast frames: As for authenticating and encrypting broadcast frames from a node A, ILOS generates the CCM nonce by concatenating ID_*,

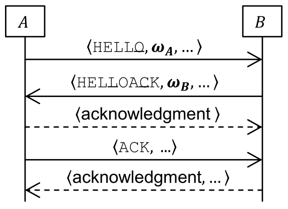


Figure 6.26: Initialization of wake-up counters in parallel to AKES' three-way handshake

$\omega_A + 1$, and 0xFE. Here, ω_A is A's wake-up counter as A begins to strobe. To aid receivers in restoring $\omega_A + 1$, $\omega_A + 1$ needs to be even and A must begin to strobe at $t - \frac{t_w}{2}$, where t is when A increments ω_A next. Thus, A may need to defer the transmission of the broadcast frame until both conditions are met. Yet, this way, a receiver B can restore $\omega_A + 1$ by rounding $\omega_{B,A} + \frac{t_{\text{awoke}} - \tau_{B,A}}{t_w}$ to the next even value, where t_{awoke} is when B awoke for doing ContikiMAC's two CCAs that led to receiving the broadcast frame. This restoration of $\omega_A + 1$ is correct (i) if SPLO keeps the uncertainty about the wake-up time of A below t_w and (ii) if B wakes up during the interval $[t - \frac{t_w}{2}, t + \frac{t_w}{2}]$.

Broadcast OTPs: As for generating the OTPs of broadcast data and command frames, the CCM nonce is generated in the same manner like for authenticating and encrypting broadcast frames, except that the final field is set to 0xFF.

6.6.1.3 Adapting AKES

Let A and B be adjacent nodes. To initialize $\omega_{A,B}$ and $\omega_{B,A}$ while A and B establish session keys, ILOS adds additional data to HELLOs and HELLOACKs, as shown in Figure 6.26. Recall that SPLO already piggybacks the time to the next wake up on HELLOs and HELLOACKs, as shown in Figure 6.15. Hence, by piggybacking the sender's current wake-up counter on HELLOs and HELLOACKs in addition, A and B can now initialize $\omega_{A,B}$ and $\omega_{B,A}$, respectively. Subsequently, if SPLO updates $\tau_{A,B}$ or $\tau_{B,A}$, ILOS updates $\omega_{A,B}$ or $\omega_{B,A}$ to the correctly predicted wake-up counter at that time, respectively.

6.6.1.4 Adapting Replay Protection

ILOS provides replay protection as follows.

Unicast frames: As for unicast frames, replay protection comes almost as a side effect of generating CCM nonces like ILOS does. This is because a receiver B increments ω_B when waking up. Thus, if B receives a replayed unicast frame during a later wake up, B will assume a CCM nonce that differs from the CCM nonce that was used to secure the replayed unicast frame. Hence, B will reject the replayed unicast frame due to an invalid OTP or an inauthentic MIC. However, a subtlety is that the sender A of a unicast frame may miss an acknowledgment frame. In this case, A may retransmit and hence, the receiver B, may accept the same frame twice.

Hence, it is still necessary to add sequence numbers to unicast data and command frames. Although duplicated unicast data and command frames could already be discarded during reception, they must be fully received and acknowledged to avoid self-imposed collision attacks.

Acknowledgment frames: As for acknowledgment frames, replay protection is provided by retaining SPLO's mechanisms for this.

Broadcast frames: As for broadcast frames, replay protection does not come automatically. Suppose a node B receives a broadcast frame from a sender A and, during the next wake up of B, an attacker replays that same broadcast frame. In this case, B may assume the same CCM nonce, thus causing B to consider both the OTP and the MIC of the replayed broadcast frame valid. Only if there is one wake up in between, B will definitely assume a different CCM nonce and hence reject the replayed broadcast frame due to an invalid OTP or an inauthentic MIC. This also holds true if B updates $\tau_{B,A}$ and $\omega_{B,A}$ in between since, in this case, $\omega_{B,A}$ is raised, which causes B to restore a different CCM nonce, too. Altogether, ILOS merely needs to take care of not accepting a broadcast frame if, during the last wake up, a broadcast frame from the same sender was accepted already. ILOS does so already during reception.

6.6.2 Security Analysis

Below, we (i) show that ILOS' CCM nonces do not reoccur in conjunction with the same CCM key and (ii) argue that ILOS achieves strong freshness.

6.6.2.1 Uniqueness of CCM Nonces

Since ILOS' CCM nonces have a common format, we can separate three cases:

0xFF: As for OTPs, the final field of CCM nonces is set to 0xFF. Furthermore, in the case of broadcast OTPs, the MAC address field of CCM nonces is set to ID_*, which distinguishes their CCM nonces from the ones of unicast OTPs. Hence, it remains to ensure that the value of the wake-up counter field of the CCM nonce of a broadcast or unicast OTP does not coincide with the one of other broadcast or unicast OTPs in conjunction with the same CCM key, respectively. As for CCM nonces of broadcast OTPs, this holds true since ILOS centers strobes of consecutive broadcast frames around distinct even wake ups. Likewise, since SPLO centers strobes of unicast frames around different wake ups of a receiver, CCM nonces of unicast OTPs do not reoccur in conjunction with the same CCM key.

0xFE: As for the authentication and encryption of broadcast frames, the final field of CCM nonces is set to 0xFE. Thus, such CCM nonces differ from CCM nonces that are used in other occasions. Further, since ILOS centers strobes of consecutive broadcast frames around distinct even wake ups, such CCM nonces do not reoccur in conjunction with the same CCM key.

Else: As for the authentication and encryption of unicast frames, as well as the authentication of acknowledgment frames, the final field of CCM nonces is set to a strobe index. Let us first ensure that when authenticating and encrypting unicast frames, the corresponding CCM nonces do not collide among themselves in conjunction with the same CCM key. Note that such CCM nonces contain the sender's MAC address, the receiver's wake-up counter at time of reception, and the current strobe index. Thus, such CCM nonces can only collide if they originate from the same sender. Furthermore, if the CCM key is unchanged, a different wake-up counter or strobe index will be used. Consequently, the CCM nonces used for authenticating and encrypting unicast frames can not collide among themselves in conjunction with the same CCM key. Likewise, let us also ensure that the CCM nonces used for authenticating acknowledgment frames do not collide among themselves in conjunction with the same CCM key. This can be seen by observing that a node can acknowledge at most one unicast frame per wake up. Thus, it remains to ensure that the CCM nonces used for authenticating and encrypting unicast frames, and the CCM nonces used for authenticating acknowledgment frames do not collide among each other in conjunction with the same CCM key. In the case of group session keys, this is achieved through bitwise negating the MAC address field. In the case of pairwise session keys, additionally observe that authenticated acknowledgment frames are only sent in one direction, thereby excluding that the pathological case of $\neg ID_A = ID_B$ can ensue a nonce reuse.

6.6.2.2 Freshness Guarantees

In the case of unicast frames, ILOS achieves strong freshness up to t_w. This is because, if a unicast frame is delayed by $\geq t_w$, receivers will use a different CCM nonce to verify its MIC, which results in the rejection of the frame. Further, duplicate unicast frames are either detected by AKES or through sequence numbers. In the case of acknowledgment frames, strong freshness is ensured by retaining SPLO's mechanisms for this. In the case of broadcast frames, ILOS achieves strong freshness up to $2t_w$ since receivers will definitely assume a different CCM nonce when a broadcast frame is delayed by $\geq 2t_w$. Also, ILOS takes care of not accepting the same broadcast frame twice.

6.6.3 Implementation

Our implementation of ILOS advances our implementation of ContikiMAC, AKES, POTR, the dozing optimization, and SPLO. We preserved the software architecture and inserted the changes of ILOS surrounded by conditional preprocessor directives. This enables us to switch between frame counters and wake-up counters at compilation time. Within our conditional code, we only let information flow downwards, following the terminology introduced in [183]. That is, upper layers retrieve additional information, such as the current wake-up counter, from lower layers. Nevertheless, ILOS is not a real cross-layer optimization because ILOS only affects the MAC layer.

6.6.4 Evaluation

Using our implementation, we now (i) quantify the reduction of the security-related per-frame overhead thanks to ILOS, (ii) demonstrate the resulting reduction in energy consumption during transmissions and receptions of unicast

Table 6.5: Security-Related Per-Frame Overhead

Frame Format	Overhead (in bytes)
IEEE 802.15.4	$[5, 14] + \frac{m}{8}$
IEEE 802.15.4+TSCH	$[1, 10] + \frac{m}{8}$
IEEE 802.15.4+AKES	$5 + \frac{m}{8}$
POTR+AKES	$4 + \frac{l+m}{8}$
POTR+AKES+SPLO	$[4, 5] + \frac{l+m}{8}$
POTR+AKES+SPLO+LB	$[1, 2] + \frac{l+m}{8}$
POTR+AKES+SPLO+ILOS	$[0, 1] + \frac{l+m}{8}$

data frames, (iii) demonstrate that ILOS accelerates the on-the-fly rejection of unwanted unicast data frames, and (iv) give insight to ILOS' memory overhead.

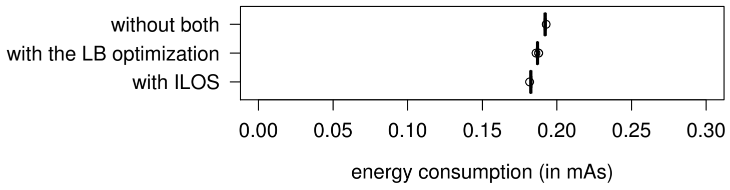
6.6.4.1 Security-Related Per-Frame Overhead

Table 6.5 compares the security-related per-frame overhead of various frame formats. As per IEEE 802.15.4, if not using TSCH, a secured IEEE 802.15.4 frame has a 1-byte Security Control field, a 4-byte Frame Counter field, an optional Key Identifier field of up to 9 bytes, as well as an m-bit CCM MIC. In TSCH networks, the Frame Counter field is obsolete. Using AKES obviates the Key Identifier field. On the other hand, while POTR removes the Security Control field, POTR adds the l-bit OTP field to data, command, and `ACK` frames. Moreover, SPLO requires adding the 1-byte Strobe Index field to unicast frames. The LB optimization can reduce the length of POTR's Frame Counter field from 4 bytes to 1 byte. ILOS further reduces the security-related per-frame by dispensing with frame counters entirely. Additionally, when using ILOS, it becomes reasonable to use shorter OTPs, e.g., 2-byte OTPs. This is because if an attacker guesses an OTP right, no anti-replay data gets corrupted.

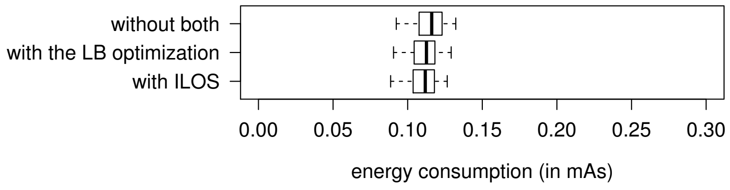
6.6.4.2 Energy Efficiency

To demonstrate that ILOS reduces the energy consumption of sending unicast data frames, the following experiment was conducted. An OpenMote A sent unicast data frames with 50 bytes of payload to another OpenMote B, which was placed 50cm apart from A. While A strobed and subsequently received B's acknowledgment frame, the current draw of A was measured like described in Section 3.6.2. This was repeated using three different configurations, namely (i) with neither the LB optimization nor ILOS, (ii) with the LB optimization, and (iii) with ILOS. For each of the three configurations, 200 samples were obtained. Throughout, short addresses, $L = 8$, Security Level 6, SPLO, POTR, and the dozing optimization were used. Furthermore, the length of OTPs l was set to 24 when ILOS was disabled and to 16 otherwise.

Figure 6.27a shows the results. Expectably, the most energy-consuming configuration is to use neither the LB optimization nor ILOS. This is because, in this configuration, frame counters are transmitted uncompressed. Enabling the LB optimization saves energy since this reduces the Frame Counter field by 3 bytes. Another 2 bytes can be saved by enabling ILOS instead, yielding a slightly lower energy consumption compared to using the LB optimization.



(a)



(b)

Figure 6.27: Energy consumption per (a) transmission and (b) reception of a unicast data frame with 50 bytes of payload in different configurations

To also demonstrate that ILOS reduces the energy consumption of receiving legitimate unicast data frames, the above experiment was repeated with two differences. First, B's energy consumption while receiving A's unicast data frames and acknowledging them was measured. Second, to avoid bias, A sent at randomized times.

Figure 6.27b shows the results. This time, the variation in the data is much higher because the energy consumption per frame reception highly depends on how long a receiver waits until the next unicast frame is being strobed. Apart from that, the results are similar. While ILOS constitutes the most energy-efficient configuration, using neither the LB optimization nor ILOS constitutes the least energy-efficient configuration.

The above results apply to unicast command and data frames alike since they are treated equally. Broadcast receptions should also consume less energy when using the LB optimization or ILOS. Broadcast transmissions, by contrast, will not become more energy efficient on average since ContikiMAC strobes broadcast frames for a whole wake-up interval anyway. Additionally, we note that thanks to reducing the security-related per-frame overhead, fragmentation at upper layers may happen less often when using ILOS, then saving further energy.

6.6.4.3 Rejection Speed

To show the acceleration of POTR's rejection speed, the following experiment was conducted. An OpenMote A sent a maximum-length unicast data frame with an invalid OTP to another OpenMote B, which stored A as a permanent neighbor. Upon reception, B stopped the time between detecting the frame's SHR and the rejection of the frame, which is signaled by `RXABO` interrupts. This was repeated using three different configurations, namely (i) with the LB optimization, as well as ILOS disabled, (ii) with the LB optimization enabled, and (iii) with ILOS enabled. For each of the three configurations, 100 samples were obtained. Throughout, short addresses and $L = 8$ were used. Furthermore, the length of OTPs l was set to 24 when ILOS was disabled and to 16 otherwise.

Figure 6.28 shows that the LB optimization accelerates POTR's rejection speed noticeably. ILOS accelerates POTR's rejection speed even more since the validation of OTPs begins even earlier. In effect, ILOS further reduces the time spent in receive mode under broadcast, unicast, and droplet attacks.

6.6.4.4 Memory Overhead

As a baseline for comparison, the memory consumption was first measured when using ContikiMAC without MAC layer security using the tool `arm-none-eabi--size`. Then, the memory overhead of enabling AKES, POTR, the dozing optimization, SPLO, as well as ILOS was measured. This procedure was repeated with different numbers of permanent neighbor slots, whereas the number of tentative neighbor slots was fixed to 5. Throughout, short addresses, Security Level 6, and 2-byte OTPs were used.

The results are shown in Figure 6.29 alongside previous ones. The program memory overhead of ILOS turned out to be small. Also, ILOS saves RAM thanks to dispensing with frame counters. This is in contrast to the LB optimization.

6.6.5 Discussion

A core concept of IEEE 802.15.4 security is the use of frame counters for both nonce generation and replay protection. While being functional, frame counters (i) cause an increased energy consumption as they incur a per-frame overhead of 4 bytes, (ii) only provide sequential freshness, and (iii) suffer from the issue that if an OTP is guessed right, anti-replay data gets corrupted. The LB optimization does reduce the per-frame overhead of frame counters, yet at the cost of an increased RAM consumption and occasional energy- and time-consuming resynchronizations. We have proposed ILOS, which outperforms the LB optimization in terms of security-related per-frame overhead, energy efficiency, rejection speed, as well as RAM footprint. Furthermore, in contrast to the LB optimization, ILOS requires no resynchronizations. Beyond that, thanks to wake-up counters, ILOS achieves strong freshness and avoids the corruption of anti-replay data in case of a successful guessing attack. The only drawback of ILOS is that ILOS intertwines ContikiMAC and MAC layer security. From a software engineer's perspective, we would actually like to decouple these aspects so that we can change one without affecting the other.

6.7 Summary

Related work proposed the embedding of OTPs in synchronization or frame headers to defend against broadcast, unicast, and droplet attacks. Thus far, it was, however, unclear how to (i) practically implement OTPs, (ii) avoid the overhead of establishing extra keys for deriving OTPs, (iii) prevent broadcast, unicast, and droplet attacks while establishing these extra keys, and (iv) mitigate successful guessing attacks without the overhead of network-wide time synchronization. In this chapter, we have proposed POTR and ILOS, which address these open issues. POTR only depends on features of commodity IEEE 802.15.4 transceivers, uses the same keys as IEEE 802.15.4 security, and integrates with AKES so as to mitigate broadcast, unicast, and droplet attacks

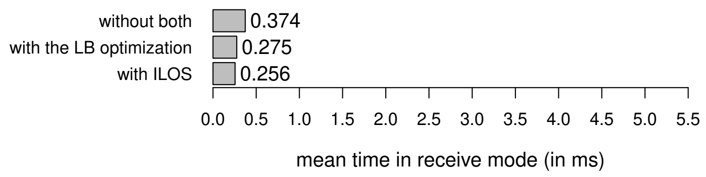
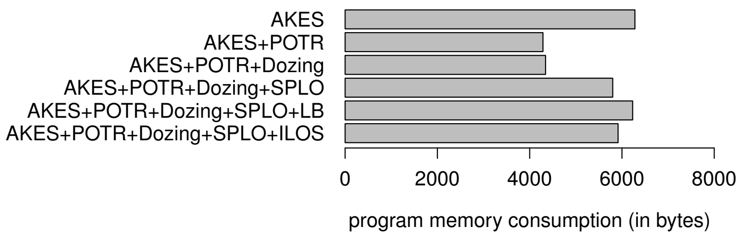
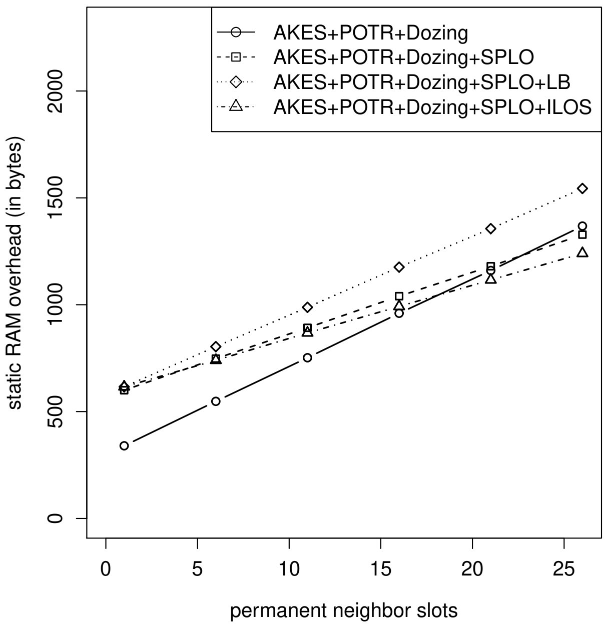


Figure 6.28: POTR's rejection speed in different configurations



(a)



(b)

Figure 6.29: Memory overhead due to ILOS

while establishing those keys. ILOS, on the other hand, adapts POTR to derive OTPs from wake-up counters. In doing so, ILOS renders successful guessing attacks against POTR benign without network-wide time synchronization.

