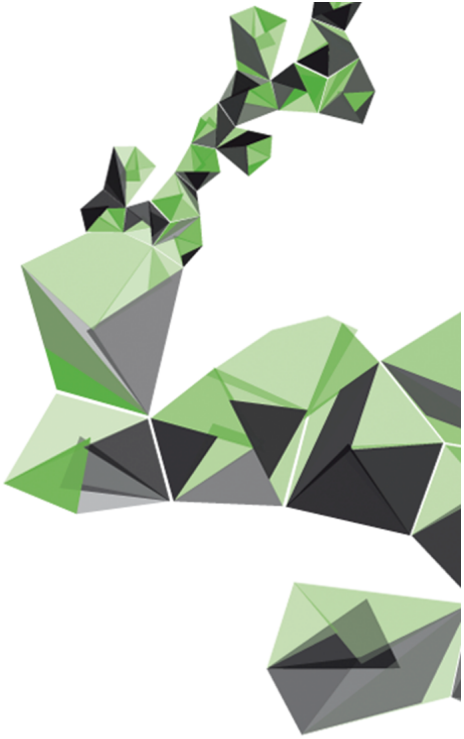
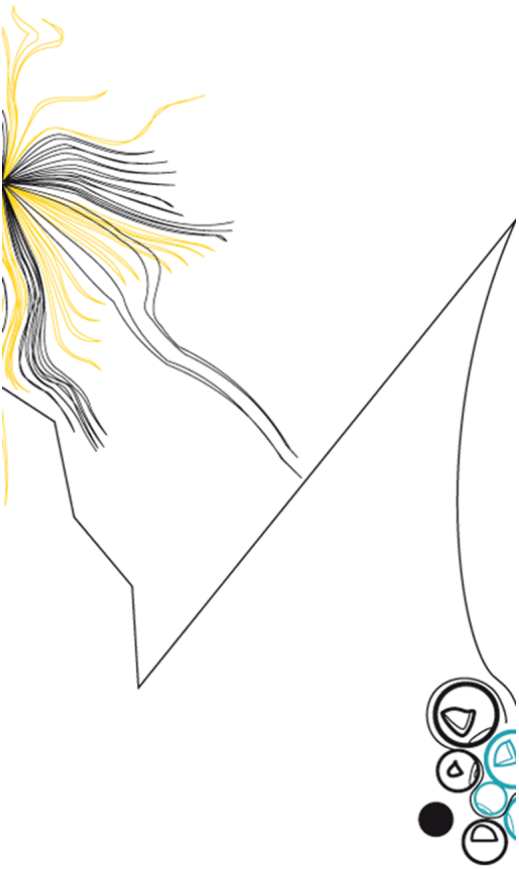


UNIVERSITY OF TWENTE.

Faculty of Electrical Engineering,
Mathematics & Computer Science

TNO

Assessing smart home security: a Zigbee case study



Lennart Jansen
M.Sc. Thesis
Computer Science
Cyber Security (4TU)
January 10th, 2022

Supervisors:
Dr. Mattijs Jonker (University of Twente)
Jan Kazemier MSc. (TNO)

Graduation committee chair:
Prof. Dr. Ir. Roland van Rijswijk-Deij (University of Twente)

Graduation committee member:
Dr.-Ing. Erik Tews (University of Twente)

Abstract

The popularity of smart home products keeps increasing and so does their variety. As more and more brands offer their own products, they are competing with each other not just with quality, but also price-wise. The Lidl Smart Home (LSH) product line offers fully connected devices that users may use to secure their homes against intruders. The products use ZigBee 3.0 as communication protocol, which is based on the IEEE 802.15.4 [6][11] standard. ZigBee has been shown to be insecure in the past [14][28][30][31] and in the latest specification [11] old vulnerabilities still exist as new security features are optional.

In this case study, we analyze the LSH product line concerning its privacy-friendliness and security. We show that users are dependent on multiple cloud service providers to fully use the network's functionalities as there is no feasible option to use the products offline. As a result, these service providers have a full picture of the active devices and the user's behavior, which is a great loss of privacy. Our security analysis shows that an attacker listening to the network traffic while a device is being added to the ZigBee network gets to know all keying material necessary to decrypt and read any traffic sent in the network. This also enables the attacker to arbitrarily control any device within the network or make them unusable until the user manually resets them. As a result, an attacker, such as a burglar, may determine whether a user is at home by looking at the network activity, turn off the lights to remain unrecognized by potential video surveillance and deactivate the motion sensor to prevent the user from getting a push notification of a possible breach. We do therefore conclude that the LSH products are not secure and should not be used for security-relevant purposes.

We describe recommendations to improve the privacy-friendliness and security of the LSH product line. To improve the user's privacy, we recommend enabling offline usage of the network, that is, users should be able to configure the network and control devices from within a local network that is not connected to the internet. To guarantee a secure pairing of devices without leaking any encryption keys to a passive adversary, we recommend the use of out-of-bound install codes, which are defined in the ZigBee standard as an optional security measure. We recommend the implementation of periodic key rotation to protect against attackers that have already obtained an encryption key. We show that the compromise of one device leads to the compromise of the entire network. To mitigate this, we recommend the use of end-to-end encryption keys instead of one key for all devices.

Contents

List of Tables

List of Figures

Acronyms

6LoWPAN	IPv6 over Low-Power Wireless Personal Area Networks
ACK	Acknowledgement
AES	Advanced Encryption Standard, *see Section 5.4*
AP	Access Point
APL	Application Layer
APS	Application Support Layer
BE	Big Endian
BPSK	Binary Phase-Shift Keying
CBC	Cipher Block Chaining
CCM	Counter with CBC-MAC
CIA	Confidentiality, Integrity, Availability
CRC	Cyclic Redundancy Check
DHCP	Dynamic Host Configuration Protocol
DoS	Denial of Service
ECDH	Elliptic Curve Diffie-Hellman
EPID	Extended PAN ID
FCF	Frame Control Field
FCS	Frame Control Sequence
IAS	Intruder Alarm System
IEEE	Institute of Electrical and Electronics Engineers
IoT	Internet of Things
IP	Internet Protocol
ISP	Internet Service Provider
IV	Initialization Vector
LAN	Local Area Network
LE	Little Endian
LED	Light-Emitting Diode
LSH	Lidl Smart Home
MAC[1]	Media Access Control Layer
MAC[2]	Message Authentication Code
MIC	Message Integrity Code
MMO	Matyas-Meyer-Oseas function, *see Section 5.3*
NWK	Network Layer
OCB	Offset Codebook Mode
OTP	One Time Pad
OQPSK	Offset Quadrature Phase Shift Keying
PAN	Personal Area Network
PAN ID	PAN Identification Number

PHY	Physical Layer
RGB	Red Green Blue
SN	Sequence Number
SSH	Secure Shell
TLS/SSL	Transport Layer Security/Secure Sockets Layer
UDP	User Datagram Protocol
URL	Uniform Resource Locator
USB	Universal Serial Bus
VPN	Virtual Private Network
WLAN	Wireless Local Area Network
WPAN	Wireless Personal Area Network
ZCL	ZigBee Cluster Library
ZDO	ZigBee Device Object

1 Introduction

The home is getting automated. More and more so-called smart home devices are being sold every year [13]. Smart home devices are digitized devices with some form of automation, such as time-based settings or being controlled by another device. Most smart home devices are directly or indirectly connected to the internet, which makes them part of the Internet of Things (IoT). The global market for IoT products is growing exponentially, as IHS showed in a survey commissioned by Forbes [13]. IHS expects that the global number of installed IoT devices will reach 75 billion by 2025 [13].
IoT devices are for example TV's, thermostats, fridges or lamps. Also, devices to secure the home, such as motion sensors, security cameras and doorbells are becoming more popular. With the cameras, a user can watch what is happening around the house. Motion sensors can be used to secure specific areas and warn the user or trigger a floodlight when someone walks by or enters a region in or around the house. 'Smart' doorbells nowadays often have a microphone, speaker and a camera in them, with which it is possible to talk to someone ringing it. Because these products are mostly easy to use and are accessible, users choose to rely more and more on such products to secure their homes against intruders, such as burglars. Therefore, these devices must not only provide convenient features, but users must also be able to rely on them.

Smart home devices are typically very resource-constrained in the sense that they do not feature much computation power or storage space. Many devices are not grid powered but battery-powered, which means that they are not built for high performance, but to consume as little energy as possible. However, to offer the promised functionality, these devices need to be connected to the internet somehow, which is often done via a wireless network, such that no cables have to be laid all around the home. The best-known technology to connect devices is WLAN (also called Wi-Fi), which is the wireless variant of LAN. Other technologies to wirelessly connect devices are for example Bluetooth, 6LoWPAN and ZigBee. All these technologies work differently and have their pros and cons. Due to these implicit resource constraints in the smart home domain connectivity protocols such as Bluetooth Low Energy, 6LoWPAN and ZigBee, are chosen instead of Wi-Fi. Bluetooth is well-known for its use for multimedia sharing between smartphones and devices such as speakers, but it is also a viable option to connect devices such as lamps, switches, power sockets and door locks. 6LoWPAN is a protocol that acts as a layer between the IEEE 802.15.4 standard and IPv6 and thus enables a device to directly connect to the internet. 6LoWPAN is designed for uses cases like healthcare monitoring, home automation and smart cities [7].
ZigBee is a wireless connectivity standard that is created for controlling smart home devices. Just like 6LoWPAN it is based on the IEEE 802.15.4 standard and is therefore suitable for low-energy networks [11]. ZigBee is promoted [29] as very secure as it uses AES-128 encryption, which is considered secure [17]. However, the ZigBee 1.0 standard has severe security flaws which enable an attacker to retrieve encryption keys, which could be used to listen to all commands sent in a network or control devices from a victim network herself by forging malicious packets [14]. In 2015 the ZigBee Alliance published the current ZigBee 3.0 specification[1] [11] which has added options for extra security. However, due to backwards compatibility, vendors are largely free to choose how many of these new security features they want to use. Therefore, the specification as such is still vulnerable, as the old vulnerable parts have not been removed, but merely made optional.
This may be a pitfall for users that choose to use ZigBee enabled devices to secure their homes. If users think they successfully secured their home and property with 'smart' devices, but those devices are actually vulnerable, users have a false sense of security, which is worse than knowing that their home is not secured using such technology.

There are countless different IoT devices on the market from different vendors. Well-known IoT product brands for private in-house use that are sold in the Netherlands are Philips and IKEA. According to Multiscope [45] almost half of the 1.2 million households in the Netherlands that have smart lights,

[1] There have been a few updated versions over the years, but a dedicated 'ZigBee 2.0' never existed.

have lamps from Philips Hue. IKEA has a market share of 23% in smart lights in the Netherlands [45]. Another large vendor of IoT products is Tuya. Tuya is an American-Chinese company that makes IoT products that other companies can sell with their own logo on them [47]. Tuya's retailers in the Netherlands include Lidl, Kruidvat, HEMA and Action [47], out of which Lidl published their products last.
That Lidl is among Tuya's retailers [53] is remarkable, considering that David van Holsteijn, responsible for non-food products at Lidl NL, claims that they have their "own development team and deliberately did not choose a white-label supplier" [46]. This way, they want to "ensure the quality and security" [46] of the products. Regarding the privacy of the users' data, van Holsteijn states that they "work with certified Microsoft Azure servers that are stationed in Europe" [46]. He also claims that Lidl developed the smartphone app themselves, however, a simple look into the different apps (see Figure 1) reveals that their app is yet another Tuya app with slightly adjusted colors and icons. Therefore, we assess Lidl's products further to look at whether critical security mistakes have been made during the implementation of the products.

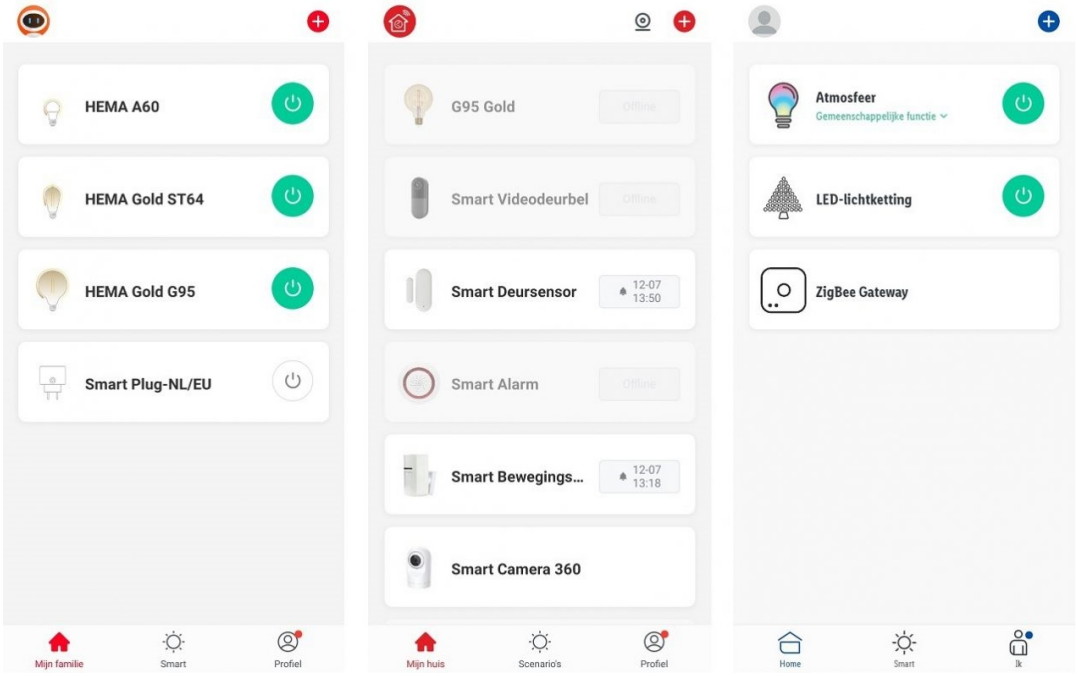


Figure 1: Three 'different' apps for smart home products, by Hema, Kruidvat and Lidl. From: [47]

Additionally to these remarkable statements, we choose to investigate the security of the LSH product line for three reasons: the relatively low pricing, the broad accessibility of the products, and the fact that from all Dutch stores with their own product line Lidl has brought out their products most recently. At the end of 2020, Lidl released its smart home product line in several countries. In the Netherlands, the release date was November 27th, 2020. This new product line, Lidl Smart Home, uses ZigBee 3.0 for its communication between devices. Due to a large number of stores throughout the Netherlands and the very low prices of the products, they are very accessible to a broad public and we expect that the products will be used to secure private homes against intruders. However, there is a possibility that Lidl did not choose to use the new security features from the ZigBee 3.0 specification [11] but instead used the old vulnerable ones because the new features are more expensive to implement and make usage of the products less convenient. In this case study, we test the LSH product series for known vulnerabilities and attacks and also investigate whether new attacks can be found to assess how secure the LSH products are and what consequences their security has for users who use these products to secure their homes.

The rest of this thesis is organized as follows. We define our research goal and present our research questions in Section 2. In Section 3 we explain our methodology. In Section 4 we discuss related work on IoT devices and ZigBee. In Section 5 we provide background information on the Lidl Smart Home products and the ZigBee protocol. We present our results in Section 6 and conclude in Section 7. In Section 9 we propose directions for future work and we state our ethical considerations in Section 10.

2 Problem statement

People want to secure their homes and properties. There are many possible ways to do so, one of which is to install internet-connected security hardware in and around the house such as an alarm system against intruders. The prices of these products vary a lot among different vendors, but there is no clear indication of how secure the products themselves are. The products' security is as important as their functionality, because if the security of the products is compromised, the functionality may be compromised, too [12]. Sivaraman et al. [12] mention that in case of compromised device security, the device's functionality could be used 'against' the user. This could therefore lead to a false sense of security.

As explained in the introduction (see Section 1), from all Dutch stores with their own product line Lidl has brought out their products most recently. The product line, therefore, represents a part of the current state-of-the-art consumer IoT. The statements from van Holsteijn [46], the product's relatively low prices, and the broad accessibility are the reasons we think that the security and privacy-friendliness of the Lidl Smart Home product line need to be assessed. The LSH products can be used for security-critical installations at home to secure it against intruders, such as burglars. This can for example be done by means of an intruder alarm system based on the motion sensor, power sockets that control the power access of some devices, and lamps that shine light into dark places. If any of these devices do not function as expected due to an attack by an adversary, that could have serious implications. If the motion sensor gets deactivated, burglary could become easier without being detected. Often turning off and on a power socket, and therefore the power of a device can lead to serious damage to that device. If lamps are used complementary to security cameras to lighten (otherwise) dark areas, a burglar could target these lamps to arbitrarily turn them off to stay unrecognized on the video imaging.
The LSH products use ZigBee 3.0 for their communication. Although ZigBee 3.0 has significant security updates compared to ZigBee 1.0, many old vulnerabilities remain because the new security features are optional. This allows vendors to get their products ZigBee Certified, although they might be vulnerable to known attacks. The securer functions can be less convenient for both the manufacturer and users. Using install codes, for example, requires the creation of those during manufacturing, both on the device (digitally) and on a physical medium, such as paper, for the user to read. These extra efforts (and thus extra costs) for the manufacturer and the reduced convenience for the user could be a decisive argument for the manufacturer not to use such features. If the devices can not be trusted, the user has a false sense of security, which is harmful. Therefore we investigate how secure these products are by means of a case study.

2.1 Goal

Our goal is to provide insight into how secure and privacy-friendly the LSH products are. As people are likely to use these products to secure their homes against burglars and other intruders, people must be able to rely on the functioning of the devices. This is, however, questionable due to the possible weaknesses in the ZigBee 3.0 specification and the vendor's freedom to choose which security features of that specification they use. The claims [46] that Lidl has their own development team to ensure the security of the products, but we see that the Lidl products are simply Tuya products [53], are another reason to doubt the products' security. We want to get insight into the security of the devices such that we know to what extent and in what situations devices can be manipulated or deactivated by an outside attacker. Testing the network for known and new vulnerabilities results in recommendations for (potential) users, Lidl, and the ZigBee Alliance. Based on our assessment, users can better decide what to use the products (not) for. Lidl and the ZigBee Alliance may use our results to improve their product security to prevent attacks from happening in the future.

2.2 Research questions

To achieve our goal of assessing the security of the LSH products, we define the following research questions. Main research question:

> **How secure are the Lidl Smart Home products from a network perspective?**

This question is meant to help us assess how likely it is that outside attackers, such as burglars, can compromise the security to gain access to the house the products are securing. To further structure the upcoming research, we divide the main research question into the following sub-questions:

1. **What can be learned about the network and its devices by an outside attacker?**
2. **To what extent can device behaviour be controlled by an outside attacker?**
3. **To what extent can an outside attacker prevent the user from normal device operation?**

By answering these sub-questions we aim to study how likely it is that intruders can gain access to a home that is secured with LSH products and what privacy risks the user has by using the products.

3 Methodology

To investigate how secure the LSH devices are, we set up different network topologies with these devices and a phone with the LSH app. We aim at simulating 'normal' domestic use and therefore create different setups of possible security setups that users of the products are likely to use to secure their homes. To test the LSH network on its security, we will use passive sniffing and active packet injection. With sniffing we can learn about the usual traffic in the network and how devices act. This knowledge then enables us to forge our own (possibly malicious) packets, which we can inject to test whether the device behavior can be manipulated. Therefore, with the iterative process of listening and sending messages, we will be able to test how secure the LSH network is.
To structure our research, we use the CIA model, which stands for Confidentiality, Integrity and Availability [20]. We use these security goals to measure how secure a system is. The extent of how far those goals can be infringed upon by an outside attacker tells how secure the network is. If the security goals CIA are met by the LSH network, it can be considered secure, if they are breached, however, it has to be considered insecure.
This section is built up as follows. In Section 3.1 we define the scope of our research. The security goals and corresponding methods are further explained in Section 3.2. The specific attacks we perform are described in Section 3.3. In Section 3.4 we describe our setup.
The related work and background are explained later in Sections 4 and 5 respectively.

3.1 Scope

As stated in the main research question (Section 2.2), we focus on the network perspective of the LSH security because we see in Sections 4 and 5.2 that the ZigBee protocol used for wireless network communication has severe weaknesses. The focus is mainly on the security of the (wireless) ZigBee communication by means of the CIA security goals. However, the ethernet traffic between the gateway, phone and servers is investigated as well with regards to its privacy implications for the user towards the services providers. For both the ethernet and ZigBee communication the scope is on Layer 2 (MAC) and upwards (see Section 5.2.3). We do not investigate the hardware or software security of the LSH devices, as did for example Paul Banks [40], who got root access to the LSH gateway by replacing the `/etc/passwd` file with a file that holds a known root password.
We focus our research on the LSH products in small setups, how they can be expected to be built by users in private homes. The implications of large-scale industrial use of the products are not tested as the products will probably not be used in such a setting. However, a relatively simple setup will not guarantee the absence of networking flaws.

3.2 Security goals

For modelling the security threats for the LSH products, we use the CIA model. CIA stands for Confidentiality, Integrity and Availability and it is used to define the security goals [20]. We describe the security goals and give examples in the context of the LSH products. The security threats are implicitly defined as the possible infringements of these security goals.

- **Confidentiality** is about restricting the visibility of information or data to certain entities [20]. For example, only the user of the LSH network should be able to see what devices are used within the network, when they are used, and what their status (color, on/off) is. Confidentiality can for example be infringed upon by an outside attacker by reading encrypted network traffic and thereby learning about the devices within the network without being authorized to do so.

  'Outside attackers' excludes the cloud service providers associated with these products. Although they can read all ethernet traffic, we do not consider that a breach of confidentiality in the technical sense, because they are a part of the product and traffic is therefore by design sent to them. However, as we will see in the results (6.1.1) and the conclusion (7), we do consider this a substantial loss of privacy.

- **Integrity** is about data and devices not being tampered with. Tampering is the act of "modifying a system or user data with or without detection" [2]. For example, only the user should be able to change the state of the LSH devices (color, on/off) and push notifications for the motion sensor and doorbell should only be sent if indeed motion has been detected or the doorbell has been pressed, but not otherwise. Integrity can for example be infringed upon by an outside attacker by changing the state (color, on/off) of a device without being authorized to do so or sending push notifications to the user's phone as if motion was detected or the doorbell pressed, without that actually being the case.