In this chapter, we have also considered two optimizations to further mitigate external reception-oriented denial-of-sleep attacks against ContikiMAC. The dozing optimization, on the one hand, reduces ContikiMAC's time in receive mode while searching for the silence period between two successively strobed frames. Indeed, the dozing optimization has turned out to further mitigate broadcast, unicast, and droplet attacks compared to using POTR alone, as well as to mitigate jamming attacks. Beyond that, the dozing optimization reduces the energy consumption for receiving legitimate frames, too. The LB optimization, on the other hand, compresses frame counters and thereby accelerates the speed at which POTR rejects frames with invalid OTPs. However, the LB optimization is superseded by ILOS since ILOS outperforms the LB optimization in rejection speed, as well as in all other considered regards. Beyond that, ILOS achieves strong freshness, thereby overcoming a basic limitation of the frame counter-based replay protection of IEEE 802.15.4 security.

Furthermore, we have proposed SPLO. SPLO constitutes the first defense against acknowledgment spoofing attacks, the first lightweight defense against pulse-delay attacks, as well as the first effective defense against collision attacks. For preventing acknowledgment spoofing and pulse-delay attacks, SPLO not only ensures the authenticity of acknowledgment frames, but also their strong freshness and correspondence. For mitigating collision attacks, SPLO restricts the maximum length of strobes of unicast frames throughout a session.

Finally, for preventing interruption attacks, we have proposed to not perform CCAs between successively strobed frames at all.

In sum, in this chapter, we have proposed a denial-of-sleep-protected version ContikiMAC, which prevents, or at least greatly mitigates, all external denial-of-sleep attacks against ContikiMAC we identified in Chapter 5.

However, our denial-of-sleep-protected version of ContikiMAC inherits three basic limitations from ContikiMAC. First, ContikiMAC depends on CCAs to detect transmissions, thus rendering ContikiMAC difficult to configure and susceptible to interference [81, 167, 168]. Second, ContikiMAC requires padding short frames with extra bytes, as otherwise they may fall between ContikiMAC's two regular CCAs [60]. Third, ContikiMAC is prone to duplicate receptions of broadcast frames since senders need to strobe broadcast frames not only for an entire wake-up interval, but once more to cover corner cases [184].

Actually, all these three limitations of ContikiMAC can be overcome, too. An approach to automatically tune CCA thresholds to trade off false wake ups against undetected transmissions was proposed by Sha et al. [81]. An approach to avoid padding bytes is to do three CCAs instead of two CCAs [185]. Finally, an approach to avoid duplicate receptions of broadcast frames is to embed strobe indices also in broadcast frames, which would enable receivers to skip over a wake up after receiving a broadcast frame with a low strobe index. On the other hand, none of these limitations arises when using CSL.

Chapter 7

Denial-of-Sleep Defenses for CSL

This chapter presents our denial-of-sleep-protected version of CSL, which constitutes the final design of our denial-of-sleep-resilient MAC protocol. Our version of CSL resists, or at least greatly mitigates, all denial-of-sleep attacks against CSL we found. Furthermore, our version of CSL integrates a channel hopping extension, as well as CSL's throughput optimization of bursting pending frames.

7.1 Overview

To protect CSL against denial-of-sleep attacks, we propose CSL-enabled and partially enhanced versions of POTR, SPLO, and ILOS, entitled POTR++, SPLO++, ILOS++, respectively:

POTR++: The basic design of POTR and POTR++ is similar. Both adopt the emergent pattern of OTPs for defending against broadcast, unicast, and droplet attacks. Also, both integrate with AKES so as to resolve the conflict that frames exchanged during session key establishment have to be protected against broadcast, unicast, and droplet attacks without session keys in place. On the other hand, a fundamental difference between POTR and POTR++ is that POTR++ exclusively derives OTPs from pairwise session keys in order to provide a basis for pinpointing and evicting chatty nodes. Another basic difference between POTR and POTR++ is that POTR++ supports bursts (introduced in Section 2.2.2).

SPLO++: SPLO and SPLO++ use similar means for preventing acknowledgment spoofing and pulse-delay attacks, as well as for mitigating collision attacks. To prevent acknowledgment spoofing and pulse-delay attacks, both ensure not only the authenticity and sequential freshness of acknowledgment frames, but also their correspondence and strong freshness. To mitigate collision attacks, both keep the maximum uncertainty about a permanent neighbor's wake-up time below a user-defined threshold t_t by sending `UPDATE`s in the absence of upper-layer traffic. However, a crucial difference between the two is that SPLO++ leverages the predictability of clock drifts. In effect, this lowers the energy cost of sending `UPDATE`s and other unicast frames, as well as further mitigates collision attacks.

ILOS++: Like ILOS, ILOS++ has the responsibilities to generate and restore CCM nonces, as well as to initialize and maintain all relevant data.

POTR++, SPLO++, and ILOS++ are built around our concept of wake-up counters, which replace frame counters. As their name suggests, wake-up counters are incremented in lockstep with CSL's wake ups, even if CSL has to skip over a wake up due to sending at that time. Furthermore, as mentioned above, SPLO++ maintains precise estimates of the wake-up time of each neighbor. ILOS++ takes advantage of these estimates for predicting and restoring a neighbor's current wake-up counter. That is, unlike frame counters, wake-up counters need not be sent along with secured frames, which reduces the security-related per-frame overhead. More importantly, if an authenticated frame is delayed by a critical period of time, receivers will end up generating a different CCM nonce. Consequently, receivers consider delayed authenticated frames inauthentic, thereby achieving strong freshness. This is in contrast to a frame counter-based replay protection, which only provides sequential freshness.

7.2 Design

In the following, we detail the design of POTR++, SPLO++, and ILOS++.

7.2.1 Notations

For reference, we summarize our notations below. We denote by:

- ID_A the short or extended MAC address of A. In line with AKES, we assume that either short or extended addresses are used within an IEEE 802.15.4 network and that short addresses are unique within an IEEE 802.15.4 network.
- $K'_{A,B}$ the pairwise session key between A and B.
- l the length of an OTP in bits.
- t_a the duration of the reception window for acknowledgment frames.
- t_t the maximum allowable uncertainty about a permanent neighbor's wake-up time.
- t_w CSL's wake-up interval.
- α a field within the CCM nonces generated by ILOS++.
- $\eta_{A,B}$ an estimate of the clock drift of A against B.
- θ the frequency tolerance of the employed clocks.
- θ_η the frequency tolerance of estimates of clock drifts.
- t_m a parameter of SPLO++.

- λ a burst index. The first payload frame after a wake-up sequence always has burst index $\lambda = 0$, a payload frame that follows right after has burst index $\lambda = 1$, and so on.
- $\tau_{A,B}$ B's last wake-up time that is known to A.
- φ a *CSL phase*. Unless otherwise mentioned, measured as the time between when the SHR of the frame containing φ was transmitted and the sender's next wake-up time.
- ω_A the wake-up counter of A. A increments ω_A in lockstep with CSL's periodic wake ups, even if CSL skips over a wake up due to sending at that time.
- $\omega_{A,B}$ the wake-up counter of B at time $\tau_{A,B}$.

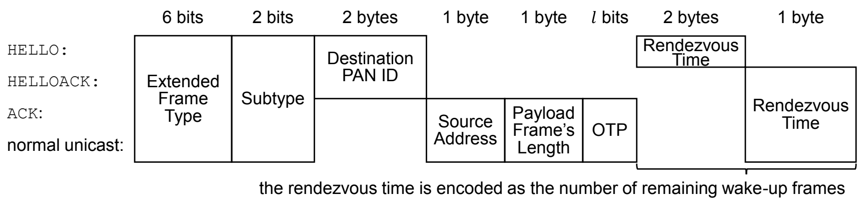
7.2.2 POTR++: CSL-Enabled and Enhanced POTR

The design of POTR++ is divided into two parts, namely a special frame format and corresponding procedures for rejecting unwanted frames during reception.

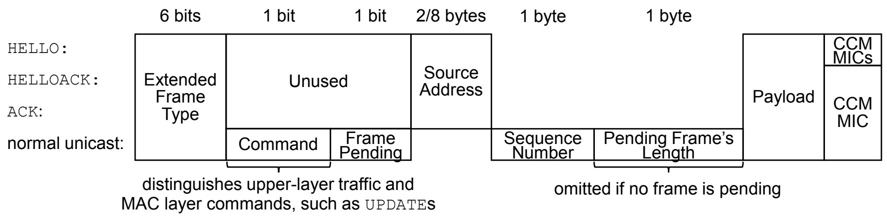
7.2.2.1 Frame Format

When we defined the frame format of POTR++, a first design decision was where to embed OTPs. A first option is to leave wake-up frames unsecured and to embed OTPs in payload frames. But, a drawback of this first option is that receivers of unwanted payload frames will always need to receive both an entire wake-up frame plus the beginning of the payload frame before they can reject it. A second option is to embed OTPs in wake-up frames along with all necessary information for deciding whether to receive the payload frame. Yet, while this second option accelerates the rejection of unwanted payload frames, it will extend wake-up frames and hence necessitate prolonging the time in receive mode during CSL's periodic wake ups. Thus, there is a trade-off between rejection speed and the duration of periodic wake ups. POTR++ goes for faster rejection speed, as shown in Figure 7.1. Therein, l denotes the length of an OTP in bits. The choice of l presents a trade-off between per-frame overhead and resistance to guessing attacks. Besides, λ denotes the *burst index* - the first payload frame after a wake-up sequence always has burst index $\lambda = 0$, a payload frame that follows right after has burst index $\lambda = 1$, and so on.

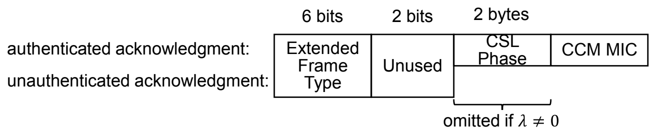
Note that a format for broadcast frames is missing in Figure 7.1. This is because POTR++ requires sending broadcast frames as normal unicast frames to each permanent neighbor, one after another, following the `unicast_strategy` introduced earlier. Though we showed the `unicast_strategy` to be energy consuming, this way of transmitting broadcast frames may actually save energy when it comes to implementing channel hopping, as we will demonstrate in Section 7.5.1. Three further advantages of sending broadcast frames as normal unicast frames are (i) that, in this way, pairwise session keys can serve to secure broadcast transmissions without difficulty (ii) that broadcast transmissions update stored wake-up times as a side effect, thereby lowering the amount of `UPDATE`s, and (iii) that failed broadcast transmissions are recognized and retried, thus improving the reliability of broadcast transmissions.



(a)



(b)


6 bits
2 bits
2 bytes
authenticated acknowledgment:
unauthenticated acknowledgment:
Extended Frame Type
Unused
CSL Phase
CCM MIC
omitted if $\lambda \neq 0$

(c)

Figure 7.1: POTR++'s format of (a) wake-up, (b) payload, and (c) acknowledgment frames. Sticking with the standardized format of these frames would require transmitting many more fields that are functionally unnecessary, only delaying on-the-fly rejection and increasing energy consumption. Therefore, POTR++ departs from the standardized format of CSL frames by leveraging the IEEE 802.15.4-compliant customization mechanism of extended frame types.

HELLOs, however, are still to be broadcasted as usual since not-yet-discovered neighbors should receive HELLOs, as well. This leaves the problem of how to authenticate HELLOs with pairwise session keys. To this end, POTR++ also follows the unicast_strategy by including multiple CCM MICs in a HELLO, one for each permanent neighbor, and generates them using the respective pairwise session key. The ordering of the CCM MICs corresponds to the ordering of the sender's neighbor list. Thus, for extracting its MIC, the receiver of a HELLO from a permanent neighbor needs to know its index within the sender's neighbor list. These indices are piggybacked on HELLOACKs and ACKs. POTR++ also uses these indices as 1-byte source addresses in some types of wake-up frames.

7.2.2.2 On-the-Fly Rejection

Depending on which type of frame is being received, POTR++ performs different checks, which we detail below. Throughout, if any check fails, POTR++ disables the receive mode immediately, thereby stopping further energy loss.

Wake-up frames: When receiving a wake-up frame, POTR++ first checks that its length complies with POTR++'s frame format. Then, POTR++

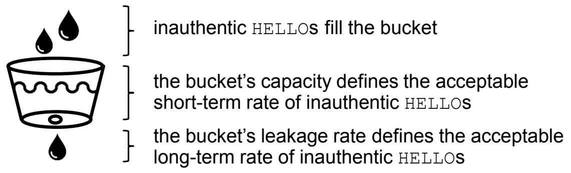

inauthentic HELLOs fill the bucket
the bucket's capacity defines the acceptable short-term rate of inauthentic HELLOs
the bucket's leakage rate defines the acceptable long-term rate of inauthentic HELLOs

Figure 7.2: Mitigation of broadcast and droplet attacks with `HELLO`s

proceeds to perform special checks, depending on which type of payload frame follows as announced through the Subtype field. If a `HELLO` follows, POTR++ ensures (i) that the `HELLO` is not destined to a co-located IEEE 802.15.4 network by inspecting the destination PAN ID contained in the wake-up frame and (ii) that receiving the `HELLO` does not result in exceeding a maximum rate of incoming `HELLO`s. POTR++ implements this rate limitation via an LBC, as shown in Figure 7.2. Yet, to avoid penalizing authentic `HELLO`s, the LBC is decremented again if a `HELLO` turns out authentic. This LBC-based approach to mitigating broadcast and droplet attacks with inauthentic `HELLO`s nicely allows the short-term rate of incoming inauthentic `HELLO`s to overshoot, while still enforcing a maximum long-term rate of incoming inauthentic `HELLO`s. If a `HELLOACK` follows, POTR++ validates (i) that AKES recently sent a `HELLO` at all, (ii) that AKES can reply with an `ACK`, which may not be true since AKES restricts itself to sending a maximum rate of `ACK`s, (iii) that the `HELLOACK` is not destined to a co-located IEEE 802.15.4 network by inspecting the destination PAN ID contained in the wake-up frame, and (iv) that receiving the `HELLOACK` does not ensue exceeding a maximum rate of incoming `HELLOACK`s. Again, POTR++ implements this rate limitation via an LBC and decrements that LBC if a `HELLOACK` turns out authentic. If an `ACK` or normal unicast frame follows, POTR++ ascertains that the OTP matches the expected one. Concretely, if A sends an `ACK` or normal unicast frame to B, the OTP is expected to match the l-bit truncated CCM MIC over the frame length of the `ACK` or normal unicast frame, respectively. Furthermore, this CCM MIC is to be generated using the pairwise session key $K'_{A,B}$ between A and B as CCM key, and the CCM nonce shown in Table 7.1. Finally, if all type-specific checks passed, POTR++ also validates that the wake-up frame's rendezvous time is not unreasonably late.

Payload frames: When it comes to receiving a payload frame, POTR++ performs the following checks. In the case of a `HELLO` from a non-permanent neighbor, POTR++ ensures that AKES can reply with a `HELLOACK` at all. This may not be the case because AKES restricts itself to sending a maximum rate of `HELLOACK`s and because AKES may run out of memory. Note, however, that AKES is interested in receiving `HELLO`s from permanent neighbors since AKES uses such `HELLO`s to trigger a reestablishment of session keys after reboots, as well as to suppress redundant `HELLO`s. In the case of a `HELLOACK`, POTR++ just validates that its length complies with POTR++'s frame format. In the case of an `ACK` or normal unicast frame, POTR++ ascertains that its length matches the one that was announced in the wake-up frame. Furthermore, if more normal unicast

Table 7.1: Modified CCM Inputs

Occasion	CCM Key	CCM Nonce
OTP of a wake-up frame from A to B	$K'_{A,B}$	$ID_A \| \alpha \| \lambda \| \omega_B$, where ID_A is A's MAC address, $\alpha = 0$, $\lambda = 0$, and ω_B is B's wake-up counter at time of receiving the wake-up frame
Authentication of a `HELLO` from A	see Section 7.2.2.1	$ID_A \| \alpha \| \lambda \| \omega_A$, where ID_A is A's MAC address, $\alpha = 1$, $\lambda = 0$, and ω_A is A's wake-up counter when sending the `HELLO` - ω_A is also contained in the `HELLO`, but a permanent neighbor B of A restores ω_A like specified in Section 7.2.4
Authentication and encryption of a unicast frame from A to B	$K'_{A,B}$	$ID_A \| \alpha \| \lambda \| \omega_B$, where ID_A is A's MAC address, $\alpha = 2$, λ is the burst index, and ω_B is B's wake-up counter at time of receiving the wake-up frame that leads to receiving the unicast frame
Authentication of an acknowledgment frame from A to B	$K'_{A,B}$	$ID_A \| \alpha \| \lambda \| \omega_A$, where ID_A is A's MAC address, $\alpha = 3$, λ is the burst index of the acknowledged unicast frame, and ω_A is A's wake-up counter at time of receiving the wake-up frame that led to receiving the acknowledged `ACK` or normal unicast frame

frames burst in, POTR++ always makes sure that their length matches the length announced in the previous one.