- **Availability** is about the user not being hindered to access or use her devices or services [20]. For example, the user should (under normal circumstances) be able to change the state (color, on/off) of a device to her liking or get a push notification when motion is detected by the motion sensor or the doorbell has been pressed. Availability can for example be infringed upon by an outside attacker by creating an error, for example by sending malicious messages, which results in the device not being usable by the user anymore. An attack on availability is called a Denial of Service (DoS) attack.

3.3 Attack scenarios

In Section 4 we discuss the related work including attacks on the ZigBee protocol, in Section 5.2 we discuss the ZigBee protocol and its weaknesses in greater detail. With this in mind, we formalize different attack scenarios to test the LSH product with regards to the security goals as defined in Section 3.2. As explained at the beginning of this section, the attacks are the result of the iterative process of listening, learning, and attacking. By listening, we learn more about the network and its communication, after which we use that knowledge to try infringing upon the security goals.
Attacks A1 and A2 are sniffing attacks on ethernet and ZigBee traffic respectively. The main goal is to understand how the network works and how devices behave. Attack A3 targets the IEEE 802.15.4 protocol, which ZigBee is based on and aims at retrieving more information on the network. The next step is to try to alter the behavior of the devices. For this, we perform Attacks A4 and A5, which are unencrypted and encrypted command injection attacks respectively. Attack A6 is based on the attack explained by Akestoridis et al. (see Section 4.2) and aims at verifying the existence and practical results of a PAN ID conflict. The purpose of Attack A7 is to learn how much of the network's and devices' functionality can be used if it is not connected to the internet, as that could be privacy-friendlier. Attack A8 aims at verifying results by Farha et al. [31]. With Attack A9 we look at the consequences of integrated functionality (reset buttons) of the devices. Attack A10 aims at preventing normal device operation for the user. By performing Attack A11, we simulate incidental packet loss to analyze the robustness of the network.
Table 1 summarizes all attacks and shows the key knowledge that an attacker is presumed to have at the beginning of the attack, the needed proximity to the network, and the security goal that is mainly attacked.

A1 *Analyze ethernet traffic by sniffing:* this is a passive sniffing attack on the ethernet traffic. By capturing traffic on the self-created Local Area Network using Wireshark, we can analyze where traffic goes and where it is coming from. This way we can analyze what information the different service providers have at their disposal and what privacy concerns might arise out of that. We analyze the traffic manually in Wireshark. We also look into whether the phone and the gateway are communicating directly or only via the cloud. We expect to see that some cloud servers are involved in setting up the network. We also expect to see some direct traffic between the phone and the gateway if they are on the same local network whilst using the LSH App.

A2 *Passively sniff ZigBee traffic:* we listen to the ZigBee traffic that the devices send in different situations. This concerns what packages look like, what kinds of packages are sent regularly or only in specific cases, but also what can be learned about the topology of the network. As

explained in Section 3.4.2, we use the CC2531 to capture packets and use Whsniff to format them for Wireshark, in which we do a manual inspection of the packets. We expect to learn some basic information about the network and the devices in it. We expect to see a star-shaped topology with the gateway at its heart because all devices are within range of one another and there are no dedicated routers among the devices. Because the ZigBee 3.0 specification [11] mentions "a brief moment of vulnerability where the key could be obtained by any device" [11] and we do not see the LSH products supporting the use of Install Codes (see Section 5.3), we expect to be able to learn the NWK Key by listening to the traffic at that vulnerable moment. As we explain in Section 5.2.5, the NWK Key is the 'main' key of the network, as virtually all traffic is encrypted with it.

A3 *Send beacon requests:* given that beacons are packets with no layers above the MAC Layer and therefore use no encryption, we expect to be able to provoke beacon responses by injecting beacon requests. As explained in Section 3.4.3, we send the beacon requests using the Raspberry Pi with the Raspbee hardware module. We listen to the responses using the CC2531. In those beacon responses, we expect to see more detailed information about the devices in the network.

A4 *Unsecured command injection:* ZigBee offers the possibility for well-formatted, but completely unencrypted packets by setting the 'Security' bit in the Frame Control Field to '0'. To test to what extent an attacker can manipulate the behavior of the devices without knowing the NWK Key, we send different unencrypted commands. Because both an attacker and the devices know the Global Key, we also send commands that are encrypted with the publicly known key to test how the devices react to those. We expect the devices to not react to the commands at all, which is, from a security perspective, the desired result.

A5 *Secured command injection:* we know from the ZigBee 3.0 specification [11] that there is a possibility to learn the NWK Key if the attacker listens to the traffic at the right moment. To test whether knowing that key suffices to control devices within the network, we send properly encrypted commands. We expect to be able to control the devices, thus for example turning devices on or off.

A6 *Cause a PAN ID Conflict:* from Akestoridis et al. [28] (see Section 4.2) we know that a PAN ID Conflict can lead to a device being removed from the network. Although we do not have the resources to perform a jamming attack, we expect to be able to create a PAN ID Conflict by injecting a forged malicious beacon. We expect to see that the PAN ID conflict gets resolved by the devices automatically.

A7 *Disconnect gateway from the internet:* for both privacy and security reasons it can be beneficial to have the gateway disconnected from the internet. To test to what extend the LSH devices can be used if the gateway is offline, we disconnect it by simply pulling the ethernet cable. We expect to be able to use the products even if the gateway is offline.

A8 *Turn off and on gateway and end devices:* Farha et al. [31] (see Section 4.2) state that the Security Counter resets when the Trust Center (the gateway) reboots. We turn off and on the gateway and other devices to test whether any counters, the Security Counter in particular (see Section 5.2.4), reset. Given that the paper was published a year before the LSH products were released, this security flaw could already have been fixed. Therefore, we expect that the counters do not reset.

A9 *Disconnect devices with dedicated button:* all LSH end devices but the lamp have a dedicated button for disconnecting the device from the network. We trigger this deauthentication to test what implications it has. This includes traffic that is directly generated by it, but also the traffic when the device is added to the network again afterward, which potentially differs from when the device is added the first time. We expect to see some ZigBee traffic dedicated to signaling the

deauthentication. We also expect that to use the device again, it has to be added to the network as if it has never been a part of it.

A10 *Send high sequence number:* we know from the ZigBee specification [11] that there is a possibility to learn the NWK Key and that sequence numbers are used to prevent replay attacks. By forging a properly encrypted packet with a very high security number, we expect to be able to let the receiving device save this high sequence number such that genuine packets (with a lower sequence number) get dropped and the sending device is effectively taken out of service.

A11 *Move device out of range:* to test the resilience of the network against incidental packet loss as a result of jamming, we move the doorbell out of range of gateway. Due to a lack of the necessary hardware, we are not able to (selectively) jam traffic, but we believe that temporarily taking the device physically out of range simulates this very accurately. We also want to see whether the user is warned about irregularities due to packet loss.

In Table 1 we summarize whether the NWK Key is presumed to be known at the beginning of the attack, what proximity to the network is required, and what security goal is mainly under attack. The meaning of the symbols in Table 1 is stated in Table 2.
Take for example Attacks A2, A5 and A9. We see in Table 1 that Attacks A2 and A5 need a **P**roximity of 70m or less to the devices, such that ZigBee packets can be received and sent. However, for Attack A9 direct physical access to the device is required in order to push the button on the device. For Attack A5 the **N**WK Key is supposed to be known to the attacker in advance, which we expect to be possible by performing Attack A2. We also see that these three attacks attack a different **S**ecurity goal (C/I/A).

Attack	**N**	**P**	**S**	**Setups**	**Comments**
A1: Ethernet sniffing	×	×	C	S1, S2	Simulates the ISP and IoT Service providers (the 'cloud')
A2: Passive ZigBee Sniffing	×	√	C	S2, S3	
A3: Send Beacon Requests	×	√	C	S3	
A4: Command injection	×	√	I	S4	Unencrypted or encrypted with the publicly known Global Key
A5: Command injection	√	√	I	S5	Encrypted with NWK Key
A6: Cause PAN ID Conflict	×	√	A	S3	
A7: Disconnect gateway from the internet	×	√√	A	S3	
A8: Turn off/on gateway and devices	×	√√	A	S3	
A9: Reset device	×	√√	A	S3	Use the dedicated button for that
A10: Send high sequence number	√	√	A	S6	Encrypted with NWK Key
A11: Move device out of range	×	√	A	S6	Simulates temporary jamming

Table 1: All attacks that we perform with the presumed knowledge and proximity and the security goal that is under attack by it.

N	=	NWK Key is presumed to be known to the attacker
	×	= No
	√	= Yes
P	=	Physical proximity required
	×	= No
	√	= Yes, within 70m of the network's devices
	√√	= Yes, direct physical access to the device
S	=	Security goal (see Section 3.2)
	C	= Confidentiality
	I	= Integrity
	A	= Availability

Table 2: Symbols used in Table 1

3.4 Setup

To do our experiments, we set up a ZigBee network using the LSH products in different setups as to how they can be expected to be used by users. In Figure 2 we show the schematic topology of the full setup, that is with all available devices as an overview. For the experiments, we use different partial setups regarding the end devices. We explain and motivate these setups in Section 3.4.4.

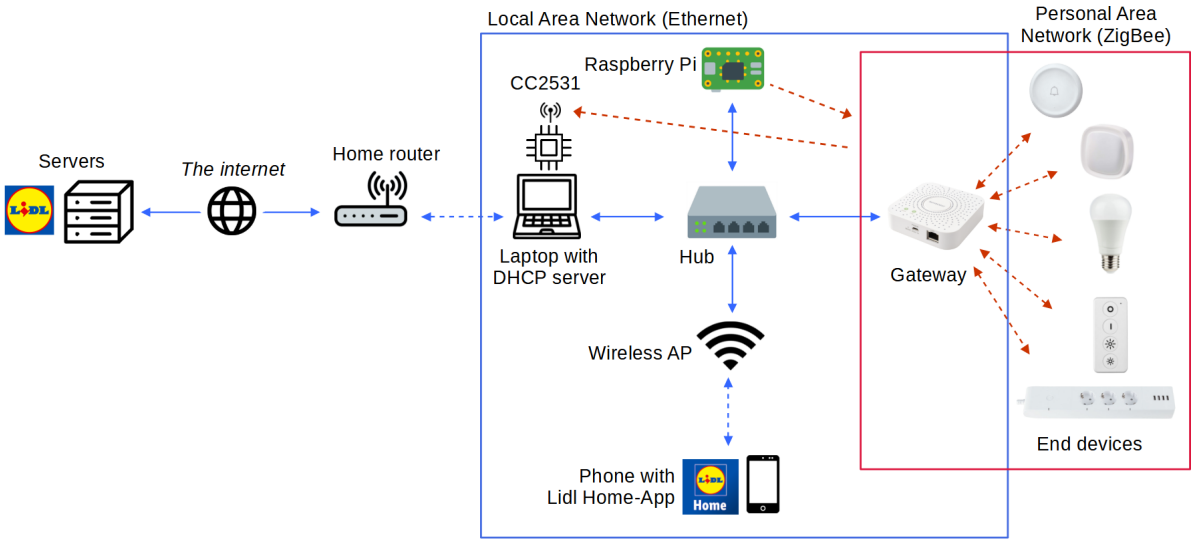


Figure 2: Topology of the full attack setup (see Section 3.4.4 for used configurations)

To control the devices with a smartphone, Lidl published the Lidl Home app. The app is available for free for Android and iOS. We use an iPhone X and therefore the app for iOS, version 1.0.16, which is the latest version at the start of our research. To use the app, the user has to create an account. For this, only an email address is needed. No further private information is asked.
Part of the Lidl Smart Home infrastructure are servers. These enable users to control their products from anyplace with an internet connection. As explained in the Introduction (Section 1), these servers are "Microsoft Azure servers that are stationed in Europe" [46]. As we will see in the results (see Section 6.1.2), the app also contacts servers from Akamai Technologies, Inc.
To capture ZigBee traffic we use a TI CC2531 as explained in Section 3.4.2. To send ZigBee packets we use a Raspberry Pi with the Raspbee hardware module, as did Gleason and Brown [26] in their research. We explain this further in Section 3.4.3.

To capture the ethernet traffic of the gateway and the phone, we set up a Local Area Network on an HP Pavilion laptop running Ubuntu 20.10 using the default network manager, which sets up and runs a DHCP server. The Raspberry Pi and the gateway are connected to the laptop's network via a hub. The phone is connected via a wireless access point (AP), which is connected to the hub. The Wireless AP does not set up its own network, it merely extends the existing one. By setting up a Local Area Network this way, all traffic within it can be captured using Wireshark on the laptop without interference by network traffic from other applications on it.

3.4.1 List of devices used in research

For this research, we choose a variety of LSH products with different functionalities. These are the products we investigate:

- **Gateway:** it is the core of the network and is responsible for creating and distributing security keys. For other devices to work properly, it needs to be connected to the internet via a LAN cable. End devices connect to it via ZigBee. The user can change the settings via the app on the phone. The gateway is grid-powered.
- **Power socket:** it has three power sockets type F [49] and four USB type-A power sockets. There is an on/off button on top and an isolator button at the side. The socket is grid-powered.
- **Motion sensor:** it registers motion on the front within a range of 10m with an angle of 90 degrees, according to the user manual. In normal use, the user receives a push notification on the phone when motion has been sensed. The device is battery-powered. The battery (CR123A/CR17345, 3V) has to be put in at the back by sliding down a plate at the back and then opening the battery case. Removing the plate also triggers the release of a button that is held pressed by that plate. In the case where the device has already been added to the network and is functioning normal (and there is a charged battery present), the release of the button triggers a manipulation warning (see Appendix E). Except for the button covered by the plate, there are no other buttons.
- **Doorbell:** the doorbell can be pressed on the front. In normal use, the user will receive a push notification on the phone about the doorbell being pressed. The device is battery-powered. To put in the battery (CR2450), the plate at the back has to be turned left (or, if the plate is fixed to a wall, the doorbell has to be turned left), which opens the device at the back and, like the motion sensor, triggers a button on the inside to be released which was being held pressed by the plate. This button triggers a manipulation warning (see Appendix E) as well.
- **LED lamp:** the lamp is an RGB lamp for which the color and brightness can be set in the app. There are no buttons on it. The lamp is grid powered and has to be put into an E27 lamp socket.
- **Remote control:** in the app, the user can choose which lamps should be controlled with the remote control. The remote can be assigned to a single lamp or a group of lamps. The remote has four buttons: 'big sun' (brighter), 'small sun' (dimmer), `I` (on) and `O` (off).

We argue that having all different kinds of lamps will not add significantly to this research because, although their appearances differ, we assume that on a technical level they work very similarly. We do not include the heater in our research because it does not use ZigBee but solely WiFi and thus is not within the scope of this research. A variety of products such as a watering computer, thermostat, and carbon monoxide alarm have been added to the assortment after the start of our research. We do therefore not consider them within our research.

3.4.2 ZigBee: passive sniffing

The network's main communication protocol is ZigBee. It is a wireless connectivity standard that is created for controlling smart home devices. ZigBee is based on the IEEE 802.15.4 standard and is therefore suitable for low-energy networks [11].

To capture ZigBee traffic, we flash a TI CC2531 USB Software Defined Radio [34] with Texas Instruments' (TI) Packet Sniffer software [33]. The output of the CC2531 gets fetched and put into `.pcap` format using whsniff [35], which is then piped to Wireshark or the WhsniffProcessor, the latter of which is part of our Neckarspwn framework[2]. Manual inspection of packets is done in Wireshark. WhsniffProcessor is used for attacks that need an automated reading of packets, as it parses `pcap` formatted data in real-time and outputs a Scapy packet object (`Dot15d4FCS` class type [36]). The `pcap` format is a standardized format for network packages that enables us to open the captured traffic with other software, such as Wireshark or Scapy. The `Dot15d4FCS` class type is part of the Scapy library and models the IEEE 802.15.4 MAC Layer with Frame Control Sequence. The upper layers are automatically parsed in Scapy.
Here we describe the hardware and software packages we use to listen to ZigBee traffic in more detail.

- **TI CC2531 USB Software Defined Radio** [34]
  The TI CC2531 is flashed with TI's Packet Sniffer software [33]. It is TI's own driver for the TI CC2531. It transforms the electromagnetic signals (PHY Layer) into bytes.
  There are alternatives, like for example ZigBee2MQTT [52], which transforms the ZigBee traffic into MQTT packets. Because we need to read raw (unedited) bytes so that we can later craft our own packets, this is not suitable for our research.

- **Whsniff** [35]
  The raw bytes from the TI CC2531 are parsed by Whsniff and transformed into `pcap` format. Whsniff is created by 'homewsn' and is publicly available on GitHub.

- **Wireshark** [55]
  Wireshark is an open-source tool for network packet inspection and supports the ZigBee protocol. We use the Wireshark ZigBee Profile [39], which has color-coding for different kinds of ZigBee packets, made by Akestoridis et al. [28] for convenience. Wireshark has a built-in decryption function for ZigBee packets which automatically decrypts packets if the corresponding key is given. The keys can be added to Wireshark in the protocol configuration (Edit > Preferences > Protocols > ZigBee > Pre-Configured Keys > Edit). If the decrypted packet is a Transport Key packet (and thus contains another encryption key), then this key is automatically used by Wireshark to decrypt all following packets within the same `.pcap` file that have been encrypted with this new key. In Sections 5.2 we explain ZigBee packets, the different keys, and the encryption of the packets in greater detail.

- **Neckarspwn: WhsniffProcessor**
  To interactively communicate with the tested devices (thus sending packets based on what we receive), we need a real-time `pcap` parser. However, to our knowledge all publicly available `pcap` parsers only take complete files as input and do not return anything before the end of the file has been reached. Therefore we use our own WhsniffProcessor written in Python, which takes the `pcap` formatted stream from `stdin` as input and for every packet it reads it immediately returns a Scapy `Dot15d4FCS` object. This enables us to interact with the tested devices by sending packets (see Section 3.4.3) based on the captured packets, for example by adjusting sequence numbers.

3.4.3 ZigBee: packet injection

To craft ZigBee packets, we use the Scapy framework [56]. Among its features are automatic parsing of packet bytes and Frame Control Sequence calculation. We send packets with a Raspberry Pi 3 with the Zigdiggity [38] framework and the Raspbee hardware module that Gleason and Brown used in their research [26]. Although this hardware module is also able to receive packets, we do not use it for this as it drops the first byte of every packet, which makes it unusable for listening. We do not know why this happens. Our solution is to use the CC2531 for receiving packets and the Raspbee module for

[2]Neckarspwn is the framework we created to test the LSH products. Although we provide it on GitHub [50], it remains a proof of concept and is by no means production code.

sending packets. Therefore, during hybrid attacks that require both sniffing and packet injection, the Neckarspwn framework uses both hardware modules (Raspbee and CC2531) at the same time. Because the Raspbee and CC2531 run on two different devices, the output from the CC2531 (on the laptop) is piped via SSH on the Local Area Network to the Raspberry Pi.

As we will show in Section 6, we encounter the 'IAS Zone Status update' ZigBee packet during many attacks. The latest Scapy version during our experiments (v2.4.5), however, does not feature such a class and thus parses it as raw data. Therefore, we extend the Scapy [56] framework with an 'IAS Zone Status Update' class. With this extension, we can easily create and modify these kinds of packets such that we can inject them into the network. The Python class is shown in Appendix F. An example of a packet that gets parsed with this class is shown in Appendix D.5.
On August 29, 2021, an update to Scapy was commited [41] by Dimitrios-Georgios Akestoridis, which includes a `ZCLIASZoneZoneEnrollResponse` class, `ZCLIASZoneZoneStatusChangeNotification` class and `ZCLIASZoneZoneEnrollRequest`. However, in contrast to our `ZCLIASZoneStatus` class (see Appendix F), the `ZCLIASZoneZoneStatusChangeNotification` class does not feature the meanings of separate bits of the packet. As we explain in Section 6.1.2, the separate bits as explained in Appendix F can be used to determine whether a device is a doorbell or a motion sensor.

3.4.4 Setup configurations

For our experiments, we set up the devices in different configurations. Here we explain and motivate them. In the diagrams, we only show the test setup of the LSH products and the hardware to attack them. In each configuration, it is implied that the phone is connected to the LAN and that the laptop is connected to the internet and that therefore the gateway is, too, if not explained differently (Attack scenario A7).
For each setup, we refer to the Attack scenarios (AS) for which we use that setup. We explain the Attack scenarios in Section 3.3.

S1 In this initial setup, we only use the *gateway*.
To test the default behavior of the gateway at initialization, we only connect the gateway (see Figure 3). This way, we do not get any disturbances from other devices, that would eventually lead to additional traffic. By only using the gateway, we can learn a 'ground zero' of ethernet communication, such that we can later distinguish what is caused by other devices.

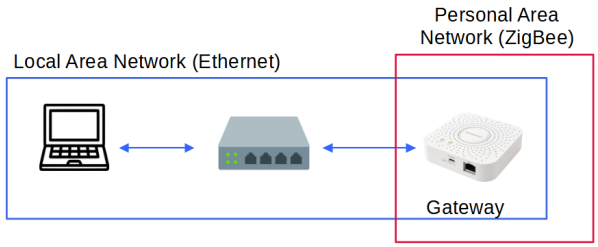


Figure 3: Setup configuration S1

S2 In this setup, we use the *CC2531, gateway, doorbell, and motion sensor*.
As we will show in Section 3.3, we perform a number of attacks (AS A1 and A2) that involve passive sniffing, push notifications, and manipulation warnings. For these attack scenarios, we organize the devices as shown in Figure 4. The two shown devices, the doorbell and motion sensor, are both capable of generating push notifications. We use this setup to capture ethernet traffic using the laptop. With the CC2531 we can listen to the ZigBee traffic to correlate that with the ethernet traffic.