Acknowledgment frames: When it comes to receiving an acknowledgment frame, POTR++ only ensures that its length complies with POTR++'s frame format.

7.2.3 SPLO++: CSL-Enabled and Enhanced SPLO

Now, we specify how SPLO++ secures acknowledgment frames and how SPLO++ limits the maximum duration of wake-up sequences. Later, in Section 7.2.4.2, we also specify how SPLO++ initializes wake-up times.

7.2.3.1 Securing Acknowledgment Frames

Once wake-up times are initialized, each node A stores for every permanent neighbor B the last known wake-up time $\tau_{A,B}$ of B. A exclusively updates $\tau_{A,B}$ when receiving an authentic acknowledgment frame from B. In contrast to the standardized version of CSL, SPLO++ does not only ensure the authenticity and sequential freshness of acknowledgment frames, but also their correspondence and strong freshness. For ensuring the correspondence and strong freshness of authenticated acknowledgment frames, SPLO++ takes three complementary measures. First, senders of authenticated acknowledgment frames implicitly echo the wake-up counter at time of receiving the wake-up frame that led to receiving the unicast frame that is being acknowledged, as well as the burst index of the unicast frame that is being acknowledged. By implicitly, we mean that this data is not explicitly sent along with authenticated acknowledgment frames, but included in the CCM nonces of authenticated acknowledgment frames, as shown in Table 7.1. Second, unlike specified by CSL, all nodes must check the authenticity of normal unicast frames before replying with authenticated acknowledgment frames. Third, SPLO++ requires a confined reception window for acknowledgment frames of duration t_a. The effectiveness of these three measures will become apparent in the course of our security analysis.

7.2.3.2 Limiting the Maximum Duration of Wake-Up Sequences

To keep the maximum uncertainty about a permanent neighbor's wake-up time below a user-defined threshold t_t, SPLO++ modifies AKES to schedule `UPDATE`s like follows. Initially, a node A has no estimate of its clock drift $\eta_{A,B}$ against a new permanent neighbor B, which is why A can only assume A's and B's clocks to diverge by 2θ at most, where θ is the frequency tolerance of the employed clocks. Hence, A has to send B an `UPDATE` $\frac{t_t}{2\theta}$ seconds after discovering B, unless A sends another normal unicast frame to B beforehand anyway. For example, if $t_t = 2\text{ms}$ and $\theta = 15\text{ppm}$, A sends the first `UPDATE` to B after 66s of inactivity, minus a bit to account for retransmissions. However, every time when A updates $\tau_{A,B}$ to $\tau'_{A,B}$, A can compute its current clock drift against B as $\eta_{A,B} = \frac{\tau'_{A,B} - \tau_{A,B}}{(\omega'_{A,B} - \omega_{A,B}) \times t_w}$, where t_w is the interval of CSL's periodic wake ups, $\omega_{A,B}$ is the wake-up counter of B at time $\tau_{A,B}$, and $\omega'_{A,B}$ is the wake-up counter of B at time $\tau'_{A,B}$. From then on, A compensates for $\eta_{A,B}$ and assumes $\eta_{A,B}$ to be correct up to $\theta_\eta < \theta$. Thus, A can not only reduce the length of wake-up sequences, but also raise the duration after which A sends B `UPDATE`s, yet not too much as $\eta_{A,B}$ may deviate from the real clock drift too much otherwise [91, 92]. We denote the maximum duration until sending an `UPDATE` by t_m. Also note that, in order to get a precise estimate of $\eta_{A,B}$, the difference $\tau'_{A,B} - \tau_{A,B}$ should be sufficiently long [91, 92], which may necessitate delaying the initialization and updating of $\eta_{A,B}$ until two sufficiently distant samples are in place.

7.2.4 ILOS++: CSL-Enabled ILOS

ILOS++ has the responsibilities to generate and restore CCM nonces, as well as to keep track of the wake-up counters of neighboring nodes.

7.2.4.1 Generation and Restoration of CCM Nonces

ILOS++ generates CCM nonces like shown in Table 7.1. The security of this construction of CCM nonces will be proven in Section 7.3, but, basically, rests on two complementary measures. On the one hand, ILOS++ avoids collisions among CCM nonces of different frame types by using a common base format and the distinguishing field α. On the other hand, ILOS++ avoids collisions among CCM nonces of the same frame type by including the burst index, as well as either the sender's or the receiver's wake-up counter.

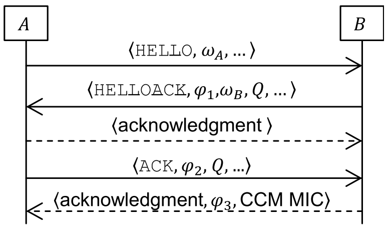

A
B
⟨HELLO, ω_A, ...⟩
⟨HELLOACK, φ_1, ω_B, Q, ...⟩
⟨acknowledgment⟩
⟨ACK, φ_2, Q, ...⟩
⟨acknowledgment, φ_3, CCM MIC⟩

Figure 7.3: Initialization of wake-up counters and times in parallel to AKES' three-way handshake by SPLO++ and ILOS++

As CCM nonces either contain the receiver's or the sender's wake-up counter, it is either necessary that the sender predicts the receiver's wake-up counter or that the receiver restores the sender's wake-up counter. Predicting a wake-up counter is necessary to generate the CCM nonces of unicast frames and OTPs. To do so, a sender A predicts a receiver B's wake-up counter at time of receiving A's wake-up frame as $\omega_{A,B} + \frac{t-\tau_{A,B}}{t_w \eta_{A,B}}$, where $\tau_{A,B}$ is A's last known wake-up time of B, $\omega_{A,B}$ is B's wake-up counter at time $\tau_{A,B}$, t is the expected wake-up time of B around which A arranges its wake-up sequence, and $\eta_{A,B} = 1$ if not yet initialized. Restoring a wake-up counter is only necessary to restore the CCM nonce of a HELLO from a permanent neighbor. To aid receivers of HELLOs in doing so, senders have to time HELLOs so that the end of the transmission of a HELLO's SHR falls right in the middle between two consecutive wake ups of the sender. This enables a permanent neighbor B of A to restore A's wake-up counter at time of transmitting a HELLO by rounding $\omega_{B,A} + \frac{t-\tau_{B,A}}{t_w}\eta_{B,A} - \frac{1}{2}$, where $\tau_{B,A}$ is B's last known wake-up time of A, $\omega_{B,A}$ is A's wake-up counter at time $\tau_{B,A}$, t is when the HELLO's SHR arrived, and $\eta_{B,A} = 1$ if not yet initialized.

For implementing the prediction and restoration of wake-up counters, ILOS++ requires every node A to store each permanent neighbor B's wake-up counter $\omega_{A,B}$ at time $\tau_{A,B}$. In this regard, note that when SPLO++ updates $\tau_{A,B}$ to $\tau'_{A,B}$, A already knows B's wake-up counter $\omega'_{A,B}$ at time $\tau'_{A,B}$ because A correctly predicted B's wake-up counter $\omega'_{A,B}$ at time $\tau'_{A,B}$. Thus, whenever SPLO++ updates $\tau_{A,B}$ to $\tau'_{A,B}$, ILOS++ updates $\omega_{A,B}$ to the correctly predicted wake-up counter $\omega'_{A,B}$ of B at time $\tau'_{A,B}$ without further calculations.

7.2.4.2 Initialization of Wake-up Counters and Times

An overlapping part of SPLO++ and ILOS++ is the initialization of wake-up times and counters. This initialization actually happens twice in parallel to AKES' three-way handshake.

1. The first initialization happens upon reception of a HELLO or HELLOACK. Specifically, as shown in Figure 7.3, the HELLO carries A's current wake-up counter ω_A. Though attackers can delay HELLOs, B takes this information for granted and initializes $\omega_{B,A}$ to ω_A and $\tau_{B,A}$ to the time when the HELLO's SHR arrived minus $\frac{t_w}{2}$ (since a HELLO's SHR is transmitted in the middle between two consecutive wake ups of the sender). As B generates the CCM nonce of the HELLOACK, B uses this data for the first time so as to predict A's wake-up counter. If this prediction is incorrect, B will not

receive an ACK in response and eventually delete A from its list of tentative neighbors. Otherwise, it may still be the case that $\tau_{B,A}$ is slightly offset due to a pulse-delay attack. Hence, B corrects $\tau_{B,A}$ upon receiving A's ACK like is explained shortly. Similarly, B's HELLOACK carries B's CSL phase φ_1 and B's current wake-up counter ω_B. A also takes this data for granted and uses it to initialize $\tau_{A,B}$ and $\omega_{A,B}$. When generating the CCM nonce of the ACK, A uses $\tau_{A,B}$ and $\omega_{A,B}$ for the first time so as to predict B's wake-up counter. Only if this prediction is correct, A will get an authentic acknowledgment frame in response and, else, will delete B from its list of permanent neighbors. Besides, B's HELLOACK carries a cryptographic random number Q that is newly generated for each transmission and retransmission. After receiving B's HELLOACK, A acknowledges with an unauthenticated acknowledgment frame. As opposed to authenticated acknowledgment frames, unauthenticated acknowledgment frames can be sent immediately without executing AKES' checks beforehand.

2. The second "corrective" initialization happens upon reception of an ACK or the corresponding authenticated acknowledgment frame. Specifically, A's ACK contains φ_2 and Q, where φ_2 is A's CSL phase when receiving the HELLOACK's SHR. Upon receiving A's ACK, B ensures that Q is unmodified and uses φ_2 to correct its value of $\tau_{B,A}$. Also, B updates $\omega_{B,A}$ to A's correctly predicted wake-up counter at time of receiving B's HELLOACK. Lastly, B replies with an authenticated acknowledgment frame containing B's CSL phase φ_3. If A finds B's authenticated acknowledgment frame authentic, A uses φ_3 to correct its value of $\tau_{A,B}$ and updates $\omega_{A,B}$ to B's correctly predicted wake-up counter at time of receiving A's ACK.

7.3 Security Analysis

In this section, we analyze the security of POTR++, SPLO++, and ILOS++. We start with showing that basic wireless security is preserved if using our wake-up counter-based CCM nonces and replay protection. Then, we argue that POTR++ and SPLO++ counter all denial-of-sleep attacks against CSL.

7.3.1 Basic Wireless Security

7.3.1.1 Uniqueness of CCM nonces

In order for CCM nonces to be secure, they must never reoccur in conjunction with the same key. Owing to using pairwise session keys and since all CCM nonces include the sender's MAC address and the distinguishing field α, it suffices to ensure that no CCM nonce coincides with other CCM nonces from a particular sender to a particular receiver for each value of α within a session.

HELLOs: As for $\alpha = 1$, the inclusion of the sender's wake-up counter ensures that such CCM nonces differ from previous ones of a particular sender within a session since CSL needs to transmit wake-up frames for an entire wake-up interval before sending another HELLO, thus causing a sender's wake-up counter to increment in the meantime.

Unicast frames: As for $\alpha = 2$, CCM nonces include the burst index and the receiver's wake-up counter at time of receiving the wake-up frame that leads to receiving the unicast frame. The inclusion of the burst index distinguishes CCM nonces of normal unicast frames within a burst. Thus, it remains to ensure that the included wake-up counter differs in "spaced" transmissions of unicast frames from a particular sender to a particular receiver within a session. As SPLO++ keeps the uncertainty about a receiver's wake-up time below $t_t < t_w$ within a session, CSL will arrange the wake-up sequences of spaced transmissions of unicast frames around distinct wake ups within a session. Hence, the included wake-up counter indeed differs in spaced transmissions of unicast frames within a session.

Authenticated acknowledgment frames: As for $\alpha = 3$, CCM nonces include the burst index of the acknowledged unicast frame and the sender's wake-up counter at time of receiving the wake-up frame that led to receiving the acknowledged unicast frame. Since the burst index differs when acknowledging individual normal unicast frames of a burst, it remains to ensure that the included wake-up counter differs when acknowledging spaced transmissions of unicast frames within a session. This holds true since CSL can receive at most one wake-up frame per wake up and because the wake-up counter at time of a wake up is unique within a session.

7.3.1.2 Freshness Guarantees

Recall that strong freshness is provided if a receiver can not only ensure that a frame was never accepted before, but also that it was sent within a limited time span prior to its reception [54]. Together, SPLO++ and ILOS++ do achieve strong freshness, as we argue below.

HELLOs: Suppose B receives a `HELLO` that originates from a permanent neighbor A delayed by $\geq t_w$. Then, B will round A's wake-up counter to a different value than was used by A to generate the CCM nonce of the `HELLO`. As a result, B will consider the `HELLO` inauthentic. The same reasoning applies to why B never considers the same `HELLO` authentic twice.

Unicast frames: Suppose B receives a unicast frame f that originates from a neighboring node A delayed by $\geq t_w$. Then, B will derive f's CCM nonce from a different wake-up counter than was predicted by A. Hence, B will consider f's CCM MIC inauthentic and discard f. However, a subtlety is that the sender of a unicast frame may miss the corresponding acknowledgment frame and decide to retransmit. As a result, a receiver may accept the same unicast frame twice. To this end, we add sequence numbers to normal unicast frames, as shown in Figure 7.1b. These sequence numbers are maintained on a per-neighbor basis and are incremented each time a normal unicast frame is transmitted to a permanent neighbor, yet not incremented when retransmitting the last normal unicast frame. Consequently, receivers can identify and filter out duplicated normal unicast frames. On the other hand, adding sequence numbers to `ACK`s and `HELLOACK`s is unnecessary since AKES considers duplicated `ACK`s and `HELLOACK`s replayed anyway.

Authenticated acknowledgment frames: Suppose B receives an authenticated acknowledgment frame f delayed by $> t_a$. Since B only accepts acknowledgment frames for t_a, f must originally have been sent in response to a different normal unicast frame or `ACK`. There are two cases how this may happen. First, f was originally sent in response to a different normal unicast frame of the same burst. In this case, B will include a different burst index in the CCM nonce. Second, f was originally sent in response to a previous spaced transmission of a normal unicast frame or `ACK`. In this case, B will include a different wake-up counter in the CCM nonce. In both cases, B ends up considering the CCM MIC of f inauthentic and rejects f.

Unauthenticated acknowledgment frames: The strong freshness of unauthenticated acknowledgment frames, which are only sent in response to `HELLOACK`s, is ensured only after the corresponding `ACK` comes in. Suppose B receives an unauthenticated acknowledgment frame f from A delayed by $> t_a$. Since B only accepts acknowledgment frames for t_a, f must originally have been sent in response to a different `HELLOACK`. However, the originally acknowledged `HELLOACK` contained a different cryptographic random number Q, thus causing B to reject A's `ACK` and AKES' three-way handshake to remain incomplete.

7.3.2 Denial-of-Sleep Resilience

7.3.2.1 Broadcast, Unicast, and Droplet Attacks

Now, we argue that POTR++ mitigates broadcast, unicast, and droplet attacks, no matter what the frame type of the injected or replayed frame or droplet is.

HELLOs and HELLOACKs: POTR++ mitigates broadcast, unicast, and droplet attacks with `HELLO`s and `HELLOACK`s by only letting such types of frames pass at a maximum rate, if they seem valid and are of interest at all. To exemplify the effectiveness of this defense, let us compare the time that CSL needs to spend in receive mode during its periodic wake ups if $t_w = 125\text{ms}$ and $l = 16$ anyway with the additional worst-case time in receive mode when accepting `HELLO`s at a rate of $\frac{1}{15}\text{Hz}$. Assuming the data rate of the 2.4-GHz O-QPSK PHY of $250\frac{\text{kbit}}{\text{s}}$, the time in receive mode per wake up is $\frac{(6+6+5)\text{byte}\times 8\frac{\text{bit}}{\text{byte}}}{250\frac{\text{kbit}}{\text{s}}} = 0.544\text{ms}$ minimum. Here, 6+ accounts for the length of the O-QPSK PHY header, +6+ is the maximum length of a wake-up frame as per POTR++'s frame format, and +5 allows for detecting a wake-up frame's SHR when the beginning of the previous wake-up frame was just missed. Thus, on a percentage basis, the base time in receive mode is $100\% \times \frac{0.544\text{ms}}{t_w} = 0.43\%$. For comparison, receiving one maximum-length `HELLO` takes $\frac{(6+127)\text{byte}\times 8\frac{\text{bit}}{\text{byte}}}{250\frac{\text{kbit}}{\text{s}}} = 4.256\text{ms}$. Moreover, we have to account for the unfortunate case that the SHR of the wake-up frame of a `HELLO` may be detected right at the end of a periodic wake up. In such cases, receivers need to stay in receive mode slightly longer to see whether the SHR is the beginning of a wake-up frame. In the case of a `HELLO`, receivers may need to receive up to 6 more bytes as per Figure 7.1a, which translates into an additional time in receive mode of up to 0.192ms. Altogether, the additional worst-case time in receive mode is 0.192ms + 4.256ms. Yet, on a percentage basis, receiving one maximum-length `HELLO` at the rate of $\frac{1}{15}$Hz only adds 0.03% to the base time in receive mode at most. Thus, when configuring POTR++ to only let `HELLO`s and `HELLOACK`s pass at a moderate rate, POTR++ makes CSL practically resistant to broadcast, unicast, and droplet attacks with `HELLO`s and `HELLOACK`s.

ACKs and normal unicast frames: One method for launching unicast and droplet attacks with `ACK`s or normal unicast frames is to (i) inject or replay a wake-up sequence and (ii) inject or replay an `ACK`, normal unicast frame, or droplet. However, for this method to work out, attackers need to come up with valid OTPs as invalid OTPs trigger an on-the-fly rejection. Crafting a valid OTP seems impossible since unicast and droplet attacks are launched by external attackers, which have no access to cryptographic keys. Also, guessing an OTP, or replaying an OTP that is older than $\geq t_w$, is pointless because such an OTP will turn out invalid with probability $1-2^{-l}$. On the other hand, a victim may consider a replayed OTP that is younger than $< t_w$ valid, but this means that the victim node missed the original transmission. Hence, in terms of energy consumption, there will be not much of a difference compared to receiving the original transmission in the first place. This even holds true if an attacker tampers with the length of an `ACK`, normal unicast frame, or droplet. This is because the length of a payload frame is announced in the preceding wake-up frames and because changing the announced length would invalidate the replayed OTP.

Another method for launching unicast and droplet attacks with `ACK`s or normal unicast frames is to "jump on" unicast transmissions. That is, after a victim node received a wake-up frame or a normal unicast frame with the Frame Pending flag set, an attacker can inject or replay an `ACK`, normal unicast frame, or droplet. But, the only repercussion is that the victim node receives the attacker's frame or droplet instead of the legitimate frame. This does not result in an increased energy consumption because, if the attacker's frame or droplet has a different length than was previously announced, the attacker's frame or droplet will be rejected on the fly.

Acknowledgment frames: Since POTR++ validates the length of acknowledgment frames during reception, over-long acknowledgment frames do not cause receivers to stay longer in receive mode.

7.3.2.2 Collision Attacks

Now, we argue that SPLO++ mitigates collision attacks.

HELLOACKs and ACKs: Owing to initializing wake-up times early on, the sender of a `HELLOACK` or `ACK` already has an estimate of its new neighbor's wake-up time and can hence shorten wake-up sequences before `HELLOACK`s and `ACK`s. Furthermore, since this estimate is initialized only a couple of seconds before transmitting a `HELLOACK` or `ACK`, the uncertainty about a new neighbor's wake-up time is tiny and hence wake-up sequences before `HELLOACK`s and `ACK`s are very short. As a result, retransmissions of `HELLOACK`s and `ACK`s also consume little energy, which is why collision attacks against transmissions of `HELLOACK`s and `ACK`s are benign.

Normal unicast frames: Recall that SPLO++ keeps the uncertainty about a permanent neighbor's wake-up time below a user-defined threshold t_t by sending `UPDATE`s in the absence of upper-layer traffic. Thus, wake-up sequences before normal unicast frames do not exceed a maximum duration of about t_t. This limits the severity of collision attacks. In practice, wake-up sequences can get slightly longer than t_t since CSL rounds up the number of required wake-up frames and always has to send one more wake-up frame in order to also reach receivers that missed the beginning of a wake-up frame. Note that collision attacks against the `UPDATE`s themselves may actually reduce a victim node's energy consumption. This is because if an `UPDATE` is not acknowledged after a configurable number of retransmissions, AKES will delete the seemingly inactive neighbor. Thus, in such an occasion, no upper-layer traffic will be sent to a deleted neighbor until reestablishing session keys with him. However, it remains to be investigated if, by blocking certain links, attackers can also cause an increased energy consumption since this prevents the use of certain routes.

7.3.2.3 Acknowledgment Spoofing Attacks

To see that acknowledgment spoofing attacks are prevented, let us assume the contrary that a node A receives an authentic acknowledgment frame f in response to a normal unicast or `ACK` frame g, though the intended receiver B did not successfully receive g. From the construction of CCM nonces we know that f correctly echoed the wake-up counter ω_B that was predicted by A, g's burst index, as well as B's MAC address. Thus, B must have found a normal unicast or `ACK` frame, say h, with g's burst index at the ω_B-th wake up authentic. As f is authentic, h must have been secured with the same up-to-date pairwise session key as f. Yet, no other node than A or B has access to their pairwise session key $K'_{A,B}$ and B can not be the sender of h. This retains A as the sender of h. But, since establishing $K'_{A,B}$, the combination of wake-up counter ω_B and g's burst index is unique in the communication direction A to B, which is why h is g. However, any other node than B would have discarded g due to an inauthentic MIC and hence not acknowledged g. Thus, there is a contradiction.

7.3.2.4 Pulse-Delay Attacks

In Section 7.3.1.2, we already showed that authenticated acknowledgment frames turn out inauthentic if they are delayed by $> t_a$. This means that, if the receiver of an authentic acknowledgment frame uses the contained CSL phase to compute the sender's last wake-up time, the offset due to pulse-delay attacks is upper-bounded by t_a. As also discussed in Section 7.3.1.2, delaying unauthenticated acknowledgment frames by $> t_a$ causes AKES' three-way handshake to fail, thereby preventing pulse-delay attacks while initializing wake-up times up to t_a, as well. Altogether, pulse-delay attacks cause no repercussions if the undetectable offset t_a is taken into account when estimating wake-up times.

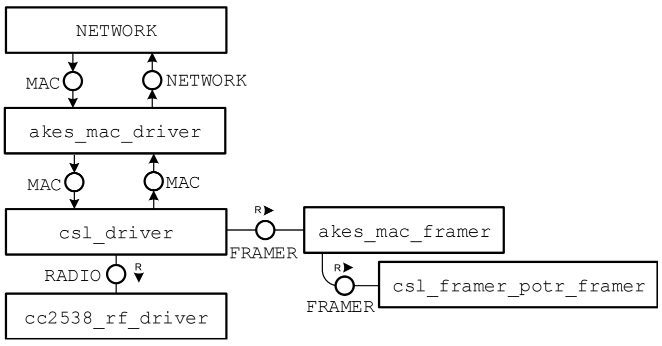


Figure 7.4: Integration of our CSL implementation into Contiki-NG's network stack, as well as into our implementation of AKES

7.3.2.5 Chatty Nodes

POTR++ provides a basis for pinpointing and evicting chatty nodes through the adoption of pairwise session keys. In fact, owing to deriving OTPs from pairwise session keys, a chatty node that sends plenty of normal unicast frames or ACKs with valid OTPs can be considered compromised as no other node than the chatty node and the victim node can access their pairwise session key (provided that the underlying key predistribution is inoculated and opaque). Likewise, authentic HELLOs and HELLOACKs can be attributed to the sender due to authenticating them with pairwise session keys.

7.3.2.6 Deaf Nodes

Deaf nodes are also mitigated by SPLO++ through limiting the maximum length of wake-up sequences that precede unicast frames.

7.4 Implementation

We implemented our denial-of-sleep defenses for CSL as part of a whole new CSL implementation for Contiki-NG. Figure 7.4 shows how our CSL implementation integrates into Contiki-NG's network stack. Instead of using the current RADIO interface "as is", our CSL implementation is written against our enriched RADIO interface, which supports an asynchronous mode of operation. Optionally, our CSL implementation can be configured to act like the standardized version of CSL. In this case, an IEEE 802.15.4-compliant FRAMER is called instead of the csl_framer_potr_framer. Besides, it is also possible to disable MAC layer security, in which case the csl_driver and an IEEE 802.15.4-compliant FRAMER are called directly, i.e., without passing through AKES' modules.

A main feature of our CSL implementation is its support for channel hopping, which generally proved to be an effective means to alleviate interference, as well as fading effects [186]. This feature is realized following Al Nahas et al.'s generic channel hopping extension "MiCMAC" to asynchronous MAC protocols [83]. The rationale of MiCMAC is to do each consecutive periodic wake up on a different pseudo-random channel. As for unicasts, a sender may be able to predict a receiver's current channel and otherwise resorts to transmit on

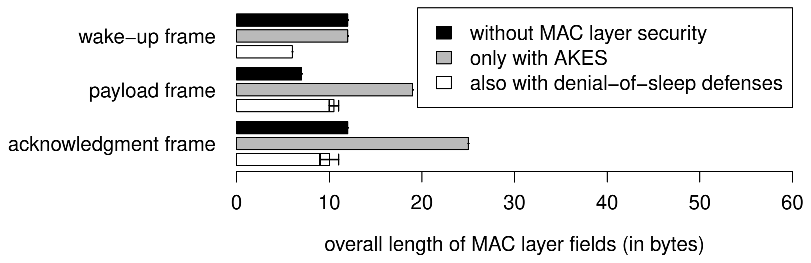


Figure 7.5: MAC layer overhead when unicasting data

one channel for as long as it takes to revisit this channel in the worst case. Fortunately, our implementation works without this fallback mechanism since we derive channels from wake-up counters and MAC addresses, both of which are known to senders of unicast frames. However, as for broadcasts, MiCMAC always requires transmitting on one channel for as long as it takes to revisit this channel in the worst case. Normally, this is a severe drawback of MiCMAC, but POTR++ only requires broadcasting `HELLO`s as usual and requires sending other broadcast frames as unicast frames anyway. Furthermore, `HELLO`s are sent very rarely once the network topology becomes stable.

7.5 Evaluation

In this section, we report on a set of experiments we conducted to assess the overhead and effectiveness of our denial-of-sleep defenses for CSL. In sum, our denial-of-sleep defenses turned out to (i) lower the communication overhead compared to the standardized version of CSL, (ii) incur a low RAM overhead, (iii) make CSL resistant to unicast attacks with normal unicast frames, and (iv) highly mitigate collision attacks.

7.5.1 Communication Overhead

An apparent drawback of our denial-of-sleep defenses for CSL is their addition of new fields, such as OTPs. To asses this overhead, the overall length of the MAC layer fields of wake-up, payload, and acknowledgment frames was logged while unicasting data. Furthermore, this was done using three different configurations. First, MAC layer security was disabled. Second, AKES was then enabled. Finally, all our denial-of-sleep defenses were enabled, too. Throughout, 2-byte OTPs, short addresses, and Security Level 6 were used.

The, maybe surprising, results are shown in Figure 7.5. As for wake-up frames, our denial-of-sleep defenses for CSL actually reduces the overall length of MAC layer fields. This is because POTR++ elides destination addresses and checksums, as well as because POTR++ encodes rendezvous times more concisely. As for payload frames, POTR++ also reduces the overall length of MAC layer fields by eliding checksums and source addresses. Besides, our denial-of-sleep defenses use wake-up counters, which, unlike frame counters, need not be transmitted. As for acknowledgment frames, POTR++, again,

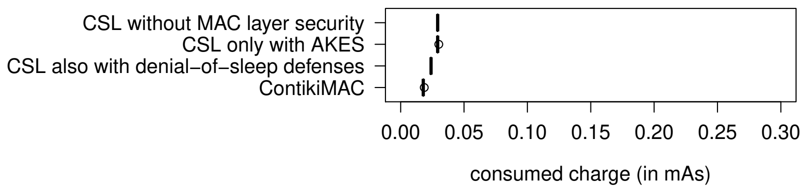


Figure 7.6: Energy consumption per wake up

achieves significant savings by encoding the CSL phase more concisely, as well as by avoiding frame counters and checksums.

Thanks to reducing the length of wake-up frames, our denial-of-sleep defenses for CSL also reduce the energy consumption of CSL's periodic wake ups. This is because CSL only has to wake up for as long as it takes to detect the SHR of a wake-up frame in the worst case. To measure the resulting energy savings, an OpenMote was connected with a μCurrent Gold and a Rigol DS1000E oscilloscope in series like described in Section 3.6.2. Then, the OpenMote's energy consumption per wake up was gauged when using the three configurations mentioned above.

Figure 7.6 shows the results alongside with our results obtained for ContikiMAC in the previous chapter. As we expected, the energy consumption of CSL's periodic wake ups decreases when using our denial-of-sleep defenses.

Another critical design decision of us is to send broadcast frames as unicast frames to each permanent neighbor, one after another. To compare the energy consumption of transmitting broadcast frames as usual and as unicast frames, a network of 20 nearby OpenMotes was set up. The behavior of the OpenMotes was as follows. At first, all OpenMote's established session keys with each other and learned each other's clock drift. Then, each OpenMote began to send maximum-length broadcast frames with a random delay of 0 - 4.5min in between. In the background, each OpenMote ran Contiki-NG's tool Energest to trace the energy consumed for transmitting these broadcasts (which are transmitted as unicast frames), as well as AKES' `HELLO`s (which are transmitted as usual) [158]. For a fair comparison, each OpenMote padded its `HELLO`s with as many zeroes as possible. Initially, all OpenMotes ran our denial-of-sleep-protected version of CSL using $t_t = 2.38\text{ms}$, $\theta = 15\text{ppm}$, $\theta_\eta = 3\text{ppm}$, $t_m = 5\text{min}$, and 16 channels. Then, this experiment was repeated using 8, 4, 2, and 1 channel(s).

As shown in Figure 7.7, the energy consumed for processing is very low, regardless of how broadcasts are sent. This is because our implementation leverages the hardware-accelerated CCM implementation of CC2538 SoCs and widely avoids busy-waiting. Also, the energy consumed for receiving is very low, but is noticeable when transmitting broadcast frames as unicast frames, which is due to performing more CCAs and receiving acknowledgment frames. The bulk of the energy is consumed for transmitting. Moreover, when transmitting broadcast frames as usual, the energy consumption for transmitting increases linearly with the number of channels, whereas, when sending broadcast frames as unicast frames, the energy consumption is independent of the number of channels. On the other hand, when sending broadcast frames as unicast frames, the energy consumption depends on the number of permanent neighbors, which was always 19 in this experiment. That said, many more neighbors than 19 do

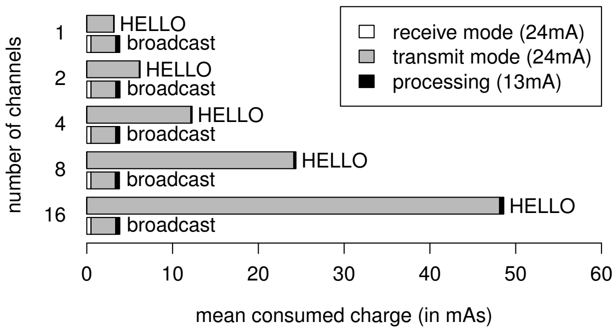


Figure 7.7: Consumed charge per transmission of a zero-padded `HELLO` and a maximum-length broadcast frame

often not even fit in the constrained RAM of IoT devices [187]. Thus, these results show that sending broadcast frames as unicast frames is not only a viable choice, but actually advantageous in combination with MiCMAC. Even when not using MiCMAC, sending broadcast frames as unicast frames does not consume much more energy compared to sending broadcast frames as usual. This can be attributed to the prediction of clock drifts by SPLO++, which shortens the length of wake-up sequences that precede normal unicast frames.

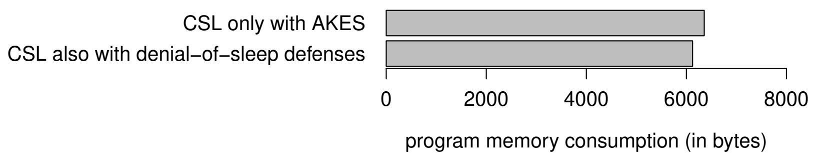
7.5.2 Memory Overhead

For measuring the memory overhead due to our denial-of-sleep defenses, the tool `arm-none-eabi-size` was used. As a baseline for comparison, the memory consumption of using CSL without MAC layer security was measured. Then, the memory overhead due to enabling AKES was measured. Next, the memory consumption when also enabling our denial-of-sleep defenses was measured. These measurements were repeated with different numbers of permanent neighbor slots, whereas the number of tentative neighbor slots was fixed to 5. Throughout, short addresses and Security Level were used.

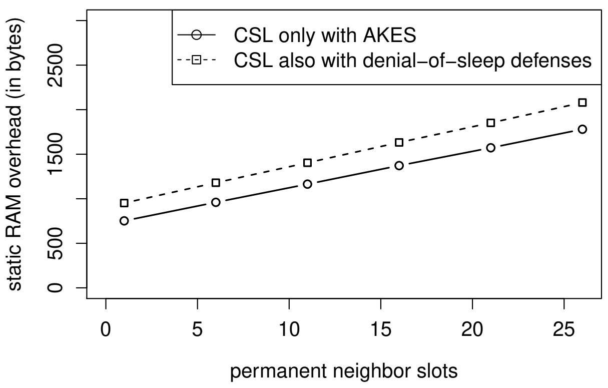
The results are shown in Figure 7.8. Our denial-of-sleep defenses even save a few bytes of program memory, which comes down to the streamlined frame format of POTR++ and Contiki-NG's unoptimized assembling and parsing of IEEE 802.15.4-compliant frames. As for the RAM consumption, our denial-of-sleep defenses incur a small overhead per permanent neighbor slot. This overhead can be attributed to the prediction of clock drifts by SPLO++, which requires storing extra data per permanent neighbor.