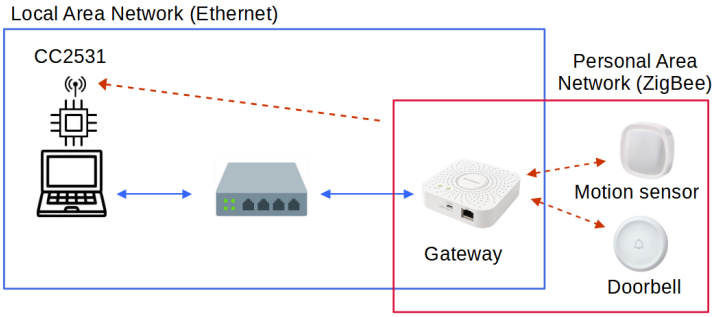


Figure 4: Setup configuration S2

S3 This is the full setup with all end devices. It consists of the *CC2531, Raspberry Pi, gateway, motion sensor, doorbell, power socket, light bulb, and remote* (see Figure 5).
By using this setup, we can learn about the ZigBee communication in a fully functioning network. First, we only listen to the ZigBee traffic and learn how the devices behave. As we show in Section 6.1.2, this suffices to learn, the encryption keys that are used in the network to encrypt the network traffic and the sequence numbers on all layers (see Section 5.2.3). Then, we use this information, the Raspberry Pi and the Raspbee module to send crafted ZigBee packets. This allows for a wide range of active attacks on the network. As we show in Section 3.3, we actively create a topology of the network, purposely create a conflict to see how it is solved, measure the effects of taking the network offline, and reset devices to see what effects that has.

Because of their functionality, the light bulb, and power socket are devices whose behavior we try to modify. The doorbell, motion sensor, and remote, however, are devices that are used to 'register an event' and notify devices about that. Therefore, we can use these devices by imitating them.

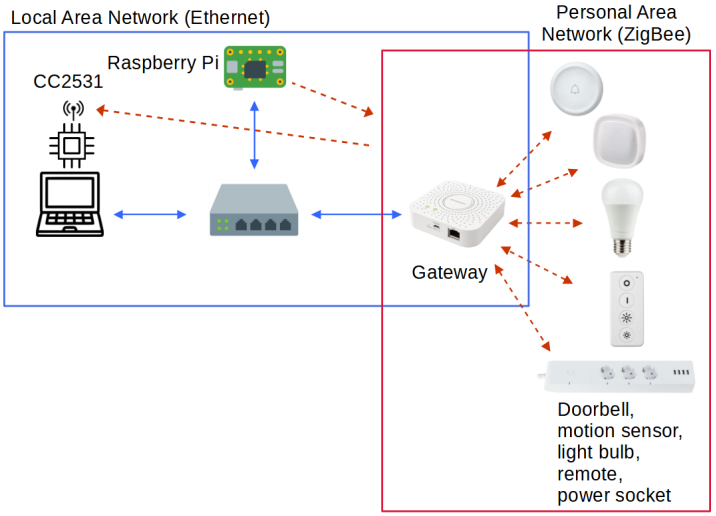


Figure 5: Setup configuration S3

S4 In this setup we use the *CC2531, Raspberry Pi, gateway, doorbell, motion sensor, and power socket* (see Figure 6).
We use these devices for forging and sending unencrypted network commands to these three end devices. Compared to Setup S3, this setup lacks the remote and the light bulb because we do

not target them. We choose the doorbell, motion sensor, and power socket to perform Attack A4 on multiple different devices. However, because the attacks do not focus on the specific functionality of the end devices, we expect that the choice of devices does not affect the results of the experiments.

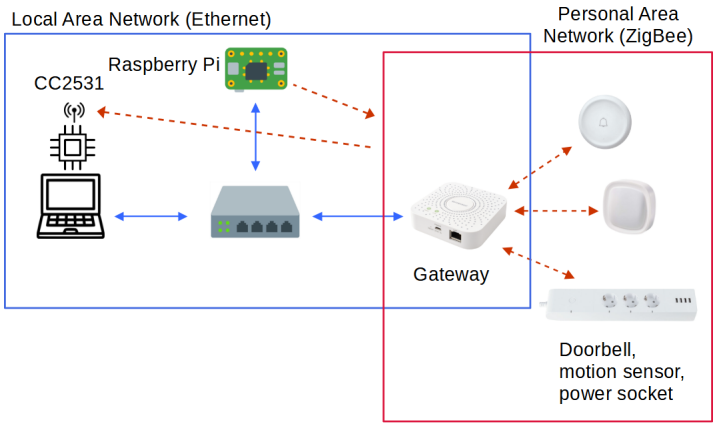


Figure 6: Setup configuration S4

S5 In this setup we use the *CC2531, Raspberry Pi, gateway, power socket, motion sensor, light bulb, and remote* (see Figure 7).
We use this setup for listening to genuine commands, such that we then can forge our own commands. With these commands, we ought to control the devices in different manners. These devices serve different purposes in the active attacks: the motion sensor can not be controlled, but by imitating it, we emulate traffic from it to the gateway and initiate traffic from there to the phone (via multiple hops). The lamp has an obvious visual function, which we manipulate. For that, we emulate commands from an end device (the remote), via the gateway to another end device. The power socket bar has multiple sockets, which could be tried to control individually. For this, we emulate the gateway directly. Therefore, we test three different 'traffic flows' with this setup. We expect that adding the doorbell would not add to the results, compared to the motion sensor.

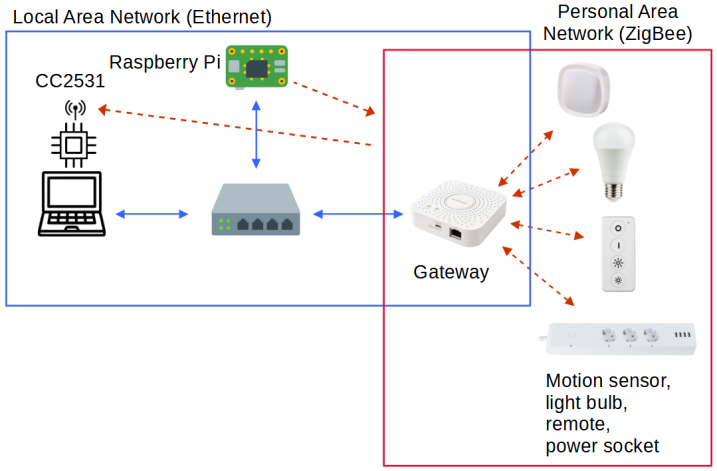


Figure 7: Setup configuration S5

S6 In this setup we use the *CC2531, Raspberry Pi, gateway, and doorbell* (see Figure 8).
We use this setup to test the ZigBee protocol for a possible Denial of Service attack that exploits the misuse of a sequence number. To be able to check whether a device is still available, we need a device that we can trigger at will, but do not trigger accidentally. Therefore the doorbell is best suited for this. As the attack itself is based on the security counter (explained in Section 5.2.4), it is for the attack not important what the device functionality is, but only that it encrypts packets and therefore has a security counter in those packets. As we will see in Section 6.1.2, this is the case for all devices.

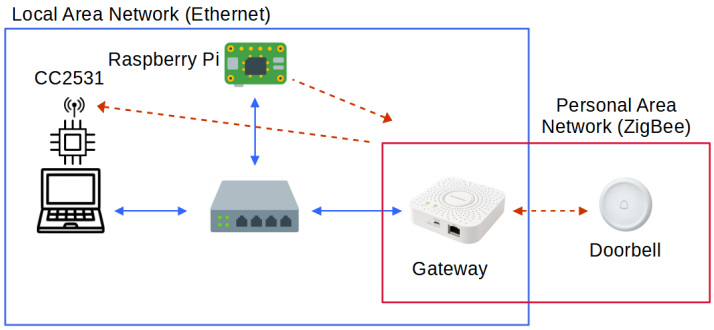


Figure 8: Setup configuration S6

4 Related work

In this section, we look into research done in the field of IoT. In Section 4.1 we discuss security mechanisms for ethernet based smart home setups. We discuss known attacks on the ZigBee protocol in Section 4.2.

4.1 Connectivity in IoT

We see in the literature on IoT that many devices are not properly secured [24][25], which results in IoT devices very often being used for botnet attacks [23]. One factor in the security of a device is its ethernet connectivity, which is often not TLS/SSL secured. To this problem, some solutions have been proposed [16][19][22] that all have in common that they rely on some hardware or software residing between a group of IoT products and the internet.
Stewart et al. [16] noted that networks attack are no longer just annoying, but "can now translate into a loss of money, a source of physical harm, or even widespread chaos" [16]. They propose a system called Community Guard which acts as a physical firewall between a user's home and the internet. It "connects to a cloud system that automatically monitors for suspicious activity and deploys the appropriate response" [16] to threats such as attempts to infect IoT devices with malware that would make them part of a botnet. Community Guard relies on collaboration between deployed instances, as they are supposed to share information on new malware, which would make the Community Guard system more secure [16]. Community Guard is based on Snort [3] rules, which can for example be used to block traffic if it uses a specified protocol or port, if the traffic contains some known malware or if the IP source address is within a specified range [3]. New IP addresses are added to the blacklist of Community Guard by means of a consensus mechanism [16]. Stewart et al. demonstrated that during a DDoS attack, legitimate traffic coming from the same IP as the attack packets was still let through [16]. This shows that Community Guard is 'smart' in the sense that it distinguishes between different traffic patterns.
Ko et al. [22] also came up with a mechanism to protect all IoT devices within a home at once. They proposed Deadbolt, a framework to hide all IoT devices behind an access point that uses a deny-by-default policy [22]. It requires IoT devices to use the latest available firmware, otherwise, the traffic from the device will be blocked. If a device communicates in clear text, DeadBolt encrypts the traffic with TLS to protect against eavesdropping attacks [22].
Heimdall by Habibi et al. [19] is yet another "whitelist-based" [19] (a.k.a. deny-by-default) framework that operates as a filter between a group of IoT products and the internet. It creates profiles by looking at traffic based on IP addresses and protocols used, including timing and patterns. After learning the 'normal' behavior of a device, it keeps enforcing this behavior, that is, it blocks traffic that is not 'normal' according to the generated profile [19]. The objective is to prevent that devices participate in a DDoS attack [19].

Ren et al. [25] took a more privacy-oriented look at IoT. They did a case study in which two rooms (one in the US and one in the UK) were filled with all kinds of IoT devices, for example, cameras, smart hubs, lights, power plugs, TVs, audio devices, and smart kitchen devices. They observed that the most traffic goes to the US, for both the UK and the US lab [25]. The most contacted parties were AWS, Google, Akamai, and Microsoft [25]. TVs often also connected to Netflix [25]. TVs are also the devices that showed different behavior based on their location (inferred from the IP address), by displaying different content to the user [25]. In the limited amount of unencrypted traffic Ren et al. [25] found relatively little sensitive information, but, as they stated, by traffic pattern analysis, an adversary could be able to infer the category of an interaction with a device [25]. Ren et al. [25] also found that some devices displayed unexpected and sometimes also unwanted behavior. One smart speaker behaved differently when it was connected to the internet via a VPN. Some cameras kept recording and sending photos or videos to a server without the user being able to access them. For one camera the user had to pay extra to get access to the device. There was also a camera that made undocumented recordings [25].

These findings show that there can be extra privacy concerns about the devices, even if according to the documentation the behavior of the device is acceptable.

4.2 Research on ZigBee

Li et al. [27] did a formal security analysis of ZigBee 1.0 and 3.0 by modeling it in Tamarin [42]. Tamarin is a protocol verification program tailored for security protocols [42]. It enables the verification of properties by creating a model of the protocol, different actors, for example "the protocol initiator, the responder, and the trusted key server" [42] and an adversary.
Li et al. found that in ZigBee 1.0 the confidentiality of the *Network key*, pre-configured *link key* and *application link keys* is violated, because Tamarin found a way how an adversary could learn all keys [27]. ZigBee 3.0's feature of out-of-band shared install codes, however, makes sure that the confidentiality of all keys is upheld. But Li et al. [27] also noticed that manufacturers, as of 2020, still prefer older ZigBee versions over ZigBee 3.0 because of the higher usability. Some of the smart-home IoT devices they tested still used the global key for encrypting and transporting the *network key*.

Akestoridis et al. [28] built Zigator [37], a software tool for security analysis of ZigBee networks. Among its features are "deriving preconfigured *Trust Center link keys* from install codes" [37], decrypting and verifying ZigBee packets and producing statistics and visualizing data from a database of ZigBee packets. To make an initial analysis of the captured ZigBee communication simpler, they also created a Wireshark profile for ZigBee packets [39].
Akestoridis et al. [28] focused their study on the "design choice to disable MAC-layer security" and show that despite NWK-layer encryption certain NWK commands can be identified with 100% accuracy with a decision tree. This can be used to perform selective jamming and spoofing attacks that force the end-user to factory reset the target devices, which could then reveal the *NWK Key*. To achieve this, a PAN ID conflict is caused by injecting a forged beacon, that contains the correct PAN ID of the victim's network, but has a different Extended PAN ID (EPID). This conflict causes the coordinator to select a new PAN ID and broadcast that throughout the network by means of the Network Update command. If a device, however, does not receive this command, it will have to rejoin the network, which means that there will be a moment of vulnerability of the network which can then be exploited [28]. As explained in greater detail in Section 5.2.2, the coordinator coordinates the ZigBee network and is among other things responsible for distributing addresses and keys amongst the devices. According to Akestoridis et al. [28], these denial-of-service attacks lead also to the following: (a) the device user will have no further control over the device, (b) the device will be unable to send messages to the user, and (c) automation rules will no longer function properly [28].
A critical detail of the IEEE 802.15.4 standard for this attack to work is that "ZigBee devices that use the OQPSK PHY layer [..] in the 2.4 GHz band will wait for 864 microseconds to receive a requested MAC acknowledgment" [28]. This means that packets have to be processed on the receiving device itself and cannot be transferred to a processing unit elsewhere, because of the short response time, according to Akestoridis et al. [28]
Akestoridis et al. [28] also found that legacy devices (that is with ZigBee version < 3.0) that do not use the *install codes*, and thus rely on the preconfigured *trust center link key*, form a critical vulnerability for the whole network. These devices can be targeted to gain the *NWK Key*, after which "the attacker can decrypt most of the encrypted payloads and inject commands that change the state of the end user's devices, including Zigbee 3.0 devices that use install codes" [28]. Therefore even one legacy device could compromise the entire network.

In order to prevent replay attacks, ZigBee has built-in 4-byte frame counters (security counter), with values ranging from `0` to `0xFFFFFFFF`. Farha et al. [31] claim that under specific circumstances, despite this mechanism, replay attacks are still possible. For this, they capture many packets, after which they reset the network, and thereby the frame counters. If devices do reset their counters, the earlier captured packets could be replayed. If those packets happen to be commands, an attacker could control

the devices with those packets until she runs out of earlier captured packets.
As a 'solution' to the replay attack, Farha et al. [31] propose to introduce the use of a timestamp instead of an increasing frame counter. It is noted that due to the resource constraints, only the coordinator and the routers will have a clock mechanism, and thus the end devices will have to ask for the current time every time they want to send some data [31]. This introduces a lot of overhead in terms of network traffic and time consumption. For end devices, this would mean that they'd have to be awake longer and thus that their battery time decreases.
Farha et al. [31] also propose a Denial of Service attack by forging a packet with a high frame counter. *If* a genuine device does indeed accepts a packet with a frame counter that is much higher than the latest saved value, this could result in this device saving the high frame counter value and then rejecting genuine packets for a long time, because their frame counters are less than the saved value. It is interesting to see whether this attack is possible in a secured ZigBee 3.0 network.

5 Background

Here we describe the Lidl Smart Home products, the ZigBee protocol, install codes, and the CCM* encryption scheme. In section 5.1, we introduce the LSH product line with regards to the functionality it offers to users. We describe the ZigBee protocol with a focus on the different security keys in Section 5.2. We describe what install codes are and how they are transformed into install code link keys in Section 5.3. In Section 5.4 we explain the CCM* encryption scheme that is used in the ZigBee protocol.

5.1 Lidl Smart Home

The LSH series contains all-day products for private domestic use like lamps and a power socket. It features dedicated security-critical products like a door/window sensor (which we do not have in our testbed). Also, objects like the motion sensor and lamps may be used as part of a security-critical setup. Therefore, security should be an unabated aspect in the design of the products, as compromised security could potentially have a Denial of Service, information leak, or other (possibly yet unknown) consequences as a result.
The LSH products use ZigBee for local device communication. The IEEE 802.15.4 protocol, on which ZigBee is built, has a physical range of approximately 70 meters indoors [29] and thus so do the LSH products. To use the devices, the user needs to have a gateway, which can be bought from Lidl. Alternatively, a third-party gateway may be used. The gateway is the heart of the network and 'translates' commands from ethernet to ZigBee commands and vice versa.
The ZigBee communication is ZigBee Alliance certified, which means that it got tested, which is mandatory [29] for all ZigBee products to become certified. The LSH products are certified by the TÜV Rheinland[3] [44]. The other authorized test service providers for ZigBee communication are ESI[4], element[5], NTS[6] and UL[7] [29]. The communication between the gateway, router, app, and server, however, is specific for the Lidl Smart Home product line. This also holds for the architecture and implementation of the app and the servers. It is for that reason that the known attacks from the literature, as described in Section 4, focus on the ZigBee 3.0 communication.
The offered devices that provide the functionality a user sets up the network for in the first place are called end devices. These are for example a lamp for light, a power socket to turn on and off 'dumb' devices, or a motion sensor to register movement. The normal setup for the LSH products is to have a gateway connect to all end devices via ZigBee, as shown in Figure 9.

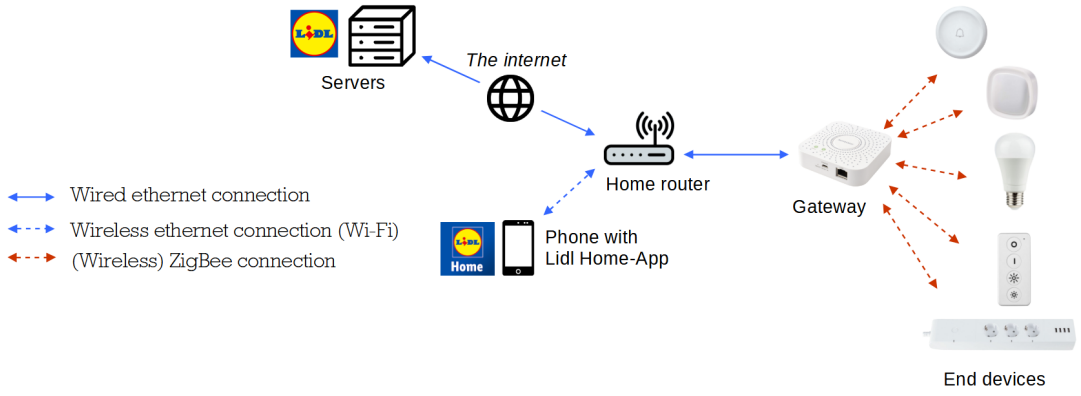


Figure 9: Topology of the LSH products in a normal setup.

[3] https://www.tuv.com/
[4] https://www.esi.net/
[5] https://www.element.com/
[6] https://nts.com/
[7] https://www.ul.com/

5.1.1 Topology

The gateway is connected to the internet via an ethernet cable which is plugged into the home router. The end devices can be controlled by the user in the Lidl Home app (iOS or Android). To initially connect the phone and the gateway with each other, both devices have to be in the same local area network. The gateway's instructions state: "Your mobile device needs to be connected to the WiFi router's wireless network to perform the installation" [48]. After the phone and gateway have been connected, it does not matter anymore whether the user's smartphone is connected to the internet via the home router or an external network (e.g. mobile network), because part of the product line is a cloud service infrastructure via which commands can be sent. The alternative topology where the phone is connected via an external mobile phone network is displayed in Figure 10.

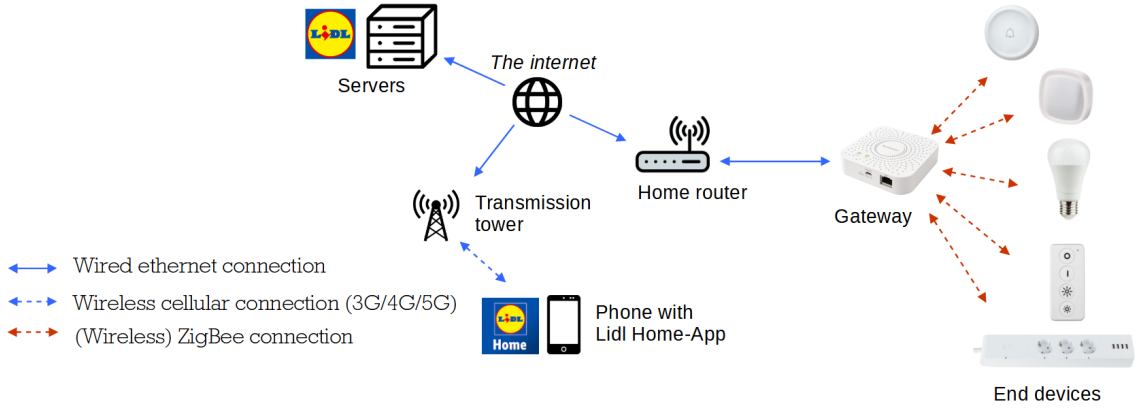


Figure 10: Alternative topology of the LSH products in a normal setup.

5.1.2 Add a device to the network

To add a device to the network, the user has to put the device into pairing mode. In most cases, this is done by pressing the dedicated button on the device three times. The lamp is already in pairing mode when it is turned on the first time. The gateway has to be told to accept the new device by the user via the app. In the app, the user can group the devices into 'rooms' and give devices an alias.

5.1.3 Remove a device from the network

After a device has been added to the ZigBee network (see Section 5.1.2), the device can be used for its normal functionality. It can also be properly disconnected from the ZigBee network. There are two main methods to do this: it can be done in the app or directly at the device. To disconnect a device via the app, the option "remove device" at the device's options has to be selected. The device is now disconnected and can be added to a (potentially different) network again, as described in Section 5.1.2. To disconnect a device without using the app, most devices have a dedicated button for (de)authenticating a device. If it is pressed three times within three seconds, the device disconnects itself from the ZigBee network. To reach this button at the motion sensor and doorbell it is necessary to open the device at the back. The lamp does not have any buttons. It can be disconnected by turning on and off the power three times within three seconds.

5.1.4 Automation

The LSH products offer the possibility of automating certain processes in the network. These automation processes can be trigger-based or time-based. An example for trigger-based automation: it is possible to set up in the app that, if the motion sensor is triggered, a lamp goes on. Time-based

automation can be based on a specific time of the day, the time the sun goes up or down, or a certain amount of time after another event happened. This way, it is for example possible to extend the previous example such that it is only triggered by night and additionally, the lamp is turned off automatically after three minutes.

5.1.5 Network size

Given that the LSH network is based on ZigBee, there is a theoretical maximum of 65,000 devices per network [29]. For regular domestic use, we don't expect people to use that many devices. However, we do consider it realistic that people buy tens of lamps, for example to lighten multiple rooms throughout their house or to use is as an anti-intruder system by scaring burglars that feel safer in the dark. Although we don't test the feasibility of having so many LSH products in one network, we do not expect users to have any problems in doing so.

5.2 ZigBee

ZigBee is a networking protocol created by the ZigBee Alliance. Its purpose is to provide a communication protocol that is suitable for IoT products by making it lightweight and flexible for developers and to have a large share of IoT devices from different vendors being compatible with each other [54]. Because of the implicit resource constraints for IoT devices, the protocol must be lightweight, such that it only needs limited power and storage capacity and the communication hardware fits in the (often very small) end devices [15].
ZigBee defines the optional use of install codes, which are further explained in Section 5.3. Install codes are used to ensure secure device communication, but their usage requires at least one of the two communicating nodes to have some form of user entry. Also, the codes would have to be provided by the vendor, for example on stickers, which also cost resources. Not using Install Codes is thus easier and cheaper, thus providers are more likely to choose not to use them. This is an implicit conflict of interest between the security and prices of products.
In this subsection we explain technical details of the ZigBee protocol, such as the different actors within a network (Section 5.2.2), the layers of a packet (5.2.3), and the different kinds of security keys (Section 5.2.5).

5.2.1 Spectrum

The ZigBee standard defines 27 channels in total [11]. Channel 0 uses the 868 MHz band, uses BPSK modulation, and has a bit rate of 20 kb/s. Channel 1 to 10 use the 915 MHz band where each channel has a bandwidth of 2 MHz, thus channel 1 uses 906 MHz and channel 10 uses 924 MHz. They also use BPSK modulation and have a bit rate of 40 kb/s. Channels 11 to 26 use the 2.4 GHz band. The channels 11 to 26 have a bandwidth of 5 MHz, where channel 11 uses 2.405 GHz and channel 26 2.480 GHz. These channels use OQPSK modulation and support a bit rate of 250 kb/s [43].

5.2.2 Actors

The different types of actors in a ZigBee network are coordinator, router, and end device [29]. The coordinator is the core of the network and is responsible for the management of the network, which can include letting other devices join, handling key requests, creating and distributing the different keys [27] and translating commands from ethernet to ZigBee and vice versa.
A router functions as a relay. By default, ZigBee uses a mesh structure, which ensures that if one device falls out of the network, this will most likely not be of influence to the other devices connecting through it, as there will be other routes established within the network. If an end device is out of range of the coordinator, it will use routers to connect to the coordinator [29]. The LSH series does not contain dedicated router devices.

The end devices are the devices a user sets the network up for: a sensor, switch, light bulb, remote, doorbell, power socket, etc. End devices can also act as a router between another end device and the coordinator [32]. An example network structure in a ZigBee network with centralized security is shown in Figure 11.

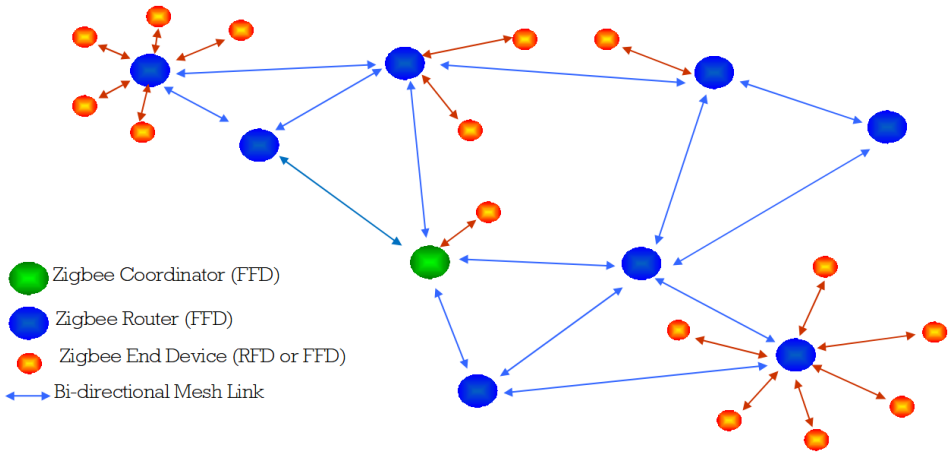


Figure 11: Example ZigBee topology in a centralized security system, from: [29]

5.2.3 Layers

On a technical level, the ZigBee standard is based on the IEEE 802.15.4 standard [6]. It has a physical range of 70 meters indoor and 300+ meters with a line of sight and it supports a data rate of 250 Kbit/sec [29]. The IEEE 802.15.4 standard defines the physical layer (PHY) and the Medium Access Control layer (MAC) [27]. The ZigBee standard defines the Network layer (NWK), the Application layer (APL), and the Application Support Sub-layer (APS), as shown in Figure 12.

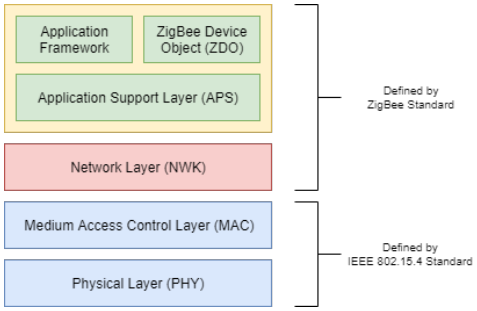


Figure 12: ZigBee Stack Architecture

The physical layer consists of the signals in the electromagnetic spectrum and is defined in the IEEE 802.15.4 standard. The MAC layer is also part of the IEEE 802.15.4 standard and is responsible for transporting a packet from one hop to another. It defines how addresses get assigned to devices and how the mesh network is maintained. It has a built-in message integrity check, called Frame Control Sequence (FCS), which is based on the CRC-CCITT checksum as defined in the Kermit protocol [1]. The NWK Layer is defined by the ZigBee standard and is responsible for carrying packets through the network. Confidentiality of the commands (in the APS layer) is ensured in the network layer by means of encryption with AES in CCM* mode, as explained in Section 5.4.

The APS Layer is the carrier of the commands that come in the ZigBee Device Objects (ZDO's). The ZDO is always either a ZigBee Cluster Library Frame (ZCL Frame) or ZigBee Device Profile (ZDP). Within a ZCL Frame, a vendor may specify vendor-specific fields and commands.
Not all packets in a ZigBee network need to have all layers. An ACK message, for example, is 5 bytes in size and only consists of a MAC Layer. It holds a Frame Control Field (2 bytes), the sequence number that is acknowledged (1 byte), and the Frame Control Sequence, which is an integrity check (2 bytes). The regular Link Status updates that are sent within the network consist of the MAC and NWK Layer (with Security Header), but do not have an APS Layer. These messages tell other devices in the network to which devices the sender is connected. This is how ZigBee manages the mesh structure within a network.

5.2.4 ZigBee Packets

There is no default format for a ZigBee packet because packets are built modular and many different options can be set per packet and layer in the Frame Control Fields (see Appendix C). The number of layers per packet is variable:

- packets can have all layers (see Appendices D.2, D.3 or D.5 for examples),
- lack only an Application Layer (example: Appendix D.4), or
- only consist of the Physical and MAC Layer (example: Appendix D.1).

The different layers in the ZigBee protocol stack are not completely separated in that sense that they share addresses and different layers can have the same auxiliary headers. Here we describe this in greater detail.

Addresses in IEEE 802.15.4 and ZigBee
Both in the IEEE 802.15.4 and ZigBee standard, a device has a short (2 bytes) address and an extended (8 bytes) address. Because the MAC layer (defined in 802.15.4) is responsible for the routing, that is also where addresses are requested and shared. This is done via beacon requests and beacon responses. Devices are open to receiving beacon responses at any moment, also if they did not send a beacon request.
The addresses used for the devices in the NWK layer are the same ones as those used in the MAC layer. This results in most packets having the same source and destination address stated twice. The PAN ID is only mentioned in the MAC layer. It has, just like the device addresses, a short and extended variant. For all addresses holds that the Frame Control Field of the corresponding layer tells what variant of the address is being used. The most used variant is the short address, which makes sense considering network efficiency. The standards also allow having no addresses in a packet on any level.

Sequence numbers
All four digital (sub)layers (MAC, NWK, APS, and ZCL) have their own sequence number. On top of that, the Security Header (also called Auxiliary Header) also has its own sequence number. The sequence numbers are a measure against replay attacks. All counters, except the one in the security header, are 1 byte long and range from 0 to 255. The security header frame counter is 4 bytes long and ranges from 0 to 0xFFFFFFFF. A packet can be encrypted on both the NWK Layer and APS Layer at the same time (examples: Appendices D.7 and D.8). In that case, the packet has two Security Headers and therefore two (different) security counters.

5.2.5 Security keys

Joining a network consists mainly of two parts: the new device is registered at the coordinator and it gets sent the keying material such that it can participate in the (encrypted) network communication. After a device has successfully joined the network, all traffic that has a NWK Layer is encrypted on the NWK Layer. The NWK header is not encrypted, but NWK commands within that layer are encrypted, just like the whole APS Layer, if it exists.

If a device knows the *NWK Key*, it can participate within the network. By extension, this also holds for an attacker, as Zillner [14] showed. By knowing the *NWK Key*, the attacker can let a device be part of the target network to the extent that it can participate in it, as it can receive and decrypt messages and encrypt new ones and send them to devices (in the name of another one) [14]. It is not possible to enter as a new device with new addresses, as the coordinator would need to approve of that. But because an attacker can imitate other devices that have already been accepted by the coordinator, an attacker does not need to have her device added to the network as a new one to participate in the network [14]. Zillner [14] notes that there is no rotation of keying material, which means that once an attacker has got hold of a key, the user cannot lock out the attacker.

All keys used in the ZigBee system are 128-bit symmetric keys that will be used in an AES encryption scheme in CCM* mode. According to Jakob Jonsson, this block cipher mode of operation "provides a level of privacy and authenticity that is in line with other proposed modes such as OCB" [4], where 'privacy' refers to what we now call 'confidentiality'. He concludes that the counter mode encryption of the CBC-MAC tag (the MIC in ZigBee) is strong enough to thwart attacks based on the birthday paradox [4].

The keys in the ZigBee protocol can be divided into two categories: link (APL/APS) keys and a network (NWK) key. Link keys are used for encryption on the application layer and application support layer only, the *NWK Key* is used for encryption on the network layer [14] but may also be used for encryption on the application layer [11].
From the network key category, there is only one kind of key: *the NWK Key*. It is used for almost all communication within a network once it is set up, including broadcast messages. The following kinds of link keys exist:

Centralized security global trust center link key
This key is defined in the ZigBee specification [11] as 5A 69 67 42 65 65 41 6C 6C 69 61 6E 63 65 30 39 (hex for "ZigBeeAlliance09") and is pre-installed on all ZigBee 3.0 devices [11]. This key is only used for encrypting the *NWK Key* to send it to a device that is joining the centralized security network [11]. A vendor should, however, prefer the use of install codes above using this key, because it is a predefined key (and therefore known by any adversary), whereas install codes that are sent out-of-band [27] cannot be guessed by a sniffing adversary and therefore provide real confidentiality of the *NWK Key*.
The very existence of this key is a compromise between the policy of never sending a key transport frame unencrypted on the one side and the lack of a secure method to guarantee confidentiality during transport on the other side. The policy came into existence after early analysis of the ZigBee protocol showed that the key was sent as plaintext [29]. Choi et al. [8] proposed in 2012 to use Elliptic Curve Diffie-Hellman (ECDH) for key establishment in the ZigBee protocol to protect against man-in-the-middle attacks and replay attacks. The current version of ZigBee 3.0 is from 2015 and does not feature any version of the Diffie-Hellman protocol [11].

Distributed security global link key
This is essentially the same as the *centralized security global trust center link key*, with the difference that it is used in a distributed security network [11]. In a distributed security network, every router is allowed to establish and distribute encryption keys [11]. This holds for the distribution of the *NWK Key* and the establishment and distribution of *application link keys*. Routers are, however, not allowed to create and/or distribute *trust center link keys*, because there is not one singular trust center, as every router can act as one [11]. Every end device has one dedicated trust center (router), but there is only one coordinator. Thus in this system, the trust center and coordinator are not necessarily the same, whereas, in a centralized security network, they are. Because the LSH network is a centralized security network, distributed security networks are further out of the scope of this paper.

Install code link key
This key is derived from the 18-byte install code (16-byte random key + 2-byte CRC [32]) that is installed on the device during manufacturing and is transported out-of-band (e.g. via a user) to another device. The transformation from Install code to *install code link key* is done via a Matyas-Meyer-Oseas (MMO) hash function [11] (see Section 5.3). The *install code link key* is used to create a unique *trust center link key* in order to be able to receive the *NWK Key* when joining a network [11][27]. This key eliminates the need of a *centralized security global trust center link key* for transporting the *NWK Key* when joining a network because the *install code link key* can be used for encrypting the key transport frame.

Application link key
ZigBee supports end-to-end encryption between two end devices by means of the *application link key* which is distributed by the coordinator upon request by one of the two corresponding end devices [11]. It is used to encrypt a packet on the Application Layer (APL). There can be an *application link key* between every pair of devices, which would result in the existence of at most

$$\frac{(N \times (N-1))}{2}$$

distinct *application link keys* within a network of N devices. That would be the situation in which all pairs of end devices have established one.
In a star-shaped network, all end devices would only communicate with the coordinator. Communication between two end devices would always go via the coordinator. In a mesh network, however, which ZigBee supports, end devices can directly communicate with each other or via a router that simply forwards packets. As we will see in the results (Section 6.1.2), the Application Layer and its content are simply forwarded by routers. Therefore, the theoretical maximum number of *application link keys* within a network does only depend on the number of devices within it, not, however, on its shape.

Trust center link key
This is the same as the *application link key*, but with one endpoint being the coordinator. It can thus be used for end-to-end encryption between an end device and the coordinator [11]. For every end device, the coordinator creates a new *trust center link key*.

5.3 Install codes and Matyas-Meyer-Oseas

To ensure confidentiality of the message during key transport, ZigBee defines the concept of install codes. Install codes are (typically numerical) strings that come with an end device and are in most cases printed on a sticker on the device. The install code is the mutual basis for a pre-shared key, the install code link key, between the coordinator and the end device [11]. The install code link key can be used to ensure confidentiality during key transport [11] under the assumption that an attacker does not have access to the install code. Therefore, an attacker can not decrypt traffic encrypted with an install code link key and thus can not retrieve the keying material within that traffic.
To transform an install code, which is most likely entered by the user via a user interface, into an install code link key, ZigBee uses the Matyas-Meyer-Oseas (MMO) function (see Figure 13). The MMO function is a one-way compression function that works by taking m_i as input for the block cipher encryption E and then XOR's the result with the original m_i. The output of the previous encryption will be used as the key.

$$H_0 = 0^*$$
$$H_i = E_{g(H_i-1)}(m_i) \oplus m_i$$

For the first round, there has to be a pre-defined value to serve as the key. If the pre-defined value and the key for the function E do not match in length, the hash has to be adjusted in a function g, as depicted in Figure 13 and stated in the equation above. In the case of ZigBee, this pre-defined value is the initialization vector $IV = 0$ [11]. ZigBee uses 128-bit AES as the encryption function E [11].

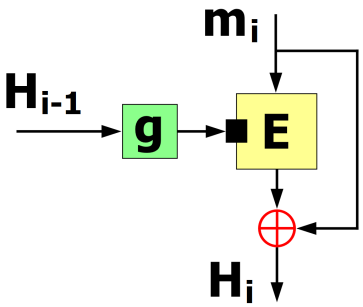


Figure 13: Matyas-Meyer-Oseas hash

5.4 AES and CCM*

CCM is a block cipher mode of operation used in ZigBee to ensure, according to the specification, both confidentiality and authentication [11]. ZigBee uses AES, which is considered secure [17], as block cipher primitive for this mode [11]. CCM stands for Counter with cipher block chaining message authentication code, or short Counter with CBC-MAC. It is a combination of the Counter mode and the Cipher Block Chaining mode [5]. The first step is to compute a tag by taking the output of the last round of the CBC encryption. In ZigBee, this tag is called the Message Integrity Code. Then, the message and the MIC are encrypted using AES in counter mode [4].
The asterisk in CCM* stands for the minor variation of CCM in the ZigBee standard where there is also the possibility to not use the authentication process, but only use the confidentiality functionality of CCM [6].

The encryption, which is AES in counter mode, is essentially an OTP, which means that a single bit-flip in the ciphertext would also result in just a single bit-flip in the decrypted plaintext. This as such could be problematic, as an attacker could modify encrypted messages by selectively flipping bits on specific positions to send an encrypted command with a changed value. This could alter the behavior of a device as opposed to how the user wanted it to behave. The OTP therefore only provides confidentiality.
The CCM's tag is called the Message Integrity Code (MIC) in the ZigBee specification [11]. The MIC is computed by encrypting the message with AES in CBC-mode where the output of the last round is taken as MIC [4]. The MIC is then appended to the original message and also encrypted using AES in counter mode and the same key. Assuming the AES primitive is secure (which is reasonable, see: [17]), the MIC cannot be forged. If an attacker alters an encrypted packet, the MIC will not be successfully verified, the packet will get dropped at the receiving end, and whatever would be the decrypted command of the packet will not get executed. Therefore, the MIC ensures the integrity of a packet with respect to an outside attacker that does not know the encryption key. For that reason, we do not include 'bit flipping' as an attack in our research.
The ZigBee specification claims that the CCM* algorithm also ensures authentication. However, in a ZigBee network all devices use the same symmetric key for encryption (as we will see in the Results in Section 6.1.2). Therefore, the authentication is limited to 'entities that know this symmetric key', because any entity with knowledge of the key can calculate a new valid MIC for any packet. As a result, if an external attacker gets to know this key somehow, there is no mechanism left to protect against her impersonating a genuine device. Whether the CCM* algorithm ensures authentication, is therefore at least debatable.

6 Results

In this section, we describe our observations and findings from the experiments. We classify the attacks with regards to the security goals they are mainly attacking: Confidentiality, Integrity, and Availability (see Section 3.2). Some attacks attack more than one security goal directly or have the purpose of attacking another security goal by extension, that is, the attack becomes a means for another attack. These dependencies are explained accordingly.
In Section 6.1 we discuss attacks on the confidentiality of the ethernet and ZigBee traffic. We investigate how much can be learned about the LSH devices, the network, and the external services it uses. The attacks on the ethernet traffic are limited to passive listening and the analysis of those measurements. The ZigBee attacks include sniffing and packet injection of different kinds of packets. In Section 6.2 we discuss how the functioning of the devices can be manipulated. These attacks are all in the ZigBee domain and are mainly based on packet injection. In Section 6.3 we discuss the availability of the LSH devices under different circumstances and how an adversary may infringe upon this security goal.

6.1 Confidentiality

Here we describe our findings based on passive measurements of the ethernet traffic. As described in Section 3.4, we capture traffic using Wireshark. The findings mostly concern the user's privacy towards the third-party service providers that run the background services needed to use the LSH products. In Section 6.1.2 we describe the results of the confidentiality related attacks on ZigBee.

6.1.1 Ethernet

In this section, we describe the findings in the ethernet domain. We describe the first setup of the gateway and the phone and the traffic that is generated by pressing the doorbell or opening it. Then we summarize Attack Scenario A1.

Initial setup (Setup configuration S1, Attack scenario A1, passive on ethernet, devices: gateway)
We see in the literature [22][25] (see Section 4.1) that it is important for the security of IoT devices that TLS/SSL is used to secure ethernet traffic. We observe that during initial setup, the gateway immediately starts downloading a software update from `fireware-weaz.tuyaeu.com`, which is a server in Amsterdam (see Appendix A.1). A screenshot of the phone during the firmware update process is shown in Figure 14. In Wireshark, we see that all traffic is TLSv1.2 encrypted. To the best of our knowledge, this is reasonably secure, under the assumption of best practices with regards to TLS. We do not investigate the security of the TLS implementation any further, because our aim on the ethernet domain is to assess the privacy-friendliness of the LSH products. We investigate what information cloud service providers can see and not what information a potential malicious machine-in-the-middle could retrieve.
After setup, the gateway keeps sending packets to the limited broadcast address `255.255.255.255` every five seconds. This UDP packet contains 188 bytes of data, which seems to be a heartbeat. The payload of the packets is exactly the same for many packets in a row, as can be seen in Appendix A.2, and changes rarely. We expect that the content describes, among other things, details about the network, such as the topology, but we do not know this for sure. Therefore we expect that changes in the content of the broadcast packet are triggered by changes in the network. As described in Appendix A.2, after a DHCP sequence has been sent, the information in the packet changes to a new value and then stays that way. As the packets are only sent to the limited broadcast address, they stay within the local area network. This means that only devices within this network (usually a private home network) will be able to capture these packets. From a privacy perspective, we can therefore conclude, even if the content should be privacy sensitive, that there is no big loss of privacy, as only users within the local network could potentially read it.