7.5.3 Denial-of-Sleep Resilience

The following experiment compares the energy consumption of protected and unprotected versions of CSL and ContikiMAC under unicast attacks. An OpenMote was connected with a μCurrent Gold and a Rigol DS1000E oscilloscope in series like described in Section 3.6.2. In six successive runs, the OpenMote's energy consumption was gauged while receiving injected maximum-length unicast data frames if running (i) CSL without MAC layer security, (ii) CSL with AKES, (iii) CSL also with our denial-of-sleep defenses ($t_t = 2.38$ms; $\theta = 15$ppm;



(a)



(b)

Figure 7.8: Memory overhead due to our denial-of-sleep defenses for CSL

θ_η = 3ppm; t_m = 5min), (iv) ContikiMAC without MAC layer security, (v) ContikiMAC with AKES, and (vi) ContikiMAC also with our denial-of-sleep defenses (θ = 15ppm). In each run, 200 samples were taken. Throughout, 2-byte OTPs, short addresses, Security Level 6, and Parameter Set 6 were used.

The results are shown in Figure 7.9. As for the configuration of CSL without any MAC layer security, unicast attacks are rather severe since injected unicast data frames are not only fully received, but also acknowledged. Moreover, when enabling IEEE 802.15.4 security, acknowledgment frames additionally carry a CCM MIC, thereby aggravating unicast attacks. As for our denial-of-sleep-protected version of CSL, the energy consumption actually slightly decreases compared to receiving no wake-up frame at all. There are two reasons for this. First, if an OTP turns out invalid, POTR++ disables the receive mode immediately, which may happen earlier than when CSL would stop listening for a wake-up frame. Second, OpenMotes consume less energy in receive mode when there is a strong input signal, which is the case under unicast attacks. The denial-of-sleep-unprotected versions of ContikiMAC behave like the denial-of-sleep-unprotected versions of CSL. They not only fully receive injected unicast data frames, but also acknowledge them. Our denial-of-sleep-protected version of ContikiMAC greatly mitigates unicast attacks, but, does not nullify the additional energy consumption due to unicast attacks.

To compare the resilience to collision attacks of protected and unprotected versions of CSL and ContikiMAC, the following experiment was conducted. In six successive runs, two OpenMotes ran the same six configurations that were mentioned above. In each run, the two OpenMotes sent maximum-length uni-

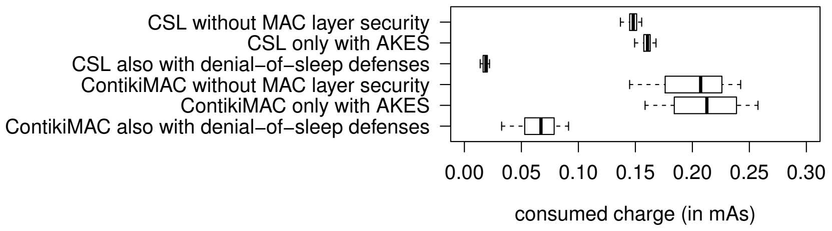


Figure 7.9: Energy consumption per reception of an injected maximum-length unicast data frame

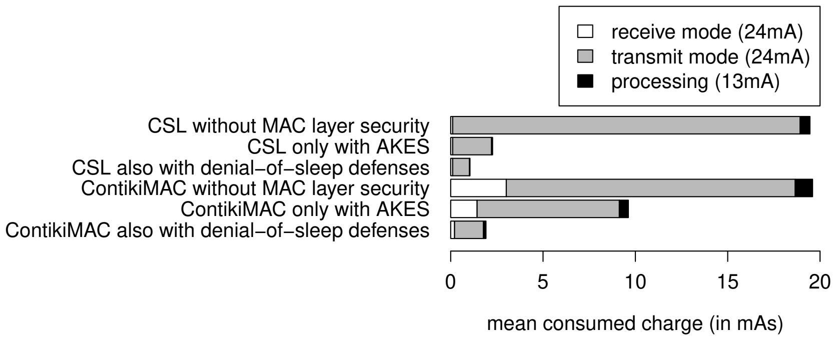


Figure 7.10: Energy consumption per transmission of a maximum-length unicast data frame in the face of collision attacks

cast data frames to each other with a random delay of 0 - 4.5min in between. Further, to simulate collision attacks, these unicast data frames were not acknowledged. The energy consumed for transmitting and retransmitting these unicast data frames was traced via Energest. Throughout, the maximum number of retransmissions was 5, bursts were disabled, and a wake-up interval of $t_w = 125$ms was used.

As shown in Figure 7.10, when using no MAC layer security, CSL turned out to be particularly susceptible to collision attacks. This is because the simulated collision attacks prevent both OpenMotes from learning each other's wake-up time, thus causing CSL to always transmit wake-up sequences that span a whole wake-up interval. When enabling IEEE 802.15.4 security, collision attacks become less severe. This positive result comes down to the use of AKES for session key establishment. Specifically, in the course of AKES' three-way handshake, each OpenMote sends one unicast frame to the other OpenMote and hence learns the other OpenMote's wake-up time. Furthermore, AKES sends `UPDATE`s so as to check if a permanent neighbor is still in range. As a side effect, these `UPDATE`s update wake-up times and hence reduce the uncertainty about the other OpenMote's wake-up time. Of course, an attacker can also interfere with these `UPDATE`s, but this would usually decrease the energy consumption of victim nodes in the long run. This is because AKES would delete a seemingly inactive neighbor and hence stop upper-layer traffic to the deleted neighbor. Nevertheless, AKES does not reliably defend against collision attacks because AKES suppresses `UPDATE`s to permanent neighbors that recently sent a fresh authentic broadcast or unicast frame, which, unlike fresh authentic acknowledgment frames, do not necessarily update wake-up times as a side effect. Our denial-of-sleep-protected version of CSL further reduces the mean consumed charge due to collision attacks from 2.268mAs to 1.044mAs by enforcing a maximum uncertainty about a permanent neighbor's wake-up time throughout a session. Thus, as opposed to AKES, SPLO++ reliably protects against collision attacks. Similar to an unsecured version of CSL, an unsecured version of ContikiMAC strobes for a whole wake-up interval if the receiver's wake-up time is unknown, thereby rendering collision attacks severe. Enabling AKES alleviates the effects of collision attacks due to sending `UPDATE`s and hence updating wake-up times. However, ContikiMAC's original phase-lock optimization relearns the wake-up time of a neighbor if unicast transmissions to the neighbor tend to fail, aggravating consecutive collision attacks. SPLO never relearns wake-up times and modifies AKES to send `UPDATE`s to permanent neighbors whose wake-up time was not updated for a critical period of time. Yet, unlike SPLO++, SPLO does not predict clock drifts, which leads to lower mitigation of collision attacks.

7.6 Summary

In this chapter, we have proposed CSL-enabled and security-enhanced versions of POTR, SPLO, and ILOS. Together, POTR++, SPLO++, and ILOS++ have made CSL resistant to broadcast, unicast, droplet, pulse-delay, and acknowledgment spoofing attacks, as well as highly resilient to collision attacks and deaf nodes. Furthermore, we have argued that our denial-of-sleep defenses for CSL also provide a basis for pinpointing and evicting chatty nodes. Thus, our denial-of-sleep-protected version resists, or at least greatly mitigates, all denial-of-sleep attacks against CSL we identified in Chapter 5.

Compared to our denial-of-sleep-protected version of ContikiMAC, our denial-of-sleep-protected version of CSL has three security advantages. First, POTR++, SPLO++, and ILOS++ enable the detection and eviction of chatty nodes because of using pairwise session keys instead of group session keys. Second, SPLO++ better mitigates collision attacks and deaf nodes owing to predicting clock drifts. Third, POTR++ makes CSL resistant to broadcast, unicast, and droplet attacks, rather than just resilient.

Additionally, our denial-of-sleep-protected version of CSL inherits three security unrelated advantages from CSL. First, unlike ContikiMAC, CSL does not require configuring CCA thresholds for detecting incoming transmissions, which renders CSL more resilient to interference and easier to configure. Second, unlike ContikiMAC, CSL does not require padding short frames with extra bytes. Third, unlike ContikiMAC, CSL does not suffer from duplicate receptions of broadcast frames. But, CSL also has two security unrelated disadvantages compared to ContikiMAC. On the one hand, CSL's base energy consumption is slightly higher. On the other hand, CSL requires sending certain frames back-to-back, which is not supported by all IEEE 802.15.4 transceivers.

Chapter 8

Conclusions and Future Work

This chapter summarizes this dissertation, paraphrases follow-up master's theses, suggests topics for future research, and closes with general observations.

8.1 Summary of Research

Several radio technologies lend themselves to low-power IoT applications. Among them, IEEE 802.15.4 features several PHY options, reliable mesh topologies, zero operational costs, cheap off-the-shelf hardware, as well as standardized upper-layer protocols. However, like all radio technologies for low-power IoT applications we have looked at, IEEE 802.15.4 comes without any protection against denial-of-sleep attacks. Moreover, the security services of IEEE 802.15.4, collectively called *IEEE 802.15.4 security*, have three more limitations. First, IEEE 802.15.4 security leaves the whole issue of key management unspecified. Second, the frame counter-based replay protection of IEEE 802.15.4 security only provides sequential freshness and thus does not filter out delayed frames. Third, receivers of a fresh authentic acknowledgment frame can not be sure whether the corresponding unicast frame was successfully received. Fixing these three limitations of IEEE 802.15.4 security is a timely since protocols on top of IEEE 802.15.4 rely on IEEE 802.15.4 security.

We have proposed a denial-of-sleep-resilient MAC layer for IEEE 802.15.4 networks, which overcomes the other three limitations of IEEE 802.15.4 security, too. Our denial-of-sleep-resilient MAC layer mainly consists of a denial-of-sleep-resilient protocol for establishing session keys among neighboring IEEE 802.15.4 nodes, called AKES, and a denial-of-sleep-protected version of CSL. AKES is particularly designed for low-power IEEE 802.15.4 nodes, handles topology changes, and survives reboots. Additionally, a main feature of AKES is its denial-of-sleep resilience. This feature stems, on the one hand, from using key predistribution, and, on the other hand, from rate-limiting outgoing session key establishment-related messages. Our denial-of-sleep-protected version of CSL combines existing ideas with plenty new ones to protect CSL against denial-of-sleep attacks. For example, our denial-of-sleep-protected version of CSL adopts the emergent OTP pattern for protecting CSL against unicast and droplet attacks. Basically, the OTP pattern is to embed OTPs in synchronization or frame headers at the sender side and to validate the embedded OTPs at the receiver side. If an OTP turns out invalid, the receiver cancels the reception, thereby stopping further energy loss. Thus far, it was however unclear how to synergistically integrate the OTP pattern with basic wireless security and how to practically implement the OTP pattern. Our denial-of-sleep-protected version of CSL resolves these unclarities by (i) integrating with CCM and AKES, and (ii) relying only on capabilities of commodity IEEE 802.15.4 transceivers.

Unfortunately, the embedding of OTPs only protects against a small subset of denial-of-sleep attacks against CSL. Further denial-of-sleep attacks against CSL are pulse-delay attacks, acknowledgment spoofing attacks, collision attacks, chatty nodes, and deaf nodes. Except for pulse-delay attacks, no effective defenses had existed against these additional denial-of-sleep attacks. Hence, we have devised defenses against those denial-of-sleep attacks from scratch.

For achieving strong freshness, our denial-of-sleep-resilient MAC layer uses our newly introduced concept of wake-up counters. Wake-up counters combine the benefits of frame counters and timeslot indices. Like frame counters, wake-up counters do not require network-wide time synchronization. This is highly beneficial since securing network-wide time synchronization against internal attackers is a hard-to-solve issue in its own right [161]. Furthermore, like timeslot indices, wake-up counters do not incur any per-frame overhead and help in achieving strong freshness.

Our denial-of-sleep-protected version of CSL also uses wake-up counters for deriving OTPs. Thus far, it was only considered to derive OTPs from frame counters [42, 44], timeslot indices [43, 72], or random numbers [40]. However, each of these approaches has its drawbacks. As for frame counters, if an OTP is guessed correctly, resynchronization may become necessary [44]. As for timeslot indices, they require network-wide time synchronization. As for random numbers, they need to be received prior to validating an OTP, thus deferring the on-the-fly rejection of unwanted frames. None of these drawbacks arises when deriving OTPs from wake-up counters.

To evaluate our denial-of-sleep-resilient MAC layer, we have integrated it into the network stack of Contiki-NG. The target platform of our implementation is the CC2538 SoC, which is built into many IoT devices, such as OpenMotes and RE-Motes [144]. CC2538 SoCs comprise up to 512KB of program memory, up to 32KB of RAM, and an 2.4-GHz O-QPSK IEEE 802.15.4 transceiver. This built-in transceiver, in particular, supports parsing incoming frames during reception, as well as sending IEEE 802.15.4 frames back-to-back. These capabilities are necessary to implement the on-the-fly rejection of unwanted frames and the wake-up sequences of CSL. Yet, not all IEEE 802.15.4 transceivers support sending IEEE 802.15.4 frames back-to-back, which is why we think our work on securing ContikiMAC against denial-of-sleep attacks is also relevant. Yet, a major disadvantage of ContikiMAC is its detection of incoming transmissions via CCAs. The problem with this is that if the CCA threshold is too high, transmissions may go undetected, while if the CCA threshold is too low, the susceptibility to interference increases [81, 167, 168]. Even when autoconfiguring CCA thresholds to trade off between undetected transmissions and susceptibility to interference, these issues do not vanish completely [81].

On our way to a fully functional denial-of-sleep-resilient MAC layer, we have also solved the problem of generating cryptographic random numbers on

CC2538-based IoT devices. To this end, we have proposed a CSPRNG, which seeds itself with power-up SRAM states and radio noise. For extracting seeds from power-up SRAM states, we have proposed a novel method, which is practical, information-theoretically secure, and insusceptible to many randomness-reducing factors [49, 50, 51, 52, 53]. However, our method for extracting seeds from power-up SRAM states depends on a feature that is only available on special hardware platforms, such as the CC2538 or the SiM3U167-B-GDI [14, 129]. Thus, care must be taken when selecting a hardware platform so that either this feature or an appropriate alternative entropy source is available.

8.2 Follow-Up Master's Theses

8.2.1 Protocol Verification of AKES

Eirik Klevstad formally verified AKES' establishment of pairwise session keys in his master's thesis [188]. More specifically, he modeled AKES' three-way handshake using the protocol verification tool Scyther. As a result, he found AKES secure concerning all checked security properties, namely entity authentication, implicit key authentication, explicit key authentication, known-key secrecy, key control, and secrecy of key. Entity authentication requires that both communication partners that are establishing a pairwise session key mutually authenticate each other in this process. Implicit key authentication requires that an established pairwise session key is only recoverable by the rightful communication partners. Explicit key authentication additionally requires the rightful communication partners to mutually confirm the knowledge of the established pairwise session key. Known-key secrecy requires that the established pairwise session key is secret from attackers, even when given access to previously established pairwise session keys. Key control requires that no communication party can enforce the establishment of a specific pairwise session key. Secrecy of key requires that the established pairwise session key is secret from attackers.

8.2.2 Key Revocation and Rekeying for AKES

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 [145]. In our context, key revocation and rekeying is usually done as part of evicting a node from its IEEE 802.15.4 network. A reason for evicting a node, e.g., is that its cryptographic keys leaked to an attacker or that a user wishes to relocate an IoT device from one IEEE 802.15.4 network to another. To do an eviction in a "clean" manner, it is necessary to (i) remove AKES' pairwise session keys shared with the node that is being evicted, (ii) change AKES' group session keys shared with the node that is being evicted, and (iii) update AKES' predistributed keying material.

To fill the gap that AKES lacks a key revocation and rekeying protocol, Daniel Werner proposed a CoAP-based solution in his master's thesis [189]. In his solution, a base station sends a CoAP request, called `Revocation Unicast`, to every node in the IEEE 802.15.4 network, one after another. A `Revocation unicast` contains, on the one hand, the MAC address of the node that is being evicted and, optionally, new keying material. After processing a `Revocation` `unicast`, a CoAP response is sent back to the base station. A major benefit of his solution compared to current solutions for key revocation and rekeying is that his solution provides feedback on the progress of key revocation and rekeying to the user. In particular, the base station can report to the user whether certain nodes could not be reached. A problem with his solution, however, arises when the node being evicted is controlled by an attacker and may hence deny to forward specific CoAP messages, particularly `Revocation unicasts`.

We addressed the problem with Daniel Werner's solution in [11]. Basically, our approach is to implement key revocation and rekeying at the MAC layer, without involving upper layers. At the MAC layer, we perform key revocation and rekeying in a breadth-first manner. That is, we first reach out to IEEE 802.15.4 nodes at the edge of the network and ask them to report back on their MAC layer neighbors. Subsequently, we reach out to these MAC layer neighbors via source routing and also ask them to report back on their MAC layer neighbors, and so on. Thanks to source routing, we can avoid paths via nodes that are being evicted and can even try alternative paths if necessary.

8.2.3 Denial-of-Sleep-Resilient Opportunistic Routing

"ORPL" can be thought of as a cross-layer optimization between RPL and ContikiMAC [80]. The approach of ORPL consists in forwarding an IPv6 packet to the hop that wakes up next, rather than forwarding an IPv6 packet to a specific hop as RPL does it now. ORPL implements this by disabling ContikiMAC's phase-lock optimization and instead strobing an IPv6 packet until an acknowledgment frame comes back. Furthermore, receivers only reply with an acknowledgment frame if they are willing to forward a received IPv6 packet. In comparison to RPL, ORPL features lower latencies, higher delivery ratios, lower energy consumption, as well as less RAM consumption [80].

Unfortunately, our denial-of-sleep-protected version of ContikiMAC conflicts with ORPL's "anycasts". Recall that POTR's unicast OTPs only wake up a specific neighbor, whereas POTR's broadcast OTPs wake up all neighbors. Thus, when embedding a unicast OTP in an anycast, only a specific node will wake up, whereas when embedding a broadcast OTP in an anycast, attackers can replay the broadcast OTP to wake up uninvolved nodes within a short time window. Moreover, ILOS requires delaying broadcast OTPs so that receivers can correctly restore the wake-up counter from which a broadcast OTP was derived. Apart from this conflict, another problem with ORPL is that jamming acknowledgment frames can lead to a duplication of IPv6 packets because multiple nodes may then decide to forward an IPv6 packet.

Felix Wolff considered two solutions to these issues in his master's thesis [190]. On the one hand, he evolved our broadcast OTPs into *anycast OTPs*. The main advantage of anycast OTPs is that they do not require delaying anycasts. Yet, anycast OTPs do neither help against the duplication of IPv6 packets nor prevent attackers from waking up uninvolved nodes within a short time window. On the other hand, he considers adapting RPL to forward IPv6 packets to any possible hop that wakes up next according to the wake-up times maintained by SPLO. This approach leaves no window for attack, prevents the duplication of IPv6 packets, and still brings advantages concerning delivery ratios, energy consumption, and reliability according to his experimental results.

8.3 Topics for Future Research

While our denial-of-sleep-resilient MAC layer provides a basis for detecting and evicting chatty nodes, the actual detection and eviction of chatty nodes is not performed, yet. A logical next step is thus to take advantage of the fact that a node that sends plenty of frames with valid OTPs or authentic CCM MICs can be considered compromised, provided that the underlying key predistribution scheme is inoculated and opaque. One possible reaction to such an occasion is to evict the chatty node via key revocation and rekeying. Another possible reaction is to ignore further frames from the chatty node temporarily or permanently.

Furthermore, numerous countermeasures against collision attacks and deaf nodes remain to be explored. For example, to further reduce the energy consumption of retransmissions, IEEE 802.15.4 nodes can adjust their transmission power according to the intended receiver. Such power control techniques already exist and can be tailored to our denial-of-sleep-protected version of CSL [191, 192]. A synergistic effect of doing so will be that the energy consumption of sending broadcast frames as unicast frames to each permanent neighbor, one after another, decreases. Another countermeasure is to detect anomalous amounts of retransmissions like suggested by Ren et al. [41], but to react differently than suggested by Ren et al. by cancelling further retransmissions. Finally, cross-layer countermeasures against collision attacks and deaf nodes could be considered. For instance, if a 6LoWPAN fragment could not be forwarded, it is pointless to try forwarding pending fragments. As a result of not sending pending fragments when a fragment could not be conveyed, there will be less opportunities for collision attacks and deaf nodes.

Apart from transmission powers, three other parameters of our denial-of-sleep-protected version of CSL could be autoconfigured, too. First, the selection of channels could be autotuned via reinforcement learning [193]. Second, CSL's wake-up interval could be adapted dynamically at runtime [86, 87]. Third, PAN identifiers are currently preloaded, but could be autoconfigured so as to avoid overhearing `HELLO`s and `HELLOACK`s from co-located IEEE 802.15.4 networks [29].

A last suggested topic for future research concerns the integration of AKES with the upper-layer routing protocol. Presently, the coupling between AKES and the upper-layer routing protocol is loose. In fact, AKES operates independently from the upper-layer routing protocol. However, it may be advantageous to, e.g., let AKES and RPL cooperate on discovering adjacent IEEE 802.15.4 nodes. In this regard, it is also worthwhile to consider implementing a mesh-under routing protocol instead of sticking with RPL so as to avoid cross-layer dependencies. Actually, a standardized mesh-under version of RPL exists already, namely IEEE 802.15.10 [62]. Alternatively, Deng et al.'s INtrusion-tolerant routing protocol for wireless SEnsor NetworkS (INSENS) may lead to synergies with our denial-of-sleep-resilient MAC layer [194]. INSENS collects neighborhood information, which can be provided by AKES, at a central node. This central node then distributes ready-to-use routing tables in the network. A synergistic effect of INSENS is that key revocation and rekeying can be performed like we proposed [11], but without having to collect topology information in this process. Another synergistic effect is that routing then works without broadcast transmissions, which are much more energy consuming than unicast transmissions when using our denial-of-sleep-resilient MAC layer.

8.4 General Observations

Regardless of using public- or symmetric-key cryptography, IoT devices have to be preloaded with cryptographic material in addition to other configuration settings. Our denial-of-sleep-resilient MAC layer is no exception in this regard since it, e.g., requires preloading IoT devices with keying material for use by the underlying key predistribution scheme. Yet, preloading cryptographic material via cables is not user-friendly, and preloading cryptographic material via radio often raises a chicken-and-egg problem since cryptographic material needs to be conveyed securely via cryptographic material. Our self-seeding CSPRNG resolves this chicken-and-egg problem by enabling IoT devices to generate cryptographic random numbers entirely by themselves. This, in turn, enables the use of secure wireless preloading schemes [8, 115, 116]. For example, in our 6doku [8], cryptographic random numbers are needed for generating hash chains, which are then used for mutual authentication between an IoT device and the owners's smartphone. Subsequently, 6doku generates a pairwise key between the IoT device and the owners's smartphone via PHY key generation, and finally uses the pairwise key for securing the wireless preloading process.

Another observation is a dependency of asynchronous MAC protocols on session keys. Since asynchronous MAC protocols neither use network-wide nor cluster-wide time synchronization, they are typically combined with a frame counter-based replay protection instead of a timeslot index-based replay protection. However, a frame counter-based replay protection seems only practical when establishing session keys. Otherwise, when using predistributed keys unchanged, it becomes, e.g., necessary to persist anti-replay data across reboots so as to prevent replay attacks after reboots. This is problematic since the only non-volatile memory on most IoT devices is flash memory, which is slow, energy consuming, as well as prone to wear. As a result, there is a dependency of asynchronous MAC protocols on session keys. This dependency does not vanish completely when switching to a wake-up counter-based replay protection. When using wake-up counters, we found session keys still useful in two regards. First, for avoiding nonce reuses without storing data in non-volatile memory. Second, for handling the case when a wake-up counter is about to reach its maximum value. In such an occasion, we simply issue a reboot, which causes the wake-up counter to start over from zero and new session keys to be established.

Though this dissertation has focused on IEEE 802.15.4, patterns emerged that may help in securing other radio technologies against denial-of-sleep attacks, too. One of these patterns certainly is the use of OTPs, which serve well in countering broadcast, unicast, and droplet attacks. Admittedly, retrofitting OTPs into an existing radio technology is difficult because it necessitates changes to frame formats and because transceivers have to support validating OTPs during reception. But, this pattern can be adopted in future versions of radio technologies, brand new radio technologies, and in proprietary systems. A complementary pattern is to derive OTPs from wake-up counters, which is applicable when an asynchronous MAC protocol is used. Furthermore, synergies arise when the asynchronous MAC protocol learns the wake-up times of neighboring nodes anyway. Finally, a third pattern that emerged is the use of LBCs. LBCs enable enforcing a maximum long-term rate of incoming and outgoing messages, while permitting the rate of incoming and outgoing messages to overshoot temporarily, respectively.

Bibliography

[1] Konrad-Felix Krentz and Christoph Meinel. Denial-of-sleep defenses for IEEE 802.15.4 coordinated sampled listening (CSL). *Computer Networks*, 148(15):60–71, 2019.

[2] Konrad-Felix Krentz, Christoph Meinel, and Hendrik Graupner. Denial-of-sleep-resilient session key establishment for 802.15.4 security: from adaptive to responsive. In *Proceedings of the International Conference on Embedded Wireless Systems and Networks (EWSN 2018)*. Junction, 2018.

[3] Konrad-Felix Krentz, Christoph Meinel, and Hendrik Graupner. More lightweight, yet stronger 802.15.4 security through an intra-layer optimization. In *Proceedings of the 10th International Symposium on Foundations & Practice of Security (FPS 2017)*. Springer, 2017.

[4] Konrad-Felix Krentz, Christoph Meinel, and Hendrik Graupner. Secure self-seeding with power-up SRAM states. In *Proceedings of the 22nd IEEE Symposium on Computers and Communications (ISCC 2017)*. IEEE, 2017.

[5] Konrad-Felix Krentz, Christoph Meinel, and Hendrik Graupner. Countering three denial-of-sleep attacks on ContikiMAC. In *Proceedings of the International Conference on Embedded Wireless Systems and Networks (EWSN 2017)*, pages 108–119. Junction, 2017.

[6] Konrad-Felix Krentz, Christoph Meinel, and Maxim Schnjakin. POTR: practical on-the-fly rejection of injected and replayed 802.15.4 frames. In *Proceedings of the International Conference on Availability, Reliability and Security (ARES 2016)*, pages 59–68. IEEE, 2016.

[7] Konrad-Felix Krentz and Christoph Meinel. Handling reboots and mobility in 802.15.4 security. In *Proceedings of the 31st Annual Computer Security Applications Conference (ACSAC '15)*, pages 121–130. ACM, 2015.

[8] Konrad-Felix Krentz and Gerhard Wunder. 6doku: towards secure over-the-air preloading of 6LoWPAN nodes using PHY key generation. In *Proceedings of the European Conference on Smart Objects, Systems and Technologies (Smart SysTech 2015)*. VDE, 2015.

[9] Konrad-Felix Krentz and Gerhard Wunder. 6LoWPAN security: avoiding hidden wormholes using channel reciprocity. In *Proceedings of the 4th*

International Workshop on Trustworthy Embedded Devices (TrustED '14), pages 13–22. ACM, 2014.

[10] Konrad-Felix Krentz, Hosnieh Rafiee, and Christoph 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.

[11] Benedikt Bock, Jan-Tobias Matysik, Konrad-Felix Krentz, and Christoph Meinel. Link layer key revocation and rekeying for the adaptive key establishment scheme. In *Proceedings of the IEEE 5th World Forum on Internet of Things (WF-IoT)*. IEEE, 2019.

[12] Felix Seidel, Konrad-Felix Krentz, and Christoph Meinel. Deep en-route filtering of constrained application protocol (CoAP) messages on 6LoWPAN border routers. In *Proceedings of the IEEE 5th World Forum on Internet of Things (WF-IoT)*. IEEE, 2019.

[13] Klara Seitz, Sebastian Serth, Konrad-Felix Krentz, and Christoph Meinel. Demo: enabling en-route filtering for end-to-end encrypted CoAP messages. In *Proceedings of the 15th ACM Conference on Embedded Networked Sensor Systems (SenSys '17)*. ACM, 2017.

[14] *CC2538 SoC for 2.4-GHz IEEE 802.15.4 & ZigBee/ZigBee IP Applications User's Guide (Rev. C)*. Texas Instruments. `http://www.ti.com/lit/ug/swru319c/swru319c.pdf`.

[15] Jean-Philippe Vasseur and Adam Dunkels. *Interconnecting Smart Objects with IP: The next Internet*. Morgan Kaufmann, 2010.

[16] Adrian Perrig, Robert Szewczyk, JD Tygar, Victor Wen, and David E Culler. SPINS: security protocols for sensor networks. *Wireless networks*, 8(5), 2002.

[17] Johann Großschädl, Alexander Szekely, and Stefan 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] ZigBee IP Specification, 2014. Revision 34.

[19] 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.

[20] Antonio de la Piedra, An Braeken, and Abdellah 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.

[21] K. C. Hewage, S. Raza, and T. Voigt. Protecting glossy-based wireless networks from packet injection attacks. In *Proceedings of the 14th International Conference on Mobile Ad Hoc and Sensor Systems (MASS)*, pages 37–45. IEEE, 2017.

[22] M. Brownfield, Yatharth Gupta, and N. Davis. Wireless sensor network denial of sleep attack. In *Proceedings of the Sixth Annual IEEE SMC Information Assurance Workshop (IAW '05)*, pages 356–364. IEEE, 2005.

[23] Frank Stajano and Ross J. Anderson. The resurrecting duckling: Security issues for ad-hoc wireless networks. In *Proceedings of the 7th International Workshop on Security Protocols*, pages 172–194. Springer, 1999.

[24] T. Martin, M. Hsiao, Dong Ha, and J. Krishnaswami. Denial-of-service attacks on battery-powered mobile computers. In *Proceedings of the Second IEEE Annual Conference on Pervasive Computing and Communications (PERCOM)*, pages 309–318. IEEE, 2004.

[25] D. C. Nash, T. L. Martin, D. S. Ha, and M. S. Hsiao. Towards an intrusion detection system for battery exhaustion attacks on mobile computing devices. In *Proceedings of the 3rd IEEE International Conference on Pervasive Computing and Communications Workshops (PerCom 2005)*, pages 141–145. IEEE, 2005.

[26] D. Halperin, T. S. Heydt-Benjamin, B. Ransford, S. S. Clark, B. Defend, W. Morgan, K. Fu, T. Kohno, and W. H. Maisel. Pacemakers and implantable cardiac defibrillators: Software radio attacks and zero-power defenses. In *Proceedings of the 2008 IEEE Symposium on Security and Privacy (S&P 2008)*, pages 129–142. IEEE, 2008.

[27] Francesco Palmieri, Sergio Ricciardi, Ugo Fiore, Massimo Ficco, and Aniello Castiglione. Energy-oriented denial of service attacks: an emerging menace for large cloud infrastructures. *The Journal of Supercomputing*, 71(5):1620–1641, 2015.

[28] C. Li, Z. Wang, X. Hou, H. Chen, X. Liang, and M. Guo. Power attack defense: Securing battery-backed data centers. In *Proceedings of the 2016 ACM/IEEE 43rd Annual International Symposium on Computer Architecture (ISCA)*, pages 493–505. IEEE, 2016.

[29] IEEE Standard 802.15.4-2015, 2016.

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

[31] S. Sciancalepore, G. Piro, E. Vogli, G. Boggia, L. A. Grieco, and G. Cavone. LICITUS: a lightweight and standard compatible framework for securing layer-2 communications in the IoT. *Computer Networks*, 108 (Supplement C):66–77, 2016.

[32] Shahid Raza, Thiemo Voigt, and Vilhelm Juvik. Lightweight IKEv2: a key management solution for both compressed IPsec and IEEE 802.15.4 security. In *Proceedings of the IETF International Workshop on Smart Object Security*. IETF, 2012.

[33] Panagiotis Ilia, George Oikonomou, and Theo Tryfonas. Cryptographic key exchange in IPv6-based low power, lossy networks. In *Proceedings of the IFIP International Workshop on Information Security Theory and Practices (WISTP 2013)*, pages 34–49. Springer, 2013.

[34] Jing 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.

[35] Chae Hoon Lim. LEAP++: a robust key establishment scheme for wireless sensor networks. In *Proceedings of the 28th International Conference on Distributed Computing Systems Workshops (ICDCS '08)*, pages 376–381. IEEE, 2008.

[36] Chris Karlof and David Wagner. Secure routing in wireless sensor networks: attacks and countermeasures. *Elsevier's AdHoc Networks Journal*, 1(2–3), 2003.

[37] David R. Raymond, R.C. Marchany, M.I. Brownfield, and S.F. Midkiff. Effects of denial-of-sleep attacks on wireless sensor network MAC protocols. *IEEE Transactions on Vehicular Technology*, 58(1):367–380, 2009.

[38] A. D. Wood and J. A. Stankovic. Denial of service in sensor networks. *IEEE Computer*, 35(10):54–62, 2002.

[39] David Richard Raymond. *Denial-of-sleep vulnerabilities and defenses in wireless sensor network MAC protocols.* PhD thesis, Virginia Polytechnic Institute and State University, 2008.

[40] Ching-Tsung Hsueh, Chih-Yu Wen, and Yen-Chieh Ouyang. A secure scheme against power exhausting attacks in hierarchical wireless sensor networks. *IEEE Sensors Journal*, 15(6):3590–3602, 2015.

[41] Qingchun Ren and Qilian Liang. Secure media access control (MAC) in wireless sensor networks: intrusion detections and countermeasures. In *Proceedings of the 15th IEEE International Symposium on Personal, Indoor and Mobile Radio Communications (PIMRC 2004)*, pages 3025–3029. IEEE, 2004.

[42] R. Falk and H.-J. Hof. Fighting insomnia: a secure wake-up scheme for wireless sensor networks. In *Proceedings of the Third International Conference on Emerging Security Information, Systems and Technologies (SECURWARE '09)*, pages 191–196, 2009.

[43] Salem Aljareh and Anastasios Kavoukis. Efficient time synchronized one-time password scheme to provide secure wake-up authentication on wireless sensor networks. *International Journal of Advanced Smart Sensor Network Systems (IJASSN)*, 3:1–11, 2013.

[44] Angelo T. Capossele, Valerio Cervo, Chiara Petrioli, and Dora Spenza. Counteracting denial-of-sleep attacks in wake-up-based sensing systems. In *Proceedings of the IEEE International Conference on Sensing, Communication and Networking (SECON 2016)*, pages 1–9. IEEE, 2016.

[45] Yevgeniy Dodis, David Pointcheval, Sylvain Ruhault, Damien Vergniaud, and Daniel Wichs. Security analysis of pseudo-random number generators with input: /dev/random is not robust. In *Proceedings of the*

2013 ACM SIGSAC Conference on Computer & Communications Security (CCS '13), pages 647–658. ACM, 2013.

[46] Elaine Barker, John Kelsey, et al. Recommendation for random number generation using deterministic random bit generators. 2012. NIST special publication 800-90A.

[47] D. Eastlake, J. Schiller, and S. Crocker. Randomness Requirements for Security. RFC 4086, 2005.

[48] Helena Handschuh, Geert-Jan Schrijen, and Pim Tuyls. Hardware intrinsic security from physically unclonable functions. *Information Security and Cryptography*, pages 39–53, 2010.