Figure 14: Screenshot of the phone whilst upgrading the gateway's firmware (in Dutch)

Push notifications (Setup configuration S2, Attack scenario A1, passive on ethernet, devices: gateway, doorbell, motion sensor)
In this subsection, we describe the ethernet-related findings regarding the push notifications following pressing the doorbell and triggering the motion sensor. The ZigBee-related findings for this scenario are described in Section 6.1.2.
In our test bed there are two devices that generate push notifications when they are being triggered: the doorbell and the motion sensor. We trigger these devices and analyze the resulting ethernet traffic from those actions. To learn about the communication on ethernet level and the different (third) parties involved, we trigger events, in our this case by using the doorbell and the motion sensor, such that we know when to look for traffic related to user action. Knowing which parties are involved in processing the user information is important for privacy reasons.
By pressing the doorbell, we trigger the gateway to send a packet to a Microsoft server in Amsterdam, NL. In Appendix A.3 we see the default broadcasted ping every five seconds. In between those, at 3.9s, the doorbell is pressed. Packet no. 49 is the packet from the gateway (`10.42.0.13`) to the Microsoft server (`52.157.250.43`) that tells that the doorbell has been pressed. The Microsoft server sends one packet back and the gateway answers with an ACK. 0.6 seconds after the ACK is sent, the phone (`10.42.0.42`) receives a packet from an Apple server (`17.57.146.166`, in Cupertino, US), which is the push notification. The phone sends an ACK and an extra packet, for which it receives an ACK as well. Then, the phone sends a few requests to an Akamai server in Amsterdam, NL and gets answers to those. Figure 15 shows a schematic flow chart of the different entities that contact each other via ethernet after a device has been triggered. Because all packets are TLS encrypted, we can not read their contents. What stands out is that there is not a single packet directly sent between the mobile phone and the gateway within the local network, which would technically be enough if the phone is inside the local network. For the process from pressing the doorbell to creating a push notification, there are always remote servers involved in the communication.
Although the gateway notifies a Microsoft server of the doorbell being pressed, the Apple servers are the ones sending the push notifications to the phone on port 5223, which is a port used for Apple push notifications [51]. Thus, somewhere there has to be a connection between the Microsoft server and Apple server, potentially via a third party. Because we use an iPhone, it is not surprising that the notification comes from an Apple server. We would expect to see a different server (probably Google) contacting the phone if it was for example an Android device.

Half a second after receiving the notification from the Apple server, the phone contacts an Akamai server. The fact that the URLs contain the subdomain `static`, suggests that these are static files (like for example images or scripts) that are used in the app. We observe that the packets are no larger than 90 bytes and contain no more than 19 bytes of encrypted data. This practically rules out any 'serious' files. Due to TLS encryption, we can not read the content of the packets and can therefore not identify with certainty what the purpose of these packets is. We observe that the packets from Akamai only get sent the first time we trigger a push notification. From then, only the Microsoft and Apple servers are involved in generating a push notification.

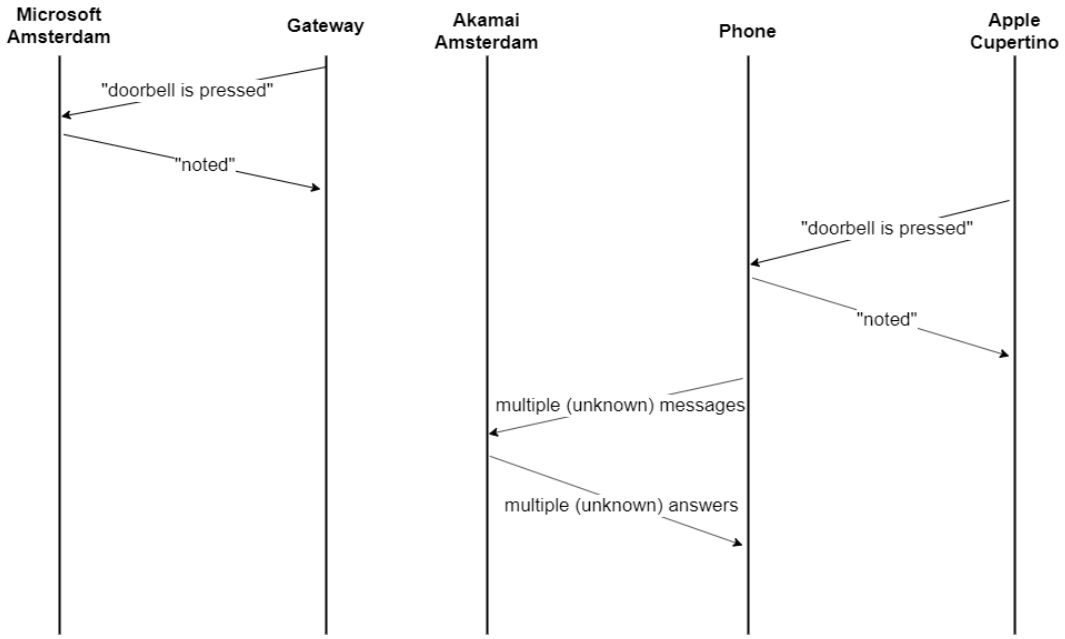


Figure 15: Schematic flow chart of ethernet communication between different entities

The other device that generates push notifications is the motion sensor. It creates a push notification every time something moves in front of it. We observe that the ethernet communication for these push notifications is the same as when the doorbell is pressed. From the end devices, we test (see Section 3.4.1), the motion sensor and doorbell are the only ones that trigger push notifications.

Manipulation warning (Setup configuration S2, Attack scenario A1, passive on ethernet, devices: gateway, doorbell, motion sensor)
In this subsection we describe the ethernet-related findings after opening a device. The ZigBee-related findings for this scenario are described in Section 6.1.2.
The doorbell and the motion sensor can be opened, which is for example necessary to switch the battery. In case a device is active while it is opened, the phone displays a push notification warning the user that the device has been manipulated, which we describe in Section 3.4.1 (see Appendix D.5 for the ZigBee packet that is sent after opening a device and Appendix E for the resulting push notification). We observe that the ethernet communication between the gateway, the Microsoft server, the Apple server, and the phone is almost identical as compared to the scenario where the doorbell is pressed. We see that no content is downloaded from the Akamai server, which seems to confirm that the communication with that server after pressing the doorbell the first time, is indeed static data.
From the other LSH devices (see Section 3.4.1) only the remote control can be opened. This does, however, not generate a warning, which is the expected result, as there is (contrary to the motion sensor and doorbell) no button triggered by opening it.

Summary Attack scenario A1
By manual inspection of traffic captures, we find that during setup and normal usage, there are several service providers contacted by the gateway, mainly Tuya, Microsoft, and Apple. Because they are contacted every time a device is controlled via the app, settings are changed or a notification has to be sent, these services have access to the information on what devices are installed and how they are used within a certain network. This is a great loss of privacy for the user towards these service providers, as these providers will always know at what time which devices are used and what their status is. By using the LSH products, the user has to implicitly accept this, as it is not possible to set up the network and fully use the devices' functionalities without an internet connection and the direct involvement of these services (see Attack scenario A7 in Section 6.3).

6.1.2 ZigBee

In Section 6.1.1 we see the results of the analysis of ethernet traffic (Attack scenario A1). In this section we discuss the results for attacks on the confidentiality of the ZigBee communication with the Attack scenario's A2 and A3, which are passive sniffing and sending beacon requests (see Section 3.3).

Passive Sniffing (Setup configuration S2, Attack scenario A2, passive on ZigBee, devices: gateway, doorbell, motion sensor)
We use whsniff in combination with the CC2531 to listen to ZigBee traffic (see Section 3.4.2) and learn about the network and its devices. As described in Section 5.2.1, ZigBee has 27 different channels, from which the LSH products support the channels 11 to 26, which are in the 2.4 GHz spectrum. By iterating through all channels while the gateway is active, we see that it uses channel 20, which uses the 2450 MHz band.
By listening to the ZigBee traffic on channel 20, we can see that some devices send packets to the broadcast address `0xfffc` roughly every 18 seconds. As the packets are encrypted, they can not be fully read without the correct key. As described in Section 3.4.2, Wireshark offers to enter encryption keys that are automatically used by Wireshark to decrypt ZigBee traffic. Entering the Global Key "ZigBeeAlliance09" does not make the 'normal' traffic readable, as it is encrypted using the NWK Key. We are, however, able to see the short source and destination addresses of all packets as part of the MAC Layer and NWK Layer. If the packet is encrypted, we also know the extended source address because it is included in the Security Header.
In the Section 'Learning the NWK Key and Trust Center Link Keys' hereafter, we learn the NWK Key and are then able to decrypt the messages that are regularly sent to the broadcast address `0xfffc`. These packets turn out to be 'Link Status' packets. They are sent by all devices that have router functionality, that is, they can forward packets. The Link Status packets contain information on the links (neighbor nodes that are also routers) that certain device has. For example, if the gateway and power socket are both active, they both regularly send a Link Status packet in which they report that they are connected.

Learning the NWK Key and Trust Center Link Keys (Setup configuration S3, Attack scenario A2, passive on ZigBee, devices: gateway, doorbell, motion sensor, light bulb, remote, power socket)
We know from the ZigBee specification [11] that "during initial key transport [there may be] a brief moment of vulnerability where the key could be obtained by any device" [11]. The specification offers the possibility to use Install Codes to mitigate this vulnerability. We see, however, no mention of Install Codes in any manual for the LSH products, nor on the products themselves or in the app. It is therefore likely that this vulnerability is present in the LSH devices.
To test this, we listen on channel 20 using the CC2531, Whsniff and Wireshark (see Section 3.4.2) while adding a device, in this case the motion sensor, to the network (see Appendix B.2). We see that the motion sensor starts by sending Beacon Requests, which are packets with only a MAC Layer, to a broadcast address (`0xffff`). The coordinator responds with Beacons. The 'Permit Join Request' sent by the coordinator can not entirely be read by the motion sensor, as it is encrypted on the NWK

Layer with the NWK Key. After some Beacons, the motion sensor sends an (unencrypted) 'Association Request', to which the coordinator answers with an (unencrypted) 'Association Response', in which the coordinator assigns the motion sensor to a short address. Then, the coordinator sends a 'Transport Key' packet (see Appendix D.6) in which the NWK is included. This packet is encrypted only on the APS Layer. Because Wireshark automatically decrypts ZigBee traffic if the correct keys are stored in the settings (see Section 3.4.2) and we have stored the Global Key, we know that the key used to encrypt this packet is indeed the Global Key. This means that anyone capturing this packet knows the NWK Key of this network.
Five seconds after the key exchange, the same 'conversation' with the Association Request/Response and Key Transport takes place again. We do not know why this sequence of packets is sent twice. Three more seconds later the motion sensor sends a 'Request Key' (see Appendix D.7) packet in which it requests a Trust Center Link Key, which is meant for end-to-end encryption between this device and the Trust Center, which is the coordinator. The coordinator answers with a 'Transport Key' packet (see Appendix D.8) in which the Trust Center Link Key is included. The Request Key and Transport Key packets are encrypted twice: on the NWK Layer using the NWK Key and on the APS Layer using the Global Key. The confidentiality of these packets is thus only dependant on knowledge of the NWK Key, as the Global Key is publicly known.
The motion sensor tries to verify the correctness of the Trust Center Link Key by sending a hash of the key to the coordinator. The hash function used for this is MMO with AES-128 as primitive (see Section 5.3). This packet is only encrypted on the NWK Layer with the NWK Key. The 'Confirm Key' packet, which confirms the correctness of the Trust Center Link Key is sent by the coordinator and is encrypted on two layers. On the NWK Layer, it is encrypted using the NWK Key, on the APS Layer it is encrypted using the newly established Trust Center Link Key.
We repeat this experiment with all other devices and manage to capture the same sequence of packets, including the NWK Key being sent twice, for all devices. This way, we know the NWK Key and all established Trust Center Link Keys (see Section 5.2.5). This again confirms that the LSH products do not use Install Codes for key transport, but instead use the Global Key to encrypt the initial transport of the NWK Key, which enables anyone listening to the process of adding any device to the network to learn the NWK Key.

We observe that Trust Center Link Keys are hardly used. In 'normal' circumstances where the device is operated as can be expected in all-day use, this key is never used. We only see packets being encrypted on the APS Layer when a device is being added to the network. Then, a link key is established, distributed, and verified. Figure 16 shows all packets that are encrypted with a link key during a time frame of more than 9 minutes in which the power socket has been added to the network and normally used by turning the sockets on and off.

zbee_aps.security==1

No.	Time	MAC Src	NWK Src	MAC Dst	NWK Dst	MAC SN	NWK SN	Sec Ctr	APS Ctr	ZCL TSN	Length	Info
125	259.514620	0x0000	0x0000	0x9fa9	0x9fa9	123	103	4097	177		73	Transport Key
136	262.911627	0x0000	0x0000	0x9fa9	0x9fa9	126	105	4098	178		73	Transport Key
202	265.160263	0x9fa9	0x9fa9	0x0000	0x0000	127	127	16406,12288	180		58	Request Key
206	265.168976	0x0000	0x0000	0x9fa9	0x9fa9	143	127	49331,4099	188		90	Transport Key
212	265.191395	0x0000	0x0000	0x9fa9	0x9fa9	145	129	49333,4100	189		67	Confirm Key, SUCCESS
214	265.197659	0x9fa9	0x9fa9	0x0000	0x0000	130	130	16409,12289	189		48	APS: Ack

Figure 16: All APS encrypted packets of a 9 minute session

We see that these packets are the earlier mentioned transport and verification of the Trust Center Link Key. Within that process, also one 'APS ACK' is sent. As opposed to the IEEE 802.15.4 ACK, this ACK is encrypted on both the NWK Layer and APS Layer.

Push notifications (Setup configuration S2, Attack scenario A2, passive on ZigBee, devices: gateway, doorbell, motion sensor)
In this subsection, we describe the ZigBee-related findings regarding the push notifications following pressing the doorbell and triggering the motion sensor. The ethernet-related findings for this scenario are described in Section 6.1.1.
By pressing the doorbell, we trigger a push notification on the phone as described in Sections 3.4.1 and 6.1.1. The origin of that notification in the ethernet domain is the gateway, as it is the only LSH device that communicates via ethernet. It gets told that the doorbell has been pressed by a ZCL 'IAS Zone Status Update' packet (see Appendix D.5). This packet is sent by the doorbell and is encrypted on the NWK Layer with the NWK Key.
As we see in Section 6.1.1, the push notification for motion that is registered by the motion sensor is generated by the same kind of traffic in the ethernet domain. The same holds for the ZigBee communication: the gateway gets also told by the motion sensor that motion has been registered with an 'IAS Zone Status Update'. The specific bits, similarities, and differences between the update packets of the two devices are further explained in Appendix D.5.

Manipulation warning (Setup configuration S2, Attack scenario A2, passive on ZigBee, devices: gateway, doorbell, motion sensor)
In this subsection we describe the ZigBee-related findings concerning opening a device. The ethernet-related findings for this scenario are described in Section 6.1.1.
The 'IAS Zone Status Update' packet is not only used to notify the gateway that the doorbell has been pressed or the motion sensor registered motion. By setting the `Tamper` bit in the `ZoneStatus` field to `1`, it tells the gateway that the device is open or has been opened. The exact specifications are explained in Appendix D.5.
By opening a device (doorbell or motion sensor), a button on the inside/backside of the device is released, which triggers the device to send an 'IAS Zone Status Update' upon which the gateway warns the user that the device is being manipulated (and eventually being stolen). By closing the device, that same button is pressed again, which also triggers an 'IAS Zone Status Update', but with the `Tamper` bit set to `0`. This then triggers the push notification that the "device is ready".

Sequence numbers (Setup configuration S3, Attack scenario A2, passive on ZigBee, devices: gateway, doorbell, motion sensor, light bulb, remote, power socket)
There are essentially five sequence numbers that a device uses and thus that all devices need to keep track of for all other devices in the network. These are the MAC SN, NWK SN, Security Frame Counter, APS Counter, and ZCL SN (see Section 5.2.4). The first three are sent in plaintext. The APS Counter and ZCL SN are a part of the payload that is encrypted on the NWK Layer using the NWK Key. Examples of this can be seen in Appendices D.2, D.3 and D.5.
All counters except the NWK SN are always incremented by 1 whenever they are used. The NWK SN is incremented by 1 or by 2, depending on the presence or absence of the APS Layer. If there is no APS Layer, the NWK SN is incremented by 1, if there is an APS Layer, it is incremented by 2. Sequence numbers can have multiple instances per device, as we explain in this section. The MAC SN has two instances: one for packets with a NWK Layer and one for packets without a NWK Layer. These instances co-exist within the device. This means that, depending on what kind of packet is sent, one of the counters is chosen and its next value is put in the place of the MAC SN. The Security Counter also has two instances: one for the NWK Layer and one for the APS Layer. These are the only two layers on which we see encryption being used.
The ZCL Sequence Number also has more than one instance per device. Depending on the packet, one of the two instances is chosen and put in place of the ZCL SN. An example of this can be seen in Appendix B.1. We see that for certain payloads (such as IAS Status Zone updates) another counter is chosen. However, we can not identify a clear determining factor, for example, a specific field with a certain value, which tells which counter is used.
The ZigBee protocol is built to support tree and mesh structures [11]. Devices that act as a router

forward messages if the packet is meant for a device (child node) that is connected to the network via that router. We see that from all tested devices (see Section 3.4.1), the lamps and power socket are the end devices that also act as a router and thus forward packets if a device is connected to the network via them. Specifically, these two products act as a router because they are not battery powered but grid powered and therefore are less resource-constrained. Packets get forwarded by a router in either of two cases: the packet is either specifically meant for the child node or it is sent to a broadcast address (`0xffff`, `0xfffd`, `0xfffc` or `0xfffb`). In case a packet is forwarded that contains all five sequence numbers, some of them are copied, whereas some are exchanged by the forwarding device. In Figure 17 we see a command (packet no. 662) from the remote control (`0xabd8`) being sent to the lamp (`0x8471`) on the MAC Layer and to a broadcast address (`0xfffd`) on the NWK Layer.

No.	Time	MAC Src	NWK Src	MAC Dst	NWK Dst	MAC SN	NWK SN	Sec Ctr	APS Ctr	ZCL TSN	Length	Info
662	483.437600	0xabd8	0xabd8	0x8471	0xfffd	24	45	12361	14	68	49	ZCL OnOff: Off, Seq: 68
663	483.439537					24					5	Ack
664	483.477717	0x8471	0xabd8	0xffff	0xfffd	228	45	26074	14	68	49	ZCL OnOff: Off, Seq: 68
665	483.486686	0x0000	0xabd8	0xffff	0xfffd	66	45	146569	14	68	49	ZCL OnOff: Off, Seq: 68

Figure 17: A ZCL command from the remote gets forwarded by the lamp and the coordinator.

Packets no. 664 and 665 are forwarded packets sent by the lamp and the coordinator (`0x0000`), as can be seen at the MAC Layer source address. The APS Counter and the ZCL Transaction Sequence Number are copied from the original packet and thus match the counters from the source device as stated on the NWK Layer. The MAC sequence number, NWK sequence number, and Security counter are not copied, instead, they match the counters from the source device on the MAC Layer. The fact that the Security counter gets changed when the packet is forwarded, means that the payload of the packet (thus the whole APS and ZCL layer) needs to be decrypted and encrypted again with the new Security counter and a different extended source address that matches the new MAC Layer source.

All this information on the different sequence numbers is not the result of an attack, other than simply listening. It is however, necessary information to perform and understand the Attacks A5, A10, and A11.

Summary attack scenario A2
By passively listening to the ZigBee traffic at the 'right' moment, which is when the user adds a device to the network, the NWK Key can be learned by anyone within range (ca. 70m) of the gateway. This is because the LSH devices do not offer the possibility of using Install Codes, which means that for the transport of the NWK Key, the global key is used, which is publicly known and therefore offers no confidentiality whatsoever.
The Trust Center Link Key transport packet is also sent during addition to the network and is encrypted with the NWK Key and Global Key. Therefore it can also be learned alongside the NWK Key during that process. However, the key is hardly used. Almost all encrypted traffic is encrypted using the NWK Key. As there is no forward secrecy protection, an attacker can capture any (encrypted) traffic without any key knowledge and decrypt it later when she learnt the NWK Key.
By capturing 'IAS Zone Status Update' packets, we know when the devices let the gateway know that they registered an event (pressing/motion) or that they are being manipulated (opened/closed). We also use these captured packets to forge and send custom commands (see Attack scenario A5 in Section 6.2) that trigger push notifications. These 'IAS Zone Status Update' packets also tell an attacker what kind of devices are used within a network. As explained in Appendix D.5, we can distinguish a motion sensor from a doorbell by looking at the `Restore reports` bit.
Without knowledge of the NWK Key, an attacker can read three out of five sequence numbers: MAC Sequence number, NWK Sequence number, and Security Frame Counter. With knowledge of the NWK Key, it is also possible to read the APS Counter and ZCL Sequence number. We see that encrypted packets that get forwarded, are decrypted by the forwarding device and then again encrypted with the

corresponding Security Counter. The APS Counter and ZCL Sequence number, however, remain the same.

Beacon Request (Setup configuration S3, Attack scenario A3, active on ZigBee, devices: gateway, doorbell, motion sensor, light bulb, remote, power socket)
We find that any device, both authenticated or unauthenticated, can send a Beacon Request, which is a MAC Layer-packet without a source address. To proactively learn about the network and its devices, we send a beacon request (see Appendix D.1). We observe that any device in the network that receives it, responds with a Beacon including the short and extended PAN ID, short source address, protocol version, device depth (within the tree of the network topology) and whether it has router capacity (that is, it forwards packets to extend the network reach) and end device capacity (that is, it does more than *just* forwarding). Because there is no encryption used for confidentiality or authentication, anyone can send beacon requests and receive beacons. As a result, anyone with ZigBee compatible hardware can create a topology of a ZigBee network by sending a beacon request, regardless if it's a self-owned network or someone else's. This can have consequences for the user's privacy, as anyone can learn at any moment how a certain network is built with regards to its topology. Users have to be aware that the network topology of a ZigBee network should therefore always be considered public knowledge. If for an IoT application 'hidden' devices are needed, ZigBee should not be chosen as the communication protocol.

Beacon requests are packets that have no ZigBee NWK Layer but only consist of a MAC Layer, which is defined in the IEEE 802.15.4 standard [6]. The 6LoWPAN protocol, which we briefly describe introduction (see Section 1, is also based on the IEEE 802.15.4 protocol [10]. We expect that this attack also works on other networks, whose communication protocol is based on IEEE 802.15.4, including 6LoWPAN. In Section 9 we propose future work based on the fact that problems from a certain protocol could also exist in a different one, based on the fact that they share a low-level protocol.

6.2 Integrity

In this Section we discuss the results for the Attack scenarios A4 and A5. These are attacks on the integrity of the ZigBee communication and the LSH devices.

Command injection without NWK Key knowledge (Setup configuration S4, Attack scenario A4, active on ZigBee, devices: gateway, doorbell, motion sensor, power socket)
We craft command packets and send them to the devices to manipulate their behavior. Here we describe the forged packets that we send, the expected results *if* the attacks succeed, and the actual results. The objective is to test how the devices react to packets for which an attacker does not need to have any key knowledge, in particular the NWK Key.

- *Unencrypted network leave:* the plan behind this attack is to deauthenticate a device, in this case, the doorbell, from the network. To use the device again, the user would have to add it to the network again, which would lead to the coordinator sending the NWK Key again, which can then be learned by an adversary. For this attack, we assume no key knowledge, as the purpose is to learn the NWK Key. Therefore we forge an unencrypted NWK Layer 'Leave' command, as shown in Appendix D.9. By using Attack A2, we know how these packets look like in their encrypted form. By using this knowledge in combination with the Scapy framework, we are able to craft unsafe variants of those packets by slightly changing the header (namely flipping the security bit) and removing the security sub-header. The content of the packet remains the same.

  We observe that the coordinator does not respond to the forged packet and that the doorbell can still be used normally, that is, pressing it still results in a push notification on the phone. The doorbell is thus not forced out of the network.

- *Unsafe rejoin requests:* the ZigBee protocol [11] offers the possibility to enter a network unsecured, that is without the use of any encryption. It is, however, not possible to join a network as a new device without the user. To join a network, the coordinator needs to be in pairing mode, which only happens if the user adds a device using the app. Therefore we send a rejoin request on behalf of a device that has already joined the network before, in this case, the motion sensor. To test if an unsecured rejoin is possible, we send an unencrypted rejoin request on behalf of the motion sensor to the coordinator. If the attack succeeds, we expect to see packets being sent from the coordinator to the motion sensor unencrypted. However, after sending the packet in Appendix D.10, the coordinator's only response to this is an ACK on the MAC Layer. There is no response on the NWK Layer, from which we conclude that the unsafe rejoin attack did not succeed.

- *Use global key as NWK Key:* because the devices seem to reject any unencrypted command, we also try a different approach: encrypt a packet with the key known to both an adversary and a target device: the Global Key, defined in the ZigBee standard [11] as "ZigBeeAlliance09". The kind of key used for encryption is set in the Security Control Field, which is the first byte of the Security Header. The options are `0b00` (Link Key), `0b01` (Network Key) and `0b10` (Key Transport Key). For every possible value of the 2-byte Key ID, we send a turn-off command encrypted with the Global Key to the power socket. None of the packets get accepted, as the device does not turn off any socket. We do, however, receive ACKs on the MAC Layer, meaning that the device received the packets.

We see that the devices do not react to commands in ZigBee packets that are unencrypted. They expect these commands to be properly encrypted with the NWK Key. This is, from a security perspective, the desired behavior of the devices, as only a device knowing the NWK Key can send proper commands. From this follows that, in the ZigBee protocol, knowledge of the NWK Key is not only used for confidentiality but also for authorization purposes. Because the three attacks described here are properly mitigated, these three attacks do not infringe upon the integrity of the devices.

Command injection with NWK Key knowledge (Setup configuration S5, Attack scenario A5, active on ZigBee, devices: gateway, motion sensor, light bulb, remote, power socket) We see in Attack scenario A4 that unencrypted commands will not get executed by devices and that there is a possibility to learn the NWK Key, as described in Section 6.1.2, Attack scenario A2. The attacks in this section aim at altering the behavior of the devices with the assumption that the adversary knows the NWK Key. We test this to know whether knowledge of the NWK Key is enough to alter the behavior or other information is needed to do so. Therefore we forge commands that are properly encrypted with the NWK Key.

- *Turn off the power socket:* To turn off the power socket, we capture, modify, encrypt and send a 'turn-off' command to the power socket. In this attack, we impersonate the coordinator. That means that we use the coordinator's source addresses (short and extended) and its sequence numbers. We need all five sequence numbers to be correct: the MAC Sequence Number, NWK Sequence Number, Security Frame Counter, APS Counter, and ZCL Sequence Number. The first three can be read without any key knowledge. The last two are encrypted as part of the payload on the NWK Layer and therefore knowledge of the NKW Key is needed to read them, as explained in Section 6.1.2.

  The ZCL Sequence Number has more than one instance, as explained in Section 6.1.2. Thus, to send a command with the correct ZCL SN, it is necessary to wait and listen until the coordinator sends at least one genuine command in which the 'other' ZCL SN is included. This happens for example if the user turns on or off a power socket herself using the app.

  If all five sequence numbers are incremented with respect to their latest seen value, the packet can be encrypted with the NWK Key and sent to the power socket. The forged packet is shown in Appendix D.2. We observe that sending this packet suffices to turn on and off the specified

sockets in the power socket bar at will. We, therefore, conclude that knowing the NWK Key and catching at least one genuine command suffices to control the power socket.

Although the state of the socket can be modified by an attacker, the user can easily switch back the state herself in the app. Therefore, just controlling the power socket does not result in a Denial of Service, but it is an infringement on the integrity of the device.

This attack only works against a target device for which the attacker captures a genuine command, as the sequence numbers (in particular the ZCL SN) need to be known to the attacker for that specific device. After one genuine packet is captured and therefore all sequence numbers are known, the attacker can continue controlling the target device with multiple commands by continuously increasing the sequence numbers after each forged command. As we see in Attack A10, the genuine sender of the original command will get out of sync with the target device, which results in a (temporary) Denial of Service, depending on how many sequence numbers have been skipped (see Section 6.3).

- *Trigger a push notification by imitating movement at the motion sensor:* for this attack we capture, modify, encrypt and send a 'ZCL IAS Zone Status Update' packet (see Appendix D.5) that is usually sent by the motion sensor when it is triggered by the motion of something physically in front of it. Scapy [56], which we use for building packets, does not have a class to parse or build the 'IAS Zone Status Update' ZigBee packet. As previously described in Section 3.4.3, we present an implementation of this packet for Scapy in Appendix F. This implementation enables us to more easily modify the packets and use them in this attack.

  To send this packet, we need to have all five sequence numbers and both source addresses correctly to imitate the motion sensor, as is the case with the packet that turns off the power socket. For this attack, we also need to capture at least one genuine packet to read the correct 'other' ZCL SN value. By encrypting the modified packet with the NWK Key and sending it to the coordinator, we are able to trigger a push notification on the phone which says that the motion sensor registered (physical) motion. This is an infringement on the integrity of the LSH network, as the functionality is tampered with.

  By further increasing the sequence numbers and sending more packets, arbitrarily many push notifications can be triggered. This could for example lead to a user getting very annoyed by the many notifications, after which she might decide that the motion sensor is 'broken' and thus she disables the motion sensor by taking out the battery. Therefore sending many push notifications can, by extension, lead to a denial of service. There is, however a more direct way to force a DoS as a result of triggering push notifications, as we explain next.

  By repeatedly sending forged packets that result in a push notification with increasing sequence numbers, we force the stored sequence number that the gateway has about the motion sensor to go up. We observe that sending these packets does not change the counters on the motion sensor, it does thus not adapt to this. This effectively results in a temporary Denial of Service, as the motion sensor keeps increasing the sequence numbers as usual, which means that the packets get rejected by the gateway because the sequence numbers are too low. A DoS period of one minute can be created by already increasing all sequence numbers by 10 (and the NWK Sequence Number by 20). The longest DoS we observe takes more than four minutes to resolve. During that period, although the motion sensor is registering motion and keeps sending 'IAS Zone Status Updates', the phone shows not a single push notification, from which we conclude that the gateway rejects the packets sent by the motion sensor. This temporary DoS attack inspires to take increasing the sequence numbers to the extreme, which we do in Attack scenario A10, the results of which are explained in Section 6.3.

- *Turn on and off the light bulb by imitating the remote control:* for this attack we capture, modify, encrypt and send a 'ZCL On/Off' packet that is usually sent by the remote control (see Appendix D.3). At this point, the remote control has been assigned to the lamp by the user using the app.

The remote control thus controls the brightness of the lamp and it can turn it on or off. By adjusting the five sequence numbers accordingly and encrypting the packet with the NWK Key, we can turn on and off the lamp to our liking. We see that our packets get forwarded by the router-enabled devices, just like the genuine packets from the remote (see Figure 17).

- *Increase sequence numbers to test resilience to packet loss:* in Section 6.1.2 we describe the different sequence numbers in the ZigBee protocol that an outside attacker can see. To test the resilience of the network against incidental packet loss, we again send 'ZCL On/Off' packets (see Appendix D.3) that we properly encrypt, but with all five sequence numbers overly increased. We increase the sequence numbers such that we skip up to 24 values, which simulates the loss of 24 packets in a row. In Figure 18 we see a genuine packet (no. 35) from the remote. The packets 665, 793, 853, 873, 946, 1073, and 1244 are forged packets with sequence numbers that have an increased gap to their previous value. The packets are sent to the gateway (`0x0000`) and are then forwarded as described in Section 6.1.2 (not shown in Figure 18). All commands get executed and have therefore been accepted by the gateway. From this, we conclude that the network is resilient to incidental packet loss, regarding the sequence numbers.

No.	Time	MAC Src	NWK Src	MAC Dst	NWK Dst	MAC SN	NWK SN	Sec Ctr	APS Ctr	ZCL TSN	Length	Info
35	20.129662	0xea57	0xea57	0x0000	0xfffd	68	147	16461	176	54	49	ZCL OnOff: Off, Seq: 54
39	20.230028	0xea57		0x0000		69					12	Data Request
45	20.331761	0xea57		0x0000		70					12	Data Request
49	20.434009	0xea57		0x0000		71					12	Data Request
51	20.939995	0xea57		0x0000		72					12	Data Request
665	536.310946	0xea57	0xea57	0x0000	0xfffd	73	149	16462	177	55	49	ZCL OnOff: On, Seq: 55
793	617.261810	0xea57	0xea57	0x0000	0xfffd	75	153	16464	179	57	49	ZCL OnOff: Off, Seq: 57
853	657.832015	0xea57	0xea57	0x0000	0xfffd	78	159	16467	182	60	49	ZCL OnOff: On, Seq: 60
873	673.300099	0xea57	0xea57	0x0000	0xfffd	83	169	16472	187	65	49	ZCL OnOff: Off, Seq: 65
946	726.265455	0xea57	0xea57	0x0000	0xfffd	92	187	16481	196	74	49	ZCL OnOff: On, Seq: 74
1073	822.979768	0xea57	0xea57	0x0000	0xfffd	110	223	16499	214	92	49	ZCL OnOff: Off, Seq: 92
1244	963.749361	0xea57	0xea57	0x0000	0xfffd	135	17	16524	239	117	49	ZCL OnOff: On, Seq: 117

Figure 18: ZCL commands with overly increased sequence numbers

We see that knowing the NWK Key suffices to read all packets sent within the ZigBee network for normal use of the devices' functionalities such as switching a bulb's color, receiving a notification of motion being sensed, or turning off a power socket. The attacker can read all sequence numbers and anticipate those to then properly encrypt and send (modified) commands, which get executed at the receiving device, enabling an attacker to control devices at will.
By triggering many push notifications on the user's phone, the user might get annoyed and decide to disable the respective device herself, which turns this attack into a possible Denial of Service attack. However, there is also the possibility for a temporary Denial of Service attack: if the attacker sends multiple commands with increasing Security counter, APS Counter, and ZCL Sequence number, these counters get increased at the gateway as well, which results in the coordinator temporarily dropping all genuine packets, until the sending devices catches up with the incremental of the sequence numbers. This attack, which is originally designed to attack the integrity of the devices, then becomes an attack on the availability (a more sophisticated attack on availability resulting from this is Attack scenario A10, as discussed in Section 6.3).
For these reasons, we conclude that leakage of the NWK Key, as described in Section 6.1.2, leads to a severe compromise of all devices within the network as devices can be controlled by an adversary and push notifications can be triggered arbitrarily. Therefore, the network's security with regards to its integrity has to be considered completely broken.

6.3 Availability

In this section we discuss the results for the Attack scenarios A6, A7, A8, A9 and A10. These are attacks on the availability of the LSH products.

PAN ID conflict (Setup configuration S3, Attack scenario A6, active on ZigBee, devices: gateway, doorbell, motion sensor, light bulb, remote, power socket)
The PAN ID is the identification number of a ZigBee network. It is included in the header of virtually every packet on the MAC Layer and allows for multiple distinct ZigBee networks within a certain physical area. The goal of this attack is to test the feasibility of creating a PAN ID conflict and learn about the resolving process.
We cause a PAN ID conflict in the network by sending a Beacon Response with the correct short PAN ID, which is visible in almost every packet sent within the network, and a non-matching (pseudo-random) extended PAN ID (see Appendix D.1). We observe that, to resolve the issue, devices send a Network Report PAN Identifier Conflict packet, which contains the short and extended source addresses of the device and the short and (correct) extended PAN ID.
As Akestoridis et al. [28] stated, the solving process would have to be jammed in order to force devices out of the network (see Section 4.2). Jamming, however, is on Layer 1 (physical layer), which is out of the scope of this research due to a lack of the necessary hardware. Because we do not jam the responses to our beacon, the Network Report PAN Identifier Conflict packets resolve the PAN ID conflict, which means that this attack does not result in Denial of Service of a device, which is the expected result. We can therefore confirm that a PAN ID can be created easily. However, due to the lack of hardware, we can neither confirm nor deny whether this attack, in combination with selective jamming, leads to a Denial of Service.

Disconnecting the gateway from the internet (Setup configuration S3, Attack scenario A7, active on ZigBee, devices: gateway, doorbell, motion sensor, light bulb, remote, power socket)
In the literature [16][19][23][22] we see that malware is often focused on IoT products, as they are more likely to be vulnerable due to the resource constraints inherent to the domain. 'Being connected to the internet' can therefore be considered a potential security threat. Disconnecting IoT products from the internet (or not connecting them in the first place) can therefore be desirable if the functionality is not impacted (too much) by this. Another reason to disconnect IoT products from the internet is because of the privacy of the user. In Section 6.1.1 we see that many services know exactly what devices are used at what time within the LSH network. To prevent this, offline usage of the LSH products may be desirable. However, this is only possible to some extent. Adding new devices to the network only functions while the gateway and the phone are both online. The devices may be in different subnets, that is, the phone may even be on the other side of the world, as long as they are both connected to the internet because the communication between phone and gateway is always going via the server. The same condition holds for controlling devices via the phone. For example, changing the color of a lamp, turning on the power socket, or receiving a push notification from the doorbell all require the gateway and the phone to be connected to the internet. It does not suffice to have them both connected to the same Local Area Network. This way, it is possible for the service providers to analyze behavioral patterns in the data generated by the phone and the gateway.
Although for configuring or controlling the devices an internet connection is required, the end devices keep working (for example shining or giving electricity) if the gateway is offline or even turned off. After configuring it accordingly in the app with the gateway being online, controlling the lamps with the remote (on/off/brighter/darker) works also offline. It is therefore not necessary to have the gateway connected to the internet to control some lamps. It is, however, not possible to change which lamps are controlled with the remote if the gateway is offline. This has to be set up in the app with the phone and gateway being online.
Setting automated schemes can also only be done as long as the gateway is connected to the internet. However, after setting it up, the automation still works, from which we conclude that the gateway stores

the rules locally in its memory.

The fact that devices cannot be added or configured without the gateway being online makes that the service providers (Microsoft/Apple) know exactly what devices the user has (see Section 6.1.1). By using the LSH products, the user thus implicitly has to accept that these external parties know what devices are set up in the network and how and when they are being used. This is a major loss of privacy for the user that comes with the usage of the LSH devices.
Automated schemes and the remote control still work properly if the gateway is offline. By carefully setting up the desired devices in a way such that offline usage suffices, the user can improve her privacy situation a bit. This way, the service providers only know which devices are used, but they are no longer able to see when precisely devices are actively being used.

Resetting Security Sequence numbers? (Setup configuration S3, Attack scenario A8, active on ZigBee, devices: gateway, doorbell, motion sensor, light bulb, remote, power socket)
Farha et al. [31] proposed some attacks that try to circumvent the replay attack mitigation mechanism: the sequence numbers. They proposed an attack with which it would still be possible to perform replay attacks, but these are based on a non-secured ZigBee network, that is, packets are not encrypted. This is not the case for our network and therefore this attack does not work. Farha et al. [31] also proposed a DoS attack that exploits the replay attack mitigation: the idea is to send a packet with a very large Security Counter, which would then get stored at the receiving end as being the latest used sequence number from the imitated device [31]. When the genuine device then sends packets, they will have lower sequence numbers, which means that they get dropped at the receiving end, therefore the genuine device will not be able to successfully communicate until it reaches the sequence number in the injected packet. It is assumed that the attacker does not know any keys used in the network. Therefore, for the attack to work, an attacker must first capture a genuine packet with a large Security Counter from the target device and after that, there needs to be a reset of the security counter. Farha et al. [31] state that the Security Counter will reset when the Coordinator restarts. During our experiments, we turned the LSH gateway, which acts as the ZigBee Coordinator, off and on multiple times and also disconnected it many times from the power grid over a period of three months. We observe that the Security Counter does not get reset, but instead keeps increasing, also after a restart of the gateway. This also holds for all other devices, which have also been turned off and on multiple times. We do not observe any reset of any Security Counter. This means that the proposed attack by Farha et al. [31] does not work on the LSH products.
We do, however, observe that in some cases, a new short address is given to an end device by the coordinator. To our knowledge, this has no security-critical implications.

Reset devices with dedicated button (Setup configuration S3, Attack scenario A9, active on ZigBee, devices: gateway, doorbell, motion sensor, light bulb, remote, power socket)
As explained in Section 5.1.3, all devices but the lamp can be disconnected from the network by pressing a dedicated button on the device itself. The lamp can be disconnected by turning it off and on three times within three seconds. Although it could seem silly to classify 'pressing a button' as an attack, we argue that in this case, it is because of its simplicity and the implications it has. We analyze this scenario to learn about the process of disconnecting a device from a network and adding it to the network again.
In this scenario, the attacker has to have direct physical access to a device. As most devices are designed for in-house use, getting physical access may not be trivial for an outside attacker. Therefore we focus on objects that are by their functionality implicitly likely to be used outside: the motion sensor and the doorbell. These two can be opened, as explained in Section 3.4.1, after which the disconnect button can be pressed.
Directly after pressing the disconnect button inside the doorbell three times, we observe a 'Leave' packet (see Appendix D.4) being sent from the doorbell to the gateway. Then the doorbell is disconnected from the network, which we confirm by pressing the doorbell on the front, which does not result in a

push notification nor an 'IAS Zone Status Update'. To add the device to the network again, the full procedure as described in Section 5.1.2 has to be followed. We observe that re-adding the doorbell to the network results in the same packets being sent as described in Section 6.1.2, Attack scenario A2, which includes the NWK Key transport packet, which is only encrypted with the Global Key and thus anyone listening can decrypt it.

Although this attack may look a bit silly at first, as it describes the intended functionality of the LSH devices, it is actually very powerful. Especially the doorbell and motion sensor are at risk to this attack, as they are by their functionality most likely to be used outside, where an attacker could easily obtain the necessary physical access.
This attack enables an attacker to trick the user into adding the device while the attacker is listening to the ZigBee traffic (Attack scenario A2) to learn the NWK Key. After this, the attacker would be able to control any device within the network (see Section 6.2, Attack scenario A5). Therefore, being able to disconnect a single device, which is a Denial of Service attack, can lead to the attacker being able to control all devices within the network. Therefore this attack does not only compromise the availability of a device, but by extension is also a risk for the integrity of the whole network.

Send command with very high Security Counter (Setup configuration S6, Attack scenario A10, active on ZigBee, devices: gateway, doorbell)
In Section 6.2 we see that by sending many properly encrypted commands with increasing sequence numbers, we are able to perform a temporary Denial of Service attack. This attack inspires to take the increasing of sequence numbers to the extreme. To test the feasibility of a denial of service attack, we discuss here what happens if a command with a very high Security Counter is sent.
We know from Section 6.1.2 (Attack scenario A2) and Section 6.2 (Attack scenario A5) that an attacker can forge properly encrypted packets that get accepted and executed by the receiving device. By sending a packet with a very high Security Counter, such as the maximum value minus 1 (`0xFFFFFFFE`) with the source address being the victim device and the destination address the coordinator, we force the coordinator to save that high Security Counter as being the last used value for the victim device. We use the doorbell as a victim device in this experiment and send an 'IAS Zone Status Update' packet (see Appendix D.5) with the high Security Counter in its name to the coordinator. When the doorbell then sends a message (with a much lower Security Counter) because it has been pressed, the packet will have a much lower Security Counter. As a result, the packet gets dropped by the coordinator, because it expects the Security Counter to have the next value (`0xFFFFFFFF`).
We observe that the devices (doorbell and coordinator) automatically try for circa one minute to resolve the issue, but they fail in doing so. We observe that during this attempt in resolving the counter mismatch, the Security Counter at the victim device gets increased by a value of 11 in ca. 15 seconds. By simple extrapolation, we see that, if in this attack such a large Security Counter is used, the issue will not get resolved in foreseeable time. The issue can, however, be resolved by the user by pulling the battery out of the victim device and putting it back in a few seconds later. We see that in that case the victim device sends a rejoin request, which gets accepted, after which the Security Counters are synced and the device functions normally again.
From this, we conclude that forging and sending a properly encrypted command with a very high sequence number is a very effective Denial of Service attack and therefore clearly infringes upon the availability of the LSH network. This attack is relatively persistent, as it does not resolve automatically. It can, however, easily be solved by the user if she has physical access to the victim device.

During this experiment, we also observe that it is possible to continue increasing the Security Counter until we reach the maximum value (`0xFFFFFFFF`) and then continue with the counter value `0`. The command with the low Security Counter gets executed just like the others before it. After that, we can increase the Security Counter by an arbitrary value, just like before. We find it remarkable that the gateway does not initiate a key rotation, that is, it does not establish a new NWK Key after the maximum value has been used. As a result, the same Security Counter values as before are used with

the same encryption key (the NWK Key). Therefore replay attacks, as described by Farha et al. [31], suddenly become possible.