[49] D.E. Holcomb, W.P. Burleson, and K. Fu. Power-up SRAM state as an identifying fingerprint and source of true random numbers. *IEEE Transactions on Computers*, 58(9):1198–1210, 2009.

[50] Anthony Van Herrewege, Vincent van der Leest, André Schaller, Stefan Katzenbeisser, and Ingrid Verbauwhede. Secure PRNG seeding on commercial off-the-shelf microcontrollers. In *Proceedings of the 3rd International Workshop on Trustworthy Embedded Devices (TrustED '13)*, pages 55–64. ACM, 2013.

[51] Nikolaos Athanasios Anagnostopoulos, Stefan Katzenbeisser, Markus Rosenstihl, Andr Schaller, Sebastian Gabmeyer, and Tolga Arul. Low-temperature data remanence attacks against intrinsic sram pufs. Technical report, 2016. `https://eprint.iacr.org/2016/769`.

[52] Z. Liao, G. T. Amariucai, R. K. Wong, and Y. Guan. The impact of discharge inversion effect on learning SRAM power-up statistics. In *Proceedings of the 2017 Asian Hardware Oriented Security and Trust Symposium (AsianHOST)*, pages 31–36. IEEE, 2017.

[53] R. Maes and V. van der Leest. Countering the effects of silicon aging on SRAM PUFs. In *Proceedings of the 2014 IEEE International Symposium on Hardware-Oriented Security and Trust (HOST)*, pages 148–153. IEEE, 2014.

[54] D. R. Raymond, R. C. Marchany, and S. F. Midkiff. Scalable, cluster-based anti-replay protection for wireless sensor networks. In *Proceedings of the IEEE SMC Information Assurance and Security Workshop (IAW '07)*, pages 127–134. IEEE, 2007.

[55] IEEE Standard 802.15.4-2003, 2003.

[56] IEEE Standard 802.15.4-2006, 2006.

[57] IEEE Standard 802.15.4-2011, 2011.

[58] Naveen Sastry and David 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.

[59] K. B. Hein, L. B. Hörmann, and R. Weiss. Using a leaky bucket counter as an advanced threshold mechanism for event detection in wireless sensor networks. In *Proceedings of the Tenth Workshop on Intelligent Solutions in Embedded Systems (WISES)*, pages 51–56. IEEE, 2012.

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

[61] Federico Ferrari, Marco Zimmerling, Luca Mottola, and Lothar Thiele. Low-power wireless bus. In *Proceedings of the 10th ACM Conference on Embedded Network Sensor Systems (SenSys '12)*, pages 1–14. ACM, 2012.

[62] IEEE 802.15.10-2017 - IEEE Recommended Practice for Routing Packets in IEEE 802.15.4 Dynamically Changing Wireless Networks, 2017.

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

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

[65] ADVANCED ENCRYPTION STANDARD (AES), 2001. Federal Information Processing Standards Publication 197.

[66] LoRaWAN 1.1 Specification. LoRa Alliance, 2017.

[67] NARROWBAND IOT BAHNBRECHEND FüR DAS INTERNET DER DINGE. Deutsche Telekom, 2017.

[68] S. Farrell. Low-Power Wide Area Network (LPWAN) Overview. RFC 8376, 2018.

[69] J. Muoz, E. Riou, X. Vilajosana, P. Muhlethaler, and T. Watteyne. Overview of IEEE802.15.4g OFDM and its applicability to smart building applications. In *Proceedings of the 2018 Wireless Days (WD)*, pages 123–130. IEEE, 2018.

[70] CEPT Recommendation ERC/REC 70-03, 2015.

[71] F. Adelantado, X. Vilajosana, P. Tuset-Peiro, B. Martinez, J. Melia-Segui, and T. Watteyne. Understanding the limits of LoRaWAN. *IEEE Communications Magazine*, 55(9):34–40, 2017.

[72] A. Wood, J.A. Stankovic, and Gang Zhou. DEEJAM: defeating energy-efficient jamming in IEEE 802.15.4-based wireless networks. In *Proceedings of the 4th Annual IEEE Communications Society Conference on Sensor, Mesh and Ad Hoc Communications and Networks (SECON '07)*, pages 60–69. IEEE, 2007.

[73] Pei Huang, Li Xiao, S. Soltani, M.W. Mutka, and Ning Xi. The evolution of MAC protocols in wireless sensor networks: a survey. *IEEE Communications Surveys Tutorials*, 15(1):101–120, 2013.

[74] Gang Li and Robin Doss. Energy-efficient medium access control in wireless sensor networks. In *Guide to Wireless Sensor Networks*, pages 419–438. Springer, 2009.

[75] Amre El-Hoiydi and Jean-Dominique Decotignie. WiseMAC: an ultra low power MAC protocol for multi-hop wireless sensor networks. In *Proceedings of the First International Workshop on Algorithmic Aspects of Wireless Sensor Networks (ALGOSENSORS 2004)*, pages 18–31. Springer, 2004.

[76] David Moss and Philip Levis. BoX-MACs: exploiting physical and link layer boundaries in low-power networking. Technical Report SING-08-00, Stanford University, 2008.

[77] Heping Wang, Xiaobo Zhang, Farid Naït-Abdesselam, and Ashfaq Khokhar. DPS-MAC: an asynchronous MAC protocol for wireless sensor networks. In *Proceedings of the International Conference on High Performance Computing (HiPC 2007)*, pages 393–404. Springer, 2007.

[78] Sha Liu, Kai-Wei Fan, and Prasun Sinha. CMAC: an energy-efficient MAC layer protocol using convergent packet forwarding for wireless sensor networks. *ACM Transactions on Sensor Networks*, 5(4):29:1–29:34, 2009.

[79] Simon Duquennoy, Fredrik Österlind, and Adam Dunkels. Lossy links, low power, high throughput. In *Proceedings of the 9th ACM Conference on Embedded Networked Sensor Systems (SenSys '11)*, pages 12–25. ACM, 2011.

[80] Simon Duquennoy, Olaf Landsiedel, and Thiemo Voigt. Let the tree bloom: scalable opportunistic routing with ORPL. In *Proceedings of the 11th ACM Conference on Embedded Networked Sensor Systems (SenSys '13)*, pages 2:1–2:14. ACM, 2013.

[81] Mo Sha, Gregory Hackmann, and Chenyang Lu. Energy-efficient low power listening for wireless sensor networks in noisy environments. In *Proceedings of the 12th International Conference on Information Processing in Sensor Networks (IPSN '13)*, pages 277–288. ACM, 2013.

[82] G. Z. Papadopoulos, A. Gallais, T. Noel, V. Kotsiou, and P. Chatzimisios. Enhancing ContikiMAC for bursty traffic in mobile sensor networks. In *Proceedings of IEEE SENSORS 2014*, pages 257–260. IEEE, 2014.

[83] B. Al Nahas, S. Duquennoy, V. Iyer, and T. Voigt. Low-power listening goes multi-channel. In *Proceedings of the 2014 IEEE International Conference on Distributed Computing in Sensor Systems (DCOSS)*, pages 2–9. IEEE, 2014.

[84] Badis Djamaa and Mark Richardson. Improved broadcast communication in radio duty-cycled networks. Technical report, 2015.

[85] S. S. Guclu, T. Ozcelebi, and J. J. Lukkien. Improving broadcast performance of radio duty-cycled internet-of-things devices. In *Proceedings of IEEE Global Communications Conference (GLOBECOM)*, pages 1–7. IEEE, 2016.

[86] M. F. Youssef, K. M. F. Elsayed, and A. H. Zahran. Contiki-AMAC: the enhanced adaptive radio duty cycling protocol: proposal and analysis. In *Proceedings of the International Conference on Selected Topics in Mobile Wireless Networking (MoWNeT)*, pages 1–6. IEEE, 2016.

[87] S. Homayouni and R. Javidan. ERA-ContikiMAC: an adaptive radio duty cycling layer in Internet of things. In *Proceedings of the 2018 9th International Symposium on Telecommunications (IST)*, pages 74–79. IEEE, 2018.

[88] K Pister and Lance Doherty. TSMP: time synchronized mesh protocol. *IASTED Distributed Sensor Networks*, pages 391–398, 2008.

[89] X. Vilajosana, K. Pister, and T. Watteyne. Minimal IPv6 over the TSCH Mode of IEEE 802.15.4e (6TiSCH) Configuration . RFC 8180, 2017.

[90] Tengfei Chang, Thomas Watteyne, Kris Pister, and Qin Wang. Adaptive synchronization in multi-hop TSCH networks. *Computer and Telecommunications Networking*, 76(C):165–176, 2015.

[91] Tengfei Chang, Thomas Watteyne, Kris Pister, and Qin Wang. Adaptive compensation for time-slotted synchronization in wireless sensor network. *International Journal of Distributed Sensor Networks*, 2014.

[92] D. Stanislowski, X. Vilajosana, Qin Wang, T. Watteyne, and K.S.J. Pister. Adaptive synchronization in IEEE802.15.4e networks. *IEEE Transactions on Industrial Informatics*, 10(1):795–802, 2014.

[93] F. Ferrari, M. Zimmerling, L. Thiele, and O. Saukh. Efficient network flooding and time synchronization with glossy. In *Proceedings of the 10th ACM/IEEE International Conference on Information Processing in Sensor Networks (IPSN 2011)*, pages 73–84. IEEE, 2011.

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

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

[96] Z. Shelby, K. Hartke, and C. Bormann. The Constrained Application Protocol (CoAP). RFC 7252, 2014.

[97] M.R. Palattella, N. Accettura, X. Vilajosana, T. Watteyne, L.A Grieco, G. Boggia, and M. Dohler. Standardized protocol stack for the internet of (important) things. *Communications Surveys Tutorials*, 15(3):1389–1406, 2013.

[98] D. Thaler. Privacy Considerations for IPv6 Adaptation-Layer Mechanisms. RFC 8065, 2017.

[99] C. Bormann and P. Hoffman. Concise Binary Object Representation (CBOR). RFC 7049, 2013.

[100] K. Hartke. Observing Resources in the Constrained Application Protocol (CoAP). RFC 7641, 2015.

[101] C. Bormann and Z. Shelby. Block-Wise Transfers in the Constrained Application Protocol (CoAP). RFC 7959, 2016.

[102] Shahid Raza, Simon Duquennoy, Joel Höglund, Utz Roedig, and Thiemo Voigt. Secure communication for the Internet of Things - a comparison of link-layer security and IPsec for 6LoWPAN. *Security and Communication Networks*, 2012.

[103] E. Rescorla and N. Modadugu. Datagram Transport Layer Security Version 1.2. RFC 6347, 2012.

[104] G. Selander, J. Mattsson, F. Palombini, and L. Seitz. Object Security for Constrained RESTful Environments (OSCORE). Draft, 2018.

[105] René Hummen, Jens Hiller, Hanno Wirtz, Martin Henze, Hossein Shafagh, and Klaus Wehrle. 6LoWPAN fragmentation attacks and mitigation mechanisms. In *Proceedings of the Sixth ACM Conference on Security and Privacy in Wireless and Mobile Networks (WiSec '13)*, pages 55–66. ACM, 2013.

[106] Mahmud Hossain, Yasser Karim, and Ragib Hasan. SecuPAN: a security scheme to mitigate fragmentation-based network attacks in 6LoWPAN. In *Proceedings of the Eighth ACM Conference on Data and Application Security and Privacy (CODASPY '18)*, pages 307–318. ACM, 2018.

[107] Anthéa Mayzaud, Rémi Badonnel, and Isabelle Chrisment. A taxonomy of attacks in rpl-based internet of things. *International Journal of Network Security*, 18(3):459–473, 2016.

[108] Jing Deng, Richard Han, and Shivakant Mishra. Defending against path-based DoS attacks in wireless sensor networks. In *Proceedings of the 3rd ACM Workshop on Security of Ad Hoc and Sensor Networks (SASN '05)*, pages 89–96. ACM, 2005.

[109] 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.

[110] Emmanuel Baccelli, Oliver Hahm, Matthias Wählisch, Mesut Günes, and Thomas Schmidt. RIOT: one OS to rule them all in the IoT. Technical Report RR-8176, INRIA, 2012.

[111] Adam Dunkels, Oliver Schmidt, Thiemo Voigt, and Muneeb Ali. Protothreads: Simplifying event-driven programming of memory-constrained embedded systems. In *Proceedings of the 4th International Conference on Embedded Networked Sensor Systems (SenSys '06)*, pages 29–42. ACM, 2006.

[112] H. Will, K. Schleiser, and J. Schiller. A real-time kernel for wireless sensor networks employed in rescue scenarios. In *Proceedings of the 34th Conference on Local Computer Networks (LCN 2009)*, pages 834–841. IEEE, 2009.

[113] Martine Lenders, Peter Kietzmann, Oliver Hahm, Hauke Petersen, Cenk Gündoğan, Emmanuel Baccelli, Kaspar Schleiser, Thomas C Schmidt, and Matthias Wählisch. Connecting the world of embedded mobiles: The riot approach to ubiquitous networking for the internet of things. *arXiv preprint arXiv:1801.02833*, 2018.

[114] Ming Ki Chong, Rene Mayrhofer, and Hans Gellersen. A survey of user interaction for spontaneous device association. *ACM Computing Surveys*, 47(1):8:1–8:40, 2014.

[115] Deji Chen, Mark Nixon, Thomas Lin, Song Han, Xiuming Zhu, Aloysius Mok, Roger Xu, Julia Deng, and An Liu. Over the air provisioning of industrial wireless devices using elliptic curve cryptography. In *Proceedings of the 2011 IEEE International Conference on Computer Science and Automation Engineering (CSAE)*, volume 2, pages 594–600. IEEE, 2011.

[116] Nitesh Saxena and Md.Borhan Uddin. Blink 'em all: scalable, user-friendly and secure initialization of wireless sensor nodes. In *Proceedings of the 8th International Conference on Cryptology and Network Security (CANS 2009)*, pages 154–173. Springer, 2009.

[117] Nicolas Tsiftes, Adam Dunkels, Zhitao He, and Thiemo 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.

[118] Rabia Latif and Mukhtar Hussain. Hardware-based random number generation in wireless sensor networks (WSNs). In *Proceedings of the Third International Conference on Advances in Information Security and Assurance (ISA 2009)*, pages 732–740. Springer, 2009.

[119] A. Francillon and C. Castelluccia. TinyRNG: a cryptographic random number generator for wireless sensors network nodes. In *Proceedings of the 5th International Symposium on Modeling and Optimization in Mobile, Ad Hoc and Wireless Networks (WiOpt 2007)*, pages 1–7. IEEE, 2007.

[120] C. Hennebert, H. Hossayni, and C. Lauradoux. The entropy of wireless statistics. In *Proceedings of the 2014 European Conference on Networks and Communications (EuCNC)*, pages 1–5. IEEE, 2014.

[121] Christine Hennebert, Hicham Hossayni, and Cédric Lauradoux. Entropy harvesting from physical sensors. In *Proceedings of the Sixth ACM Conference on Security and Privacy in Wireless and Mobile Networks (WiSec '13)*, pages 149–154. ACM, 2013.

[122] Jonathan Voris, Nitesh Saxena, and Tzipora Halevi. Accelerometers and randomness: Perfect together. In *Proceedings of the Fourth ACM Conference on Wireless Network Security (WiSec '11)*, pages 115–126. ACM, 2011.

[123] NG Bardis, AP Markovskyi, N Doukas, and NV Karadimas. True random number generation based on environmental noise measurements for military applications. In *Proceedings of the 8th WSEAS International*

Conference on Signal Processing, Robotics and Automation, pages 68–73, 2009.

[124] *Random number generation using the MSP430*. Texas Instruments. `http://www.ti.com/lit/an/slaa338/slaa338.pdf`.

[125] K. Mowery, M. Wei, D. Kohlbrenner, H. Shacham, and S. Swanson. Welcome to the entropics: boot-time entropy in embedded devices. In *Proceedings of the 2013 IEEE Symposium on Security and Privacy (SP)*, pages 589–603, 2013.

[126] H. Handschuh. Hardware-anchored security based on SRAM PUFs, part 2. *IEEE Security Privacy*, 10(4):80–81, 2012.

[127] Vincent van der Leest, Erik van der Sluis, Geert-Jan Schrijen, Pim Tuyls, and Helena Handschuh. Efficient implementation of true random number generator based on SRAM PUFs. In David Naccache, editor, *Cryptography and Security: From Theory to Applications*, pages 300–318. Springer, 2012.

[128] Peter Kietzmann, Cenk Gündoğan, Thomas C. Schmidt, and Matthias Wählisch. Poster: a PUF seed generator for RIOT: introducing crypto-fundamentals to the wild. In *Proceedings of the 16th Annual International Conference on Mobile Systems, Applications, and Services (MobiSys '18)*, pages 513–513. ACM, 2018.

[129] *High-Performance, Low-Power, 32-Bit Precision32 USB MCU Family with up to 256 kB of Flash Die in Wafer Form*. Silicon Labs. `https://www.silabs.com/documents/public/data-sheets/SiM3U167-B-GDI.pdf`.

[130] Yevgeniy Dodis, Rafail Ostrovsky, Leonid Reyzin, and Adam Smith. Fuzzy extractors: how to generate strong keys from biometrics and other noisy data. *SIAM journal on computing*, 38(1):97–139, 2008.

[131] Yevgeniy Dodis, Rosario Gennaro, Johan Håstad, Hugo Krawczyk, and Tal Rabin. Randomness extraction and key derivation using the CBC, Cascade and HMAC modes. In *Proceedings of the 24th Annual International Cryptology Conference (CRYPTO 2004)*, pages 494–510. Springer, 2004. ISBN 978-3-540-22668-0.

[132] John von Neumann. Various techniques used in connection with random digits. 1951.