Take the doorbell out of range (Setup configuration S6, Attack scenario A11, passive on ZigBee, devices: gateway, doorbell)
In Section 5.2.4 we explain the different kinds of sequence numbers in the ZigBee protocol. We want to assess how jamming can be used to disturb device communication, apart from purely preventing messages from being sent. That is, can jamming be used during normal device operation to more permanently disturb the communication, do the devices recognize that they are being jammed, and does the user get a notification of such an event? For this experiment we use the doorbell, but due to the very large similarities between this device and the motion sensor during other experiments, we can reasonably assume that the results also hold for the motion sensor. The use of the doorbell for this experiment is much more practical, as it is, as opposed to the motion sensor, not triggered accidentally. As explained earlier, we do not have the resources to perform jamming. Therefore, to simulate temporary jamming, we take the doorbell physically far out of range of the gateway at a distance of ca. 300 meters. While being out of reach, we press the doorbell three times with an interval of approximately 5 seconds. Because the CC2531, which we use for capturing packets (see Section 3.4.1), is physically close to the gateway and therefore also out of reach of the doorbell, we do not know what packets are sent by the doorbell when it finds out that there is no other device within range. We keep the doorbell out of range for 15 minutes, which simulates 15 minutes of active jamming. During this period, no push notification is being sent to the phone and the ZigBee traffic of the gateway looks normal. In the app, the status of the doorbell has not changed. It shows that the doorbell is fully functioning. The gateway thus does not know that the doorbell is being 'jammed'.
When bringing the doorbell back into the range of the gateway, there is no reaction from either of the devices in form of ZigBee packets or otherwise. However, by pressing the doorbell once, we see that it immediately sends out a Beacon Request, to which it gets a response from the gateway, and a Rejoin Request, which gets accepted. Only after this communication, the expected IAS Zone Status Update is sent, which results in a push notification. The Beacon Request and Rejoin Request mean that the doorbell has given up on sending the three notifications while it was being 'jammed' and decided to (temporarily) leave the network.
If the device is jammed but not triggered while being jammed, there will be no Rejoin Request as none of the involved devices will know that the traffic between them is being jammed due to the lack of heartbeat messages between the doorbell (or motion sensor) and the gateway. The Rejoin Request is thus the indicator that the communication has been jammed *and* the device was triggered during jamming. It is, however, not possible to tell how many times or when a device has been triggered during the jamming period, as it actively leaves the network when it notices that it doesn't get any replies.
When the doorbell is within range again (that is, not being 'jammed' anymore) and it is being pressed, it rejoins the network automatically and triggers a push notification. For the user, this looks like normal behavior. The only indicator that the doorbell has been jammed, is the rejoin request which can only be seen if low-level sniffing hardware is used such as the CC2531 that we use. In the app, there is no indication whatsoever that the doorbell has been temporarily out of the network. Therefore, there is for a user without dedicated hardware no possibility to know that a device has been jammed.

7 Conclusions and answers to research questions

We describe the conclusions that we draw from our results from Section 6 in Section 7.1, each followed by our recommendations for improvement. Using these conclusions we answer our research questions in Section 7.2.

7.1 Conclusions

Here we describe our conclusions based on the results from Section 6.

Conclusion 1: loss of privacy
We show that cloud service providers (Microsoft and Apple) can always see what devices are added to the network (Section 6.1.1, Attack A1) as it is not possible to do this offline (Section 6.3, Attack A7). Although, after setup, it is possible to partially use the functionalities of the LSH products, a lot of the functionality gets lost. Therefore most users will keep the network connected to the internet, which means that the cloud providers know about everything that is happening within the network. Therefore, these service providers are for example able to determine usage patterns of the user or determine when she is on vacation. All this is a loss of privacy for the user.

We recommend extending the usability of the LSH network to enable setting it up offline. This can be achieved by letting the phone and gateway communicate directly with each other within the home's Local Area Network.

Conclusion 2: total loss of control of all devices in case of key leakage
We see in Section 6.1.2, Attack A2 that the described "brief moment of vulnerability" [11] exists in the process of adding a LSH product to the network. By exploiting it, an attacker can retrieve the key that is used to encrypt virtually all commands, which is a breach of confidentiality. Although keys for end-to-end encryption are generated and distributed throughout the network, we see that only one single key is used for all devices (Section 6.1.2, Attack A2). This has as a result that, if an attacker exploits this vulnerability at any device, she can control all devices in the network to do anything she wants by injecting forged packets (Section 6.2, Attack A5) or make them unusable (Section 6.3, Attack A10) until the user resets them. These are severe breaches of integrity and availability respectively.
This "brief moment of vulnerability" [11] may seem irrelevant at first, because an attacker may not know when (or if) new devices get added to the network. However, this can be enforced: the doorbell and motion sensor are by the nature of their functionality likely to be used outside of the house, as people would like visitors to press the doorbell or secure a certain area by placing the motion sensor in such a way that the user gets a notification when any person enters that area. This means that an attacker can easily get physical access to a device, open it and reset it (Section 6.3, Attack A9). The user will get a push notification that the device has been tampered with, but it is fair to assume that most users would not hesitate to add the device to the network again to gain back the desired functionality. As we describe in Section 6.3, Attack A9 adding a device after it has been disconnected from the network results in the same communication, and therefore the same weaknesses, as adding it the first time. Therefore the compromise of one device leads to a compromise of all devices within the network.

To ensure end-to-end confidentiality of key material, we recommend that Install Codes (see Section 5.3), as defined in the ZigBee 3.0 specification [11] shall be used for adding a device to the network. Then it will not be possible to learn any keys by just listening, which means that an attacker will also not be able to control devices (see Section 6.2, Attack A4) or make them unusable without physical access.

Conclusion 3: the user is defenseless once the network gets compromised
During our research the same key is used to encrypt virtually all packets (Section 6.1.2, Attack A2).

Over the period of six months, we see no key rotation, meaning that this key is not changed once. As a result, if an attacker is able to retrieve the key used for encryption, she will also be able to attack the network months, potentially years later. If by any means, the user knows or just has the slightest feeling that an attacker has access to the devices because of a leaked key and wants to defend against it, there is no option for that. There is no way to tell the devices, in particular the gateway, that a new key should be used. This leaves the user defenseless against an attacker that knows the key used in the network.

To make a key leakage less severe over the long term, key rotation should be implemented, meaning that the key is changed regularly after a certain period (weeks/months). Preferably, the key rotation is end-to-end confidential. Also, the user should be able to proactively initiate it.

Conclusion 4: lack of forward secrecy
As we see in the previous conclusion, the key for encrypting packets is never changed and that only one key is used to encrypt all traffic. We also see that there is no forward secrecy mechanism, which means that in case of a leak of keying material, previously captured traffic can be decrypted later (Section 6.1.2, Attack A2). Therefore if this one key gets leaked, all traffic is compromised, including all previously captured traffic. An attacker that captures traffic without knowing any encryption keys, can save this traffic, capture the key later and decrypt all traffic afterward.

To prevent an attacker to decrypt previously captured packets, forward secrecy could be implemented. However, because in the IoT domain devices are often very resource-constrained regarding computational power, we alternatively recommend using multiple keys for end-to-end encryption, instead of using one key for all traffic within the network. That way, if a key gets compromised, only the traffic for that specific pair of devices is compromised, all other traffic, however, is not.

Conclusion 5: topology can be reconstructed
Upon receiving a IEEE 802.15.4 Beacon Request, devices send back a Beacon. A Beacon contains the device depth within the tree structure, which is the hop-distance to the coordinator. A direct connection to the coordinator is depth `1`. The Beacon also tells whether the device is the coordinator or can act as a router. This way, anyone can reconstruct the topology of a network by injecting Beacon Requests (Section 6.1.2, Attack A3). This is not necessarily harmful, but users should be aware of this if they use the products for security-relevant purposes.

If the presence of certain devices should not be possible to determine, we recommend not using any products that use IEEE 802.15.4, which includes all ZigBee products.

7.2 Answers to research questions

Here we use the conclusions from Section 7.1 to give answers to our research subquestions (see Section 2.2). We use those answers to answer the main research question below.

1. **What can be learned about the network and its devices by an outside attacker?**
   ISPs and cloud service providers know about anything happening in the network, including the used devices and behavioral data (Section 6.1.1, Attack A1). The user can choose to only use the devices offline, but it drastically lowers the available functions of the devices (Section 6.3, Attack A7). Due to the drastic loss of functionality with offline usage, we expect by far most users to use the LSH network online. Therefore, their ISP and the cloud service providers will always know what devices are used when by any user. This enables very detailed behavior pattern analysis, which is a violation of privacy.

   An attacker within range (70m) of the network can determine the topology of the network at any moment (Section 6.1.2, Attack A3), which we do not consider a problem. An attacker can listen

to network traffic at any moment without prior knowledge to determine the network activity. A burglar can use this traffic activity (or the lack of it) for determining whether a user is at home. If the activity is suddenly significantly lower for a few days, it is likely that the user is not at home and that it is opportune for the burglar to break into the house.
An attacker listening while a device is being added can learn the keying material necessary to read any ZigBee traffic within the network (Section 6.1.2, Attack A2). By reading the traffic in plaintext, a burglar can not only learn the traffic activity, but also the specific devices used in the network and their current state. This allows for more precise planning of the burglary.

2. **To what extent can device behavior be controlled by an outside attacker?**
   An attacker that does not know any keys used in the network, will not be able to control the device behavior (Section 6.2, Attack A4). If, however, an attacker knows the keying material (as is shown possible), then she can arbitrarily control any device within the network (Section 6.2, Attack A5) by injecting forged commands. If the user uses lamps to lighten dark areas, possibly to enable video surveillance at night, a burglar can turn off those lights in order to not be recognizable on the video recordings. Another threat to the user is that the power socket can be turned off and on arbitrarily often. This can result in serious damage to the devices that get their power through that power socket, including total loss of the devices.

3. **To what extent can an outside attacker prevent the user from normal device operation?**
   An attacker without key knowledge and physical access can not prevent the user from operating her devices. However, if the attacker does know the keying material, she can completely deactivate any device (Section 6.3, Attack A10) until it is reset by the user by pressing the reset button on it (as shown in Section 6.3, Attack A9). It is very likely that the user will not notice that a device has been deactivated by an attacker, because after the malicious command has been sent, no other traffic from the concerned device will get accepted by the gateway, and thus it will be unable to communicate its status. If a burglar decides to deactivate the motion sensor with this attack, it will be easier to break into the house, because the user will not get any notification from the device. The user thinks she is protected by the devices, but in reality, this is a false sense of security.

Using these answers to the research subquestions, we can answer our main research question:

How secure are the Lidl Smart Home products from a network perspective?

We see in the answers to the subquestions that an attacker can learn the topology of a network and the traffic activity and is potentially able to read any traffic in plaintext (breach of confidentiality). After learning the network's encryption key, an attacker can arbitrarily control any device within the network (breach of integrity) or even deactivate it until the user manually resets it (breach of availability). If a house is secured with LSH products, an attacker can abuse this to get to know when the user is not at home, turn off lights to remain unrecognized, and deactivate the motion sensor to prevent it from notifying the user of a breach. We do therefore conclude that the Lidl Smart Home products are not secure and should not be used for security-critical purposes.

8 Recommendations

In the previous sections, we discuss our findings on the security of the LSH products which include many vulnerabilities that can be misused in real-world attack scenarios. In this section, we describe our recommendations on how to improve the security and privacy of the products. These recommendations are based on our findings.

Enable offline usage
In Section 6.3 we show that the LSH network can only be set up and configured if the gateway is connected to the internet. Most of the functionality of the LSH products is lost when the gateway is offline. In Section 6.1.1, we show that multiple service providers are involved when the network is used. Because offline usage offers limited functionality and users are likely to choose to use the network to be online, service providers know the exact setup of the network and the usage behavior of the user. This is a great loss of privacy that can partly be mitigated.
To enable a privacy-friendly alternative, we recommend Lidl to implement the possibility of setting up and configuring an LSH network within a Local Area Network that is not connected to the internet. Most of the functionality should not depend on the gateway being online. For example, switching on and off lamps in the app could be realized within the LAN. Functionality such as receiving push notifications in case of a triggered motion sensor, even if the user is far from home, will inevitably get lost in this case. However, users that only have some lamps, for example, do not necessarily need the functionality of switching lights on and off, no matter where they are. These users could choose the privacy-friendly local offline option.

Install Codes
In Section 6.1.2 we show that the use of the Global Trust Center Link Key does not add to the confidentiality of the transported keying material. As explained in Section 5.3, Install Codes are the mutual basis for a pre-shared key between the coordinator and an end device, which guarantees confidentiality for the key transport under the assumption that an attacker does not have access to the Install Code. We, therefore, recommend Lidl to adjust their devices such that they can only be added to a ZigBee network with the use of install codes. The (unique) install codes are preinstalled on the end devices during manufacturing. When adding a new device to the network, the user then has to enter this install code in the app, which then communicates the code to the gateway. This way, the mutual basis for the pre-shared key is established. Install codes could for example be written on a sticker that comes with the device.
We believe that the use of install codes should not be optional because then the vulnerability would remain. Whether the device pairing is done securely or not would then depend on the choice of the user. Devices should therefore not support the encryption of packets with the global trust center link key, but always use install code link keys for key transport instead.
We recommend the ZigBee Alliance to remove the use of the global trust center link key from the ZigBee specification [11] entirely. This will prevent vendors from deploying new devices with this weakness in the future.

Enable key rotation
During the experiment period of six months we do not observe any rotation of keys, that is, the NWK Key and all link keys stay the same. As a result, an attacker that knows the NWK Key will be able to return to the victim network later in time and will still be able to perform the same attacks that require knowledge of this key. Periodic key renewal can protect against a non-persistent attacker, that is, an attacker that does not listen to the network all the time. Earlier retrieved keying material would not be valid anymore when the attacker comes back and therefore the attacks that require key knowledge do not succeed anymore.
To kick out a persistent attacker, the user has to initiate a key rotation in which install codes are used. Under the assumption that the attacker does not know any install codes, this ensures secure transport

of the new keying material. This procedure would enable the user to actively defend against an attacker in case of earlier key leakage.

Use Link keys
In Sections 4 and 6, we see that the compromise of one device leads to the compromise of the whole network. This is since for nearly all (encrypted) communication the NWK Key is used. To counter this, end-to-end encryption should be the default with device communication.
Therefore, we recommend the ZigBee Alliance to specify that end-to-end encryption through the use of application link keys should be the default. As a result, leakage of the NWK Key would only have a limited impact and leakage of an application link key would only affect one pair of devices. All other devices would not be affected, as opposed to the current situation.
In the case of the LSH devices, the end devices only communicate directly with the coordinator, which would mean that the trust center link keys, that are already established, would suffice to perform end-to-end encryption. Therefore, no extra storage space for keys would be needed.
If more devices would directly communicate with each other, as is supported by the mesh structure in ZigBee, the extra key management would generate a little overhead. We think, however, that in favor of added security it is worth the trade-off.

9 Future work

In this section we propose research directions within the security analysis of the LSH products and/or the ZigBee protocol.

Install Codes
Because Install Codes are not used within the LSH network, our research does not focus on practical attacks against ZigBee networks that do feature those. Future work may assess products that use Install Codes for initial key transport in greater detail. In particular, we hope to see that future ZigBee devices will all use Install Codes, such that attackers can not learn the NWK Key as easily as we show it can be done.

Large-scale (mesh) networks
As described in Section 3.1, we focus our research on small setups because we expect users to use these products similarly. The ZigBee protocol also supports large-scale mesh networks, for example for industrial use. In Section 6.1.2, Attack scenario A3, we describe how IEEE 802.15.4 Beacon Requests can be used to create a topology of any ZigBee network. Future work may assess how the network's topology is created and maintained in large-scale mesh networks, including possible security challenges such as routing and hidden wormholes. These are known challenges in 6LoWPAN [9][10][18][21], which, just like ZigBee, uses the IEEE 802.15.4 standard for wireless communication. It is therefore plausible that these problems also exist in ZigBee networks.

Hybrid approach
In our research, we take a relatively practical approach to attacking the LSH ZigBee network. To test the implementation more exhaustively, future work may combine a model-driven approach with the practical approach we use. This could help to (dis)prove certain properties, as did Li et al. [27], and test the implementation for undocumented or unwanted deviations from the ZigBee standard.

Tuya
The smartphone app and the end devices are all made by Tuya. Tuya's devices are sold by many more stores and under a wide variety of brands. Future work may assess how similar the devices from different brands are. The research may assess whether the same vulnerabilities can be found, whether the same attacks can be applied, and what the effects of a combination of devices of 'different' brands within the same network are.

Multiple app users
We describe the app only very briefly. Future work may assess the app in greater detail. A topic may be whether multiple genuine users can control the network from their phone with the app at the same time. This could have a variety of implications for the security of the network.

Other domains
In Section 6.1.1 we slightly touch upon the security of the ethernet connection between the gateway and the third-party servers. In Section 3.1 we briefly mention a hardware attack [40] on the gateway. Both domains are not assessed any further in our research. Future research may investigate the products' security in the ethernet security, software security, hardware security domain. These domains could be investigated separately or in a mixture.

Crossing the domains
In our research we focus on ZigBee traffic as it is defined in the specification [11]. Future work may investigate to what extent it is possible to 'escape' the ZigBee domain and trick devices into executing arbitrary commands. Securing devices against such attacks will for example help to prevent attackers from using devices as miners for crypto-currencies or prevent attackers from building a 'bridge' to the Local

Area Network (ethernet) and thus circumvent Wi-Fi protection mechanisms such as WPA2/WPA3.

10 Ethical considerations

Here we describe our ethical considerations on the execution of the experiments and the publication of the results thereof.

Responsible experiments
To do our research we acquired all LSH objects in order to attack them. No devices of others were attacked during the experiments. The extra traffic to the servers that was caused by our devices is the normal amount of traffic when using the devices. The traffic was not modified nor amplified and therefore no harm was done to the ISP nor the cloud service providers.

Responsible disclosure of results
We describe many vulnerabilities that can be misused by real-world attackers. Publishing these results without any precautions would therefore be a large security risk to all users of the Lidl Smart Home products which can lead to damage to the image of Lidl and the ZigBee Alliance (nowadays Connectivity Standards Alliance).
In particular, our attacks are not just purely academic, but they can be used on real-world setups of real users. Therefore, a malicious attacker could for example use our attacks to determine what devices a victim user has or control them by turning them on or off. An attacker could also use a victim LSH network to spam the user with arbitrarily many push notifications on her phone, as we describe in Section 6.2, or deactivate devices, as described in Section 6.3. This together leads to a loss of privacy and a false sense of security from the user's perspective.
Therefore we plan on disclosing our work and the vulnerabilities herein responsibly by informing Lidl and the ZigBee Alliance before publication. We will contact them in the short term and try to establish a dialogue where we explain our findings and discuss our recommendations (see Section 8) on how to improve product security to provide the possibility to mitigate (some) vulnerabilities before we disclose this thesis. We will propose a reasonable period in which they can react to our findings, for example by updating their products. We aim to agree on a time that is sufficient to address the most important issues but also allows us to publish our results within a reasonable time.

References

[1] Frank da Kruz, "Kermit Protocol Manual", *Columbia University Center for Computing Activities*, June 1986.

[2] Loren Kohnfelder, Praerit Garg, "The threats to our products", *Microsoft*, 1 April 1999. [Online]. Available: https://www.microsoft.com/security/blog/2009/08/27/the-threats-to-our-products/. [Re-published: Aug. 27, 2009, Accessed: July 19, 2021].

[3] Martin Roesch, "Snort - lightweight intrusion detection for networks", *USENIX Association: Proceedings of LISA '99: the 13th System Administration Conference*, November 1999.

[4] Jakob Jonsson, "On the Security of CTR + CBC-MAC", *SAC 2002, LNCS 2595, pp. 76–93*, 2003.

[5] NIST, "Recommendation for Block Cipher Modes of Operation: The CCM Mode for Authentication and Confidentiality", *NIST Special Publication 800-38C*, May 2004.

[6] "IEEE Standard for Local and metropolitan area networks - Part 15.4: Low-Rate Wireless Personal Area Networks (LR-WPANs)" in IEEE Std 802.15.4-2011 (Revision of IEEE Std 802.15.4-2006) , vol., no., pp.1-314, 5 Sept. 2011, DOI: https://doi.org/10.1109/IEEESTD.2011.6012487.

[7] Eunsook Kim, Dominik Kaspar, JP. Vasseur, "Design and Application Spaces for IPv6 over Low-Power Wireless Personal Area Networks (6LoWPANs)", *RFC 6568*, April 2012.

[8] Kyung Chio, Minjung Yun, Kijoon Chae, Mihui Kim, "An Enhanced Key Management Using ZigBee Pro for Wireless Sensor Networks", *The International Conference on Information Network 2012*, 2012, pp. 399-403, DOI: https://doi.org/10.1109/ICOIN.2012.6164409.

[9] Tzeta Tsao, Roger Alexander, Mischa Dohler, Vanesa Daza, Angel Lozano, Michael Richardson, "A Security Threat Analysis for Routing Protocol for Low-power and lossy networks (RPL)", *RFC 7416*, August 2014.

[10] Konrad-Felix Krentz, Gerhard Wunder, "6LoWPAN Security: Avoiding Hidden Wormholes using Channel Reciprocity", *ACM 4th International Workshop on Trustworthy Embedded Devices (TrustED '14)*. Association for Computing Machinery, New York, NY, USA, 13–22. DOI: https://doi.org/10.1145/2666141.2666143.

[11] ZigBee Alliance. "ZigBee Specification, ZigBee Document 05-3474-21", *ZigBee Alliance*, August 2015.

[12] Vijay Sivaraman, Hassan Habibi Gharakheili, Arun Vishwanath, Roksana Boreli, Olivier Mehani, "Network-Level Security and Privacy Control for Smart-Home IoT Devices", *IEEE 11th International Conference on Wireless and Mobile Computing, Networking and Communications (WiMob)*, 2015, pp. 163-167, DOI: https://doi.org/10.1109/WiMOB.2015.7347956.

[13] Louis Columbus, "Roundup Of Internet Of Things Forecasts And Market Estimates, 2016", *Forbes.com*, November 2016. [online]. Available: https://www.forbes.com/sites/louiscolumbus/2016/11/27/roundup-of-internet-of-things-forecasts-and-market-estimates-2016/?sh=7df0eda6292d. [Accessed: Dec. 30, 2021].