[133] Boaz Barak, Russell Impagliazzo, and Avi Wigderson. Extracting randomness using few independent sources. *SIAM Journal on Computing*, 36 (4):1095–1118, 2006.

[134] Chia-Jung Lee, Chi-Jen Lu, Shi-Chun Tsai, and Wen-Guey Tzeng. Extracting randomness from multiple independent sources. *IEEE Transactions on Information Theory*, 51(6):2224–2227, 2005.

[135] Boaz Barak, Ronen Shaltiel, and Eran Tromer. True random number generators secure in a changing environment. In *Proceedings of Cryptographic Hardware and Embedded Systems (CHES 2003)*, pages 166–180. Springer, 2003.

[136] Maciej Skorski. True random number generators secure in a changing environment: Improved security bounds. In *Proceedings of the 41st International Conference on Current Trends in Theory and Practice of Computer Science (SOFSEM 2015)*, pages 590–602. Springer, 2015.

[137] Hwajeong Seo, Jongseok Choi, Hyunjin Kim, Taehwan Park, and Howon Kim. Pseudo random number generator and hash function for embedded microprocessors. In *Proceedings of the 2014 IEEE World Forum on Internet of Things (WF-IoT)*, pages 37–40. IEEE, 2014.

[138] *2.4 GHz IEEE 802.15.4 / ZigBee-Ready RF Transceiver (Rev. C)*. Texas Instruments. `http://www.ti.com/lit/ds/symlink/cc2420.pdf`.

[139] Y. Yan, E. Oswald, and T. Tryfonas. Cryptographic randomness on a CC2538: a case study. In *Proceedings of the 2016 IEEE International Workshop on Information Forensics and Security (WIFS)*, pages 1–6. IEEE, 2016.

[140] Jorge Guajardo, Sandeep S. Kumar, Geert-Jan Schrijen, and Pim Tuyls. FPGA intrinsic PUFs and their use for IP protection. In *Proceedings of Cryptographic Hardware and Embedded Systems (CHES 2007)*, pages 63–80. Springer, 2007.

[141] H. Handschuh. Hardware-anchored security based on sram pufs, part 1. *IEEE Security Privacy*, 10(3):80–83, 2012.

[142] G. Schrijen and V. van der Leest. Comparative analysis of SRAM memories used as PUF primitives. In *Proceedings of the 2012 Design, Automation Test in Europe Conference Exhibition (DATE)*. IEEE, 2012.

[143] Elaine Barker and John Kelsey. Recommendation for the Entropy Sources Used for Random Bit Generation, 2012. NIST DRAFT Special Publication 800-90B.

[144] Xavier Vilajosana, Pere Tuset, Thomas Watteyne, and Kris Pister. OpenMote: open-source prototyping platform for the industrial IoT. In *Ad Hoc Networks*, volume 155, pages 211–222. Springer, 2015.

[145] Yee Wei Law and Maimuthu Palaniswami. Key management in wireless sensor networks. In *Guide to Wireless Sensor Networks*, pages 513–531. Springer, 2009.

[146] Hon Sun Chiu and King-Shan Lui. DelPHI: wormhole detection mechanism for ad hoc wireless networks. In *Proceedings of the 1st International Symposium on Wireless Pervasive Computing*, pages 6–11. IEEE, 2006.

[147] Mark Luk, Ghita Mezzour, Adrian Perrig, and Virgil 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.

[148] X. Cao, D. M. Shila, Y. Cheng, Z. Yang, Y. Zhou, and J. Chen. Ghost-in-ZigBee: energy depletion attack on ZigBee-based wireless networks. *IEEE Internet of Things Journal*, 3(5):816–829, 2016.

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

[150] Chi-Yuan Chen and Han-Chieh Chao. A survey of key distribution in wireless sensor networks. *Security and Communication Networks*, 2011.

[151] Haowen Chan, Adrian Perrig, and Dawn Song. Random key predistribution schemes for sensor networks. In *Proceedings of the 2003 IEEE Symposium on Security and Privacy*, pages 197–213. IEEE, 2003.

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

[153] Wenliang Du, Jing Deng, Yunghsiang S. Han, Pramod K. Varshney, Jonathan Katz, and Aram 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.

[154] Cristina Alcaraz, Javier Lopez, Rodrigo Roman, and Hsiao-Hwa Chen. Selecting key management schemes for WSN applications. *Computers & Security*, 31(38):956–966, 2012.

[155] 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.

[156] Saurabh Ganeriwal, Christina Pöpper, Srdjan Capkun, and Mani B. Srivastava. Secure time synchronization in sensor networks. *ACM Transactions on Information and System Security (TISSEC)*, 11(4):23:1–23:35, 2008.

[157] Sencun Zhu, Sanjeev Setia, and Sushil Jajodia. LEAP: efficient security mechanisms for large-scale distributed sensor networks. In *Proceedings of the 10th ACM Conference on Computer and Communications Security (CCS '03)*, pages 62–72. ACM, 2003.

[158] Adam Dunkels, Fredrik Osterlind, Nicolas Tsiftes, and Zhitao 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.

[159] Joakim Eriksson, Fredrik Österlind, Niclas Finne, Nicolas Tsiftes, Adam Dunkels, Thiemo Voigt, Robert Sauter, and Pedro José 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.

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

[161] Wei Yang, Qin Wang, Yue Qi, and Shaobo Sun. Time synchronization attacks in IEEE802.15.4e networks. In *Proceedings of the International Conference on Identification, Information and Knowledge in the Internet of Things (IIKI 2014)*, pages 166–169. IEEE, 2014.

[162] Wenyuan Xu, Wade Trappe, Yanyong Zhang, and Timothy Wood. The feasibility of launching and detecting jamming attacks in wireless networks. In *Proceedings of the 6th ACM International Symposium on Mobile Ad Hoc Networking and Computing (MobiHoc '05)*, pages 46–57. ACM, 2005.

[163] Matthias Wilhelm, Ivan Martinovic, Jens B. Schmitt, and Vincent Lenders. Short paper: Reactive jamming in wireless networks: How realistic is the threat? In *Proceedings of the Fourth ACM Conference on Wireless Network Security (WiSec '11)*, pages 47–52. ACM, 2011.

[164] Aurélien Francillon, Boris Danev, and Srdjan Capkun. Relay attacks on passive keyless entry and start systems in modern cars. In *Proceedings of the Network and Distributed System Security Symposium (NDSS)*, 2011.

[165] Daniel Dinu and Ilya Kizhvatov. EM analysis in the IoT context: lessons learned from an attack on Thread. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, pages 73–97, 2018.

[166] Ross Anderson and Markus Kuhn. Tamper resistance - a cautionary note. In *Proceedings of the Second USENIX Workshop on Electronic Commerce (WOEC'96)*, volume 2, pages 1–11. USENIX, 1996.

[167] Mathieu Michel and Bruno Quoitin. ContikiMAC vs X-MAC performance analysis. Technical report, 2014. `http://arxiv.org/abs/1404.3589`.

[168] Maite Bezunartea, Mai Banh, Miguel Gamallo, Jacques Tiberghien, and Kris Steenhaut. Impact of cross-layer interactions between radio duty cycling and routing on the efficiency of a wireless sensor network: a testbed study involving ContikiMAC and RPL. In *Proceedings of the Second International Conference on Internet of Things, Data and Cloud Computing (ICC '17)*, pages 1–7. ACM, 2017.

[169] Zhitao He and T. Voigt. Droplet: a new denial-of-service attack on low power wireless sensor networks. In *Proceedings of the 2013 IEEE 10th International Conference on Mobile Ad-Hoc and Sensor Systems (MASS 2013)*, pages 542–550. IEEE, 2013.

[170] R. Struik. 6TiSCH Security Architectural Considerations. IETF Draft Version 1, 2015.

[171] Mario Strasser, Boris Danev, and Srdjan Capkun. Detection of reactive jamming in sensor networks. *ACM Transactions on Sensor Networks (TOSN)*, 7(2):16:1–16:29, 2010.

[172] Michael Buettner, Gary V. Yee, Eric Anderson, and Richard Han. X-MAC: a short preamble MAC protocol for duty-cycled wireless sensor networks. In *Proceedings of the 4th International Conference on Embedded Networked Sensor Systems (SenSys '06)*, pages 307–320. ACM, 2006.

[173] Prabal Dutta, Stephen Dawson-Haggerty, Yin Chen, Chieh-Jan Mike Liang, and Andreas Terzis. Design and evaluation of a versatile and efficient receiver-initiated link layer for low-power wireless. In *Proceedings of the 8th ACM Conference on Embedded Networked Sensor Systems (SenSys '10)*, pages 1–14. ACM, 2010.

[174] David R. Raymond and S.F. Midkiff. Clustered adaptive rate limiting: defeating denial-of-sleep attacks in wireless sensor networks. In *Proceedings of the Military Communications Conference (MILCOM 2007)*, pages 1–7. IEEE, 2007.

[175] Sencun Zhu, S. Setia, S. Jajodia, and Peng Ning. An interleaved hop-by-hop authentication scheme for filtering of injected false data in sensor networks. In *Proceedings of the 2004 IEEE Symposium on Security and Privacy (S&P '04)*, pages 259–271. IEEE, 2004.

[176] Fan Ye, H. Luo, Songwu Lu, and Lixia Zhang. Statistical en-route filtering of injected false data in sensor networks. *IEEE Journal on Selected Areas in Communications*, 23(4):839–850, 2005.

[177] Hui Song, Sencun Zhu, and Guohong Cao. Attack-resilient time synchronization for wireless sensor networks. In *Proceedings of the 2005 IEEE International Conference on Mobile Adhoc and Sensor Systems (MASS 2005)*. IEEE, 2005.

[178] Mohamed G. Gouda, Young ri Choi, and Anish Arora. Antireplay protocols for sensor networks. In Jie Wu, editor, *Handbook on Theoretical and Algorithmic Aspects of Sensor, Ad Hoc Wireless, and Peer-to-Peer Networks*, pages 561–574. CRC, 2005.

[179] Phillip Rogaway and David A Wagner. A Critique of CCM. *IACR Cryptology ePrint Archive*, 2003:70, 2003.

[180] T. Tsao, R. Alexander, M. Dohler, V. Daza, A. Lozano, and M. Richardson. A Security Threat Analysis for the Routing Protocol for Low-Power and Lossy Networks (RPLs). RFC 7416, 2015.

[181] Marie Paule Uwase, Maite Bezunartea, Jacques Tiberghien, Jean-Michel Dricot, and Kris Steenhaut. Poster: ContikiMAC, some critical issues with the CC2420 radio. In *Proceedings of the 2016 International Conference on Embedded Wireless Systems and Networks (EWSN '16)*, pages 257–258. Junction, 2016.

[182] Mathieu Michel, Thiemo Voigt, Luca Mottola, Nicolas Tsiftes, and Bruno Quoitin. Predictable MAC-level performance in low-power wireless under interference. In *Proceedings of the 2016 International Conference on Embedded Wireless Systems and Networks (EWSN '16)*, pages 13–22. Junction, 2016.

[183] V. Srivastava and M. Motani. Cross-layer design: a survey and the road ahead. *IEEE Communications Magazine*, 43(12):112–119, 2005.

[184] S. S. Guclu, T. Ozcelebi, and J. J. Lukkien. Dependability analysis of asynchronous radio duty cycling protocols. In *Proceedings of the 25th International Conference on Computer Communication and Networks (ICCN)*, pages 1–11. IEEE, 2016.

[185] Maite Bezunartea, Benjamin Sartori, Jacques Tiberghien, and Kris Steenhaut. Tackling malfunctions caused by radio duty cycling protocols that do not appear in simulation studies. In *Proceedings of the First ACM International Workshop on the Engineering of Reliable, Robust, and Secure Embedded Wireless Sensing Systems (FAILSAFE'17)*, pages 10–15. ACM, 2017.

[186] Andrew Tinka, Thomas Watteyne, Kristofer S. J. Pister, and Alexandre M. Bayen. A decentralized scheduling algorithm for time synchronized channel hopping. *EAI Endorsed Transactions on Mobile Communications and Applications*, 11(1), 2011.

[187] Joakim Eriksson, Niclas Finne, Nicolas Tsiftes, Simon Duquennoy, and Thiemo Voigt. Scaling RPL to dense and large networks with constrained memory. In *Proceedings of the International Conference on Embedded Wireless Systems and Networks (EWSN 2018).* Junction, 2018.

[188] Eirik Klevstad. Security and key establishment in IEEE 802.15.4. Master's thesis, 2016. Norwegian University of Science and Technology.

[189] Daniel Werner. Key revocation and rekeying for the adaptive key establishment scheme. Master's thesis, 2018. Hasso Plattner Institute, University of Potsdam.

[190] Felix Wolff. Denial-of-sleep-resilient opportunistic routing for 802.15.4 networks. Master's thesis, 2017. Hasso Plattner Institute, University of Potsdam.

[191] Luiz Henrique Andrade Correia, Daniel Fernandes Macedo, Aldri Luiz dos Santos, and Josè Marcos Silva Nogueira. Transmission power control techniques in ad hoc networks. In *Guide to Wireless Sensor Networks*, pages 469–490. Springer, 2009.

[192] W. Rukpakavong, I Phillips, and Lin Guan. Neighbour discovery for transmit power adjustment in IEEE 802.15.4 using RSSI. In *Proceedings of the 4th IFIP International Conference on New Technologies, Mobility and Security (NTMS)*, pages 1–4. IEEE, 2011.

[193] Hiba Dakdouk, Erika Tarazona, Reda Alami, Raphaël Féraud, Georgios Z. Papadopoulos, and Patrick Maillé. Reinforcement learning techniques for optimized channel hopping in IEEE 802.15.4-TSCH networks. In *Proceedings of the 21st ACM International Conference on Modeling, Analysis and Simulation of Wireless and Mobile Systems (MSWIM '18)*, pages 99–107. ACM, 2018.

[194] Jing Deng, Richard Han, and Shivakant Mishra. INSENS: intrusion-tolerant routing for wireless sensor networks. *Computer Communications*, 29(2):216 – 230, 2006.

List of Figures

List of Tables

Glossary

6LoWPAN IPv6 over LoWPANs.

6LoWPAN-ND 6LoWPAN neighbor discovery.

ACK Acknowledgment message.

AEDP Adaptive Energy Detection Protocol.

AKES Adaptive Key Establishment Scheme.

APKES Adaptable Pairwise Key Establishment Scheme.

BLE Bluetooth low energy.

C-SGN cellular IoT serving gateway node.

CARL Clustered Adaptive Rate Limiting.

CBC cipher block chaining.

CBOR Concise Binary Object Representation.

CCA clear channel assessment.

CCM Counter with CBC-MAC.

CoAP Constrained Application Protocol.

CON Confirmable messsage.

CSL coordinated sampled listening.

CSMA-CA carrier sense multiple access with collision avoidance.

CSPRNG cryptographically-secure pseudo-random number generator.

CTR counter mode.

DIO DODAG information object.

DODAG destination-oriented directed acyclic graph.

DRAM dynamic RAM.

DTLS Datagram Transport Layer Security.

EBEAP Easy Broadcast Encryption and Authentication Protocol.

ECDSA Elliptic Curve Digital Signature Algorithm.

eNB evolved node B.

FCS frame check sequence.

FFD full-function device.

FSK frequency shift keying.

GNRC Generic.

HSS home subscriber service.

HTTP Hypertext Transfer Protocol.

i.i.d. mutually independent and identically distributed.

ICMPv6 Internet control message protocol version 6.

IE information element.

IETF Internet engineering task force.

IID interface identifier.

ILOS Intra-Layer Optimization for IEEE 802.15.4 Security.

INSENS INtrusion-tolerant routing protocol for wireless SEnsor NetworkS.

IoT Internet of things.

IOWEU inoculation, opaqueness, welcomingness, efficiency, and universality.

ISR interrupt service routine.

KDC key distribution center.

KDF key derivation function.

LB last bits.

LBC leaky bucket counter.

LoRa long range.

LoRaWAN long range wide area network.

LoWPAN low-power wireless personal area network.

LPM low-power mode.

LPWAN low-power wide-area network.

LQI link quality indicator.

LSB least significant bit.

LTE long term evolution.

LWB low-power wireless bus.

MAC medium access control.

MIC message integrity code.

NB-IoT NarrowBand-IoT.

NON Non-confirmable message.

O-QPSK offset quadrature phase-shift keying.

OFB output feedback mode.

OFDM orthogonal frequency-division multiplexing.

ORPL opportunistic RPL.

OSCORE Object Security for Constrained RESTful Environments.

OTP one-time password.

PAN personal area network.

PDoS path-based denial-of-service.

PHY physical layer.

PKC public-key cryptography.

POTR Practical On-the-fly Rejection.

PUF physically unclonable function.

RAM random-access memory.

RFC request for comments.

RFD reduced-function device.

RPL IPv6 routing protocol for low-power and lossy networks.

RSSI received signal strength indicator.

RST Reset message.

SCEF service capability exposure function.

SCREWED Secure Channel REciprocity-based WormholE Detection.

SDR software-defined radio.

SHR synchronization header.

SIM subscriber identity module.

SMA SubMiniature version A.

SoC system on chip.

SPLO Secure Phase-Lock Optimization.

SPS Secure Pairwise Synchronization Protocol.

SRAM static random-access memory.

SRD short range device.

SUN smart utility network.

TCP Transmission Control Protocol.

TSCH time-slotted channel hopping.

UDP user datagram protocol.

UE user equipment.

Selbstständigkeitserklärung

Hiermit erkläre ich, dass ich diese Arbeit selbständig verfasst und keine anderen als die von mir angegebenen Hilfsmittel genutzt habe.