[14] Tobias Zillner. "Zigbee exploited: The good, the bad and the ugly", *Magdeburger Journal zur Sicherheits-forschung*, 12, 699–704, 2016.

[15] Jinesh Ahamed, Amala V. Rajan, "Internet of Things (IoT): Application Systems and Security Vulnerabilities", *IEEE 2016 5th International Conference on Electronic Devices, Systems and Applications (ICEDSA)*, 2016, pp. 1-5, DOI: https://doi.org/10.1109/ICEDSA.2016.7818534.

[16] Chase E. Stewart, Anne Maria Vasu, Eric Keller, "CommunityGuard: A Crowdsourced Home Cyber-Security System", *ACM International Workshop on Security in Software Defined Networks & Network Function Virtualization (SDN-NFVSec '17)*. Association for Computing Machinery, New York, NY, USA, 1–6. DOI: https://doi.org/10.1145/3040992.3040997.

[17] Flevina Jonese D'souza, Dakshata Panchal, "Advanced Encryption Standard (AES) Security Enhancement using Hybrid Approach", *IEEE 2017 International Conference on Computing, Communication and Automation (ICCCA)*, 2017, pp. 647-652, DOI: https://doi.org/10.1109/CCAA.2017.8229881.

[18] Ghada Glissa, Aref Meddeb, "6LoWPAN multi-layered security protocol based on IEEE 802.15.4 security features", *IEEE 2017 13th International Wireless Communications and Mobile Computing Conference (IWCMC)*, 2017, pp. 264-269, DOI: https://doi.org/10.1109/IWCMC.2017.7986297.

[19] Javid Habibi, Daniele Midi, Anand Mudgerikar, Elisa Bertino, "Heimdall: Mitigating the Internet of Insecure Things" *IEEE Internet of Things Journal*, vol. 4, no. 4, pp. 968-978, Aug. 2017, DOI: https://doi.org/10.1109/JIOT.2017.2704093.

[20] Livinus Obiora Nweke, "Using the CIA and AAA Models to Explain Cybersecurity Activities", *PM World Journal*, Vol. VI, Issue XII, December 2017. [Online]. Available: https://pmworldlibrary.net/wp-content/uploads/2017/05/171126-Nweke-Using-CIA-and-AAA-Models-to-explain-Cybersecurity.pdf. [Accessed: Dec. 30, 2021].

[21] Ghada Glissa, Aref Meddeb, "6LowPSec: an end-to-end security protocol for 6LoWPAN", *Elsevier Ad Hoc Networks*, Volume 82, 2019, Pages 100-112, ISSN 1570-8705, https://doi.org/10.1016/j.adhoc.2018.01.013.

[22] Ronny Ko, James Mickens, "DeadBolt: Securing IoT Deployments", *ACM Applied Networking Research Workshop (ANRW '18)*. Association for Computing Machinery, New York, NY, USA, 50–57. DOI: https://doi.org/10.1145/3232755.3232774.

[23] Christian Dietz, Raphael Labaca Castro, Jessica Steinberger, Cezary Wilczak, Marcel Antzek, Anna Sperotto, Aiko Pras, "IoT-Botnet Detection and Isolation by Access Routers" *IEEE 2018 9th International Conference on the Network of the Future (NOF)*, 2018, pp. 88-95, DOI: https://doi.org/10.1109/NOF.2018.8598138.

[24] Samuel Marchal, Markus Miettinen, Thien Duc Nguyen, Ahmad-Reza Sadeghi, N. Asokan, "AUDI: Toward Autonomous IoT Device-Type Identification Using Periodic Communication", *IEEE Journal on Selected Areas in Communications*, vol. 37, no. 6, pp. 1402-1412, June 2019, doi: https://doi.org/10.1109/JSAC.2019.2904364.

[25] Jingjing Ren, Daniel J. Dubois, David Choffnes, Anna Maria Mandalari, Roman Kolcun, Hamed Haddadi. "Information Exposure From Consumer IoT Devices: A Multidimensional, Network-Informed Measurement Approach", *ACM Internet Measurement Conference (IMC '19)*. Association for Computing Machinery, New York, NY, USA, 267–279. DOI: https://doi.org/10.1145/3355369.3355577.

[26] Francis Brown, Matt Gleason, "ARSENAL LAB - ZigBee Hacking: Smarter Home Invasion with ZigDiggity", Black Hat USA 2019, Aug. 7-8, 2019. [Workshop]. Slides available: https://www.slideshare.net/bishopfox/smarter-home-invasion-with-zigdiggity-165606623. [Accessed: Dec. 30, 2021].

[27] Li Li, Proyash Podder, Endadul Hoque. "A Formal Security Analysis of ZigBee (1.0 and 3.0)", *ACM 7th Symposium on Hot Topics in the Science of Security (HotSoS '20)*. Association for Computing

Machinery, New York, NY, USA, Article 12, 1–11. DOI: https://doi.org/10.1145/3384217.3385617.

[28] Dimitrios-Georgios Akestoridis, Madhumitha Harishankar, Michael Weber, Patrick Tague. "Zigator: Analyzing the Security of Zigbee-Enabled Smart Homes", *13th ACM Conference on Security and Privacy in Wireless and Mobile Networks (WiSec '20)*. Association for Computing Machinery, New York, NY, USA, 77–88. DOI: https://doi.org/10.1145/3395351.3399363.

[29] ZigBee Alliance, "ZigBee Technical: The Standard for the IoT", *ZigBee Alliance*, 2019. [Online]. Available: https://zigbeealliance.org/wp-content/uploads/2019/12/Zigbee-Technical-2019.pptx. [Accessed: Dec. 30, 2021].

[30] Salam Khanji, Farkhund Iqbal, Patrick Hung. "ZigBee Security Vulnerabilities: Exploration and Evaluating", *IEEE 2019 10th International Conference on Information and Communication Systems (ICICS)*, 2019, pp. 52-57, DOI: https://doi.org/10.1109/IACS.2019.8809115.

[31] Fadi Farha, Huansheng Ning, "Enhanced Timestamp Scheme for Mitigating Replay Attacks in Secure ZigBee Networks", *2019 IEEE International Conference on Smart Internet of Things (SmartIoT)*, 2019, pp. 469-473, DOI: https://doi.org/10.1109/SmartIoT.2019.00085.

[32] Weicheng Wang, Fabrizio Cicala, Syed Rafiul Hussain, Elisa Bertino, Ninghui Li. "Analyzing the Attack Landscape of Zigbee-enabled IoT Systems and Reinstating Users' Privacy", *13th ACM Conference on Security and Privacy in Wireless and Mobile Networks (WiSec '20)*. Association for Computing Machinery, New York, NY, USA, 133–143. DOI: https://doi.org/10.1145/3395351.3399349.

[33] Texas Instruments, "PACKET-SNIFFER: SmartRF Protocol Packet Sniffer" *Texas Instruments*. [Online]. Available: https://www.ti.com/tool/PACKET-SNIFFER. [Accessed: Dec. 30, 2021].

[34] Texas Instruments, "CC2531-RF4CE: Zigbee, IEEE 802.15.4 and RF4CE wireless MCU with up to 256kB Flash and 8kB RAM", *Texas Instruments*. [Online]. Available: https://www.ti.com/product/CC2531-RF4CE. [Accessed: Dec. 30, 2021].

[35] homewsn, "whsniff", *Github.com*, March 2020. [Online]. Available: https://github.com/homewsn/whsniff. [Accessed: Dec. 30, 2021].

[36] Scapy, "Dot15d4", *GitHub.com*, October 2020. [Online]. Available: https://github.com/secdev/scapy/blob/master/scapy/layers/dot15d4.py. [Accessed: Dec. 30, 2021].

[37] Dimitrios-Georgios Akestoridis, "zigator", *GitHub.com*, August 2020. [Online]. Available: https://github.com/akestoridis/zigator. [Accessed: Dec. 30, 2021].

[38] Matt Gleason, Francis Brown, "ZigDiggity", *Gitub.com*, August 2019. [Online]. Available: https://github.com/BishopFox/zigdiggity. [Accessed: Dec. 30, 2021].

[39] Dimitrios-Georgios Akestoridis, "wireshark-zigbee-profile", *Github.com*, February 2020. [Online]. Available: https://github.com/akestoridis/wireshark-zigbee-profile. [Accessed: Dec. 30, 2021].

[40] Paul Banks, "Hacking the Silvercrest (Lidl) Smart Home Gateway", *paulbanks.org*, February 2021. [Online]. Available: https://paulbanks.org/projects/lidl-zigbee/. [Accessed: Dec. 30, 2021].

[41] Dimitrios-Georgios Akestoridis, "Add ZDP and ZCL enhancements", *GitHub.com*, August 2021. [Online]. Available: https://github.com/secdev/scapy/commit/6ad83c513648fc1b4199a4b2d7b74b8a8c2ae0ce. [Accessed: Dec. 30, 2021].

[42] David Basin, Cas Cremers, Jannik Dreier, Simon Meier, Ralf Sasse, Benedikt Schmidt, "Tamarin Prover", *GitHub.io*. [Online]. Available: https://tamarin-prover.github.io/. [Accessed: Dec. 30, 2021].

[43] RF Wireless World, "Zigbee Frequency bands and data rates", *rfwireless-world.com*. [Online]. Available: https://www.rfwireless-world.com/Tutorials/Zigbee-frequency-bands-data-rates.html. [Accessed: Dec. 30, 2021].

[44] TÜV Rheinland, "Zigbee Testing and Certification". [Online]. Available: https://www.tuv.com/netherlands/en/zigbee%C2%AE-certification.html. [Accessed: Dec. 30, 2021].

[45] Multiscope, "1,2 miljoen huishoudens met slim licht", *Multiscope.nl* 26 August 2019. [Online]. Available: http://www.multiscope.nl/kennis/weblog/meer-dan-miljoen-huishoudens-met-slim-licht.html. [Accessed: Dec. 30, 2021].

[46] Olaf van Miltenburg, "Lidl legt Zigbee-smarthomeproducten in schappen en breidt aanbod begin 2021 uit", *tweakers.net*, November 2020. [Online]. Available: https://tweakers.net/nieuws/174824/lidl-legt-zigbee-smarthomeproducten-in-schappen-en-breidt-aanbod-begin-2021-uit.html. [Accessed: Dec. 30, 2021].

[47] Tom Dijkema, "Action, Hema, Kruidvat en Lidl hebben allemaal een eigen smarthome-collectie. Welke moet je nu kiezen? Wij vertellen het je in deze round-up." *FWD.nl*, December 2020. [Online]. Available: https://fwd.nl/smarthome/round-up-smarthome-van-action-hema-kruidvat-en-lidl/. [Accessed: Dec. 30, 2021].

[48] Lidl, "Gateway SGWZ 1 A1 – Operation and safety notes", *windows.net*, December 2020. [Online]. Available: https://stesbintegrationprod.blob.core.windows.net/public/articlemanual/91fa40c8-5e4e-4489-b03b-bcfb44d6c564.pdf. [Accessed: Dec. 30, 2021].

[49] World Standards, "Plug & socket types around the world", *worldstandards.eu*, September 2021. [Online]. Available: https://www.worldstandards.eu/electricity/plugs-and-sockets/. [Accessed: Dec. 30, 2021].

[50] Lennart Jansen, "Neckarspwn", *GitHub.com*, December 2021. [Online]. Available: https://github.com/ljonja/Neckarspwn. [Accessed: Dec. 30, 2021].

[51] Apple, "TCP and UDP ports used by Apple software products", *support.apple.com*. [Online]. Available: https://support.apple.com/en-us/HT202944. [Accessed: Dec. 30, 2021].

[52] Koen Kanters, "ZigBee 2 MQTT", *ZigBee2MQTT.io*. [Online]. Available: https://www.zigbee2mqtt.io/. [Accessed: Dec. 30, 2021].

[53] Tuya, "LIDL – Building a new IoT retail channel with lower prices, IoT-enabled products, and enhanced experience." *Tuya.com*. [Online]. Available: https://www.tuya.com/developer-stories/lidl. [Accessed: Dec. 30, 2021].

[54] Zigbee Alliance. [Online]. Available: https://zigbeealliance.org/. [Accessed: Dec. 30, 2021].

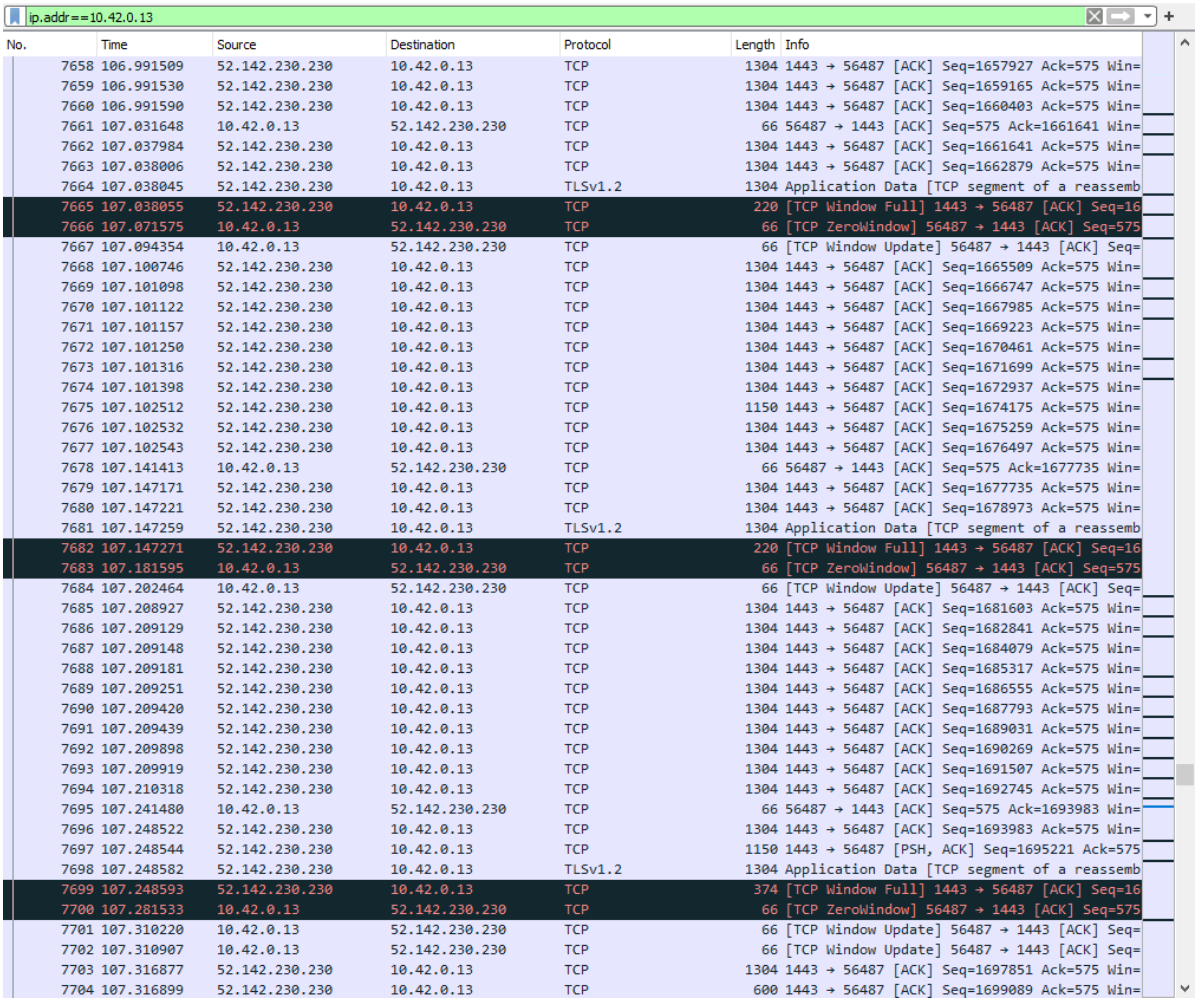
[55] Wireshark. [Online]. Available: https://www.wireshark.org/. [Accessed: Dec. 30, 2021].

[56] Scapy. [Online]. Available: https://scapy.net/. [Accessed: Dec. 30, 2021].

Appendix A Ethernet captures

A.1 TLS traffic

A screenshot of captured TLSv1.2 encrypted packets, as explained in Section 6.1.1. It is a software update for the gateway, which is automatically downloaded and installed during initial setup.



ip.addr==10.42.0.13

No.	Time	Source	Destination	Protocol	Length	Info
7658	106.991509	52.142.230.230	10.42.0.13	TCP	1304	1443 → 56487 [ACK] Seq=1657927 Ack=575 Win=
7659	106.991530	52.142.230.230	10.42.0.13	TCP	1304	1443 → 56487 [ACK] Seq=1659165 Ack=575 Win=
7660	106.991590	52.142.230.230	10.42.0.13	TCP	1304	1443 → 56487 [ACK] Seq=1660403 Ack=575 Win=
7661	107.031648	10.42.0.13	52.142.230.230	TCP	66	56487 → 1443 [ACK] Seq=575 Ack=1661641 Win=
7662	107.037984	52.142.230.230	10.42.0.13	TCP	1304	1443 → 56487 [ACK] Seq=1661641 Ack=575 Win=
7663	107.038006	52.142.230.230	10.42.0.13	TCP	1304	1443 → 56487 [ACK] Seq=1662879 Ack=575 Win=
7664	107.038045	52.142.230.230	10.42.0.13	TLSv1.2	1304	Application Data [TCP segment of a reassemb
7665	107.038055	52.142.230.230	10.42.0.13	TCP	220	[TCP Window Full] 1443 → 56487 [ACK] Seq=16
7666	107.071575	10.42.0.13	52.142.230.230	TCP	66	[TCP ZeroWindow] 56487 → 1443 [ACK] Seq=575
7667	107.094354	10.42.0.13	52.142.230.230	TCP	66	[TCP Window Update] 56487 → 1443 [ACK] Seq=
7668	107.100746	52.142.230.230	10.42.0.13	TCP	1304	1443 → 56487 [ACK] Seq=1665509 Ack=575 Win=
7669	107.101098	52.142.230.230	10.42.0.13	TCP	1304	1443 → 56487 [ACK] Seq=1666747 Ack=575 Win=
7670	107.101122	52.142.230.230	10.42.0.13	TCP	1304	1443 → 56487 [ACK] Seq=1667985 Ack=575 Win=
7671	107.101157	52.142.230.230	10.42.0.13	TCP	1304	1443 → 56487 [ACK] Seq=1669223 Ack=575 Win=
7672	107.101250	52.142.230.230	10.42.0.13	TCP	1304	1443 → 56487 [ACK] Seq=1670461 Ack=575 Win=
7673	107.101316	52.142.230.230	10.42.0.13	TCP	1304	1443 → 56487 [ACK] Seq=1671699 Ack=575 Win=
7674	107.101398	52.142.230.230	10.42.0.13	TCP	1304	1443 → 56487 [ACK] Seq=1672937 Ack=575 Win=
7675	107.102512	52.142.230.230	10.42.0.13	TCP	1150	1443 → 56487 [ACK] Seq=1674175 Ack=575 Win=
7676	107.102532	52.142.230.230	10.42.0.13	TCP	1304	1443 → 56487 [ACK] Seq=1675259 Ack=575 Win=
7677	107.102543	52.142.230.230	10.42.0.13	TCP	1304	1443 → 56487 [ACK] Seq=1676497 Ack=575 Win=
7678	107.141413	10.42.0.13	52.142.230.230	TCP	66	56487 → 1443 [ACK] Seq=575 Ack=1677735 Win=
7679	107.147171	52.142.230.230	10.42.0.13	TCP	1304	1443 → 56487 [ACK] Seq=1677735 Ack=575 Win=
7680	107.147221	52.142.230.230	10.42.0.13	TCP	1304	1443 → 56487 [ACK] Seq=1678973 Ack=575 Win=
7681	107.147259	52.142.230.230	10.42.0.13	TLSv1.2	1304	Application Data [TCP segment of a reassemb
7682	107.147271	52.142.230.230	10.42.0.13	TCP	220	[TCP Window Full] 1443 → 56487 [ACK] Seq=16
7683	107.181595	10.42.0.13	52.142.230.230	TCP	66	[TCP ZeroWindow] 56487 → 1443 [ACK] Seq=575
7684	107.202464	10.42.0.13	52.142.230.230	TCP	66	[TCP Window Update] 56487 → 1443 [ACK] Seq=
7685	107.208927	52.142.230.230	10.42.0.13	TCP	1304	1443 → 56487 [ACK] Seq=1681603 Ack=575 Win=
7686	107.209129	52.142.230.230	10.42.0.13	TCP	1304	1443 → 56487 [ACK] Seq=1682841 Ack=575 Win=
7687	107.209148	52.142.230.230	10.42.0.13	TCP	1304	1443 → 56487 [ACK] Seq=1684079 Ack=575 Win=
7688	107.209181	52.142.230.230	10.42.0.13	TCP	1304	1443 → 56487 [ACK] Seq=1685317 Ack=575 Win=
7689	107.209251	52.142.230.230	10.42.0.13	TCP	1304	1443 → 56487 [ACK] Seq=1686555 Ack=575 Win=
7690	107.209420	52.142.230.230	10.42.0.13	TCP	1304	1443 → 56487 [ACK] Seq=1687793 Ack=575 Win=
7691	107.209439	52.142.230.230	10.42.0.13	TCP	1304	1443 → 56487 [ACK] Seq=1689031 Ack=575 Win=
7692	107.209898	52.142.230.230	10.42.0.13	TCP	1304	1443 → 56487 [ACK] Seq=1690269 Ack=575 Win=
7693	107.209919	52.142.230.230	10.42.0.13	TCP	1304	1443 → 56487 [ACK] Seq=1691507 Ack=575 Win=
7694	107.210318	52.142.230.230	10.42.0.13	TCP	1304	1443 → 56487 [ACK] Seq=1692745 Ack=575 Win=
7695	107.241480	10.42.0.13	52.142.230.230	TCP	66	56487 → 1443 [ACK] Seq=575 Ack=1693983 Win=
7696	107.248522	52.142.230.230	10.42.0.13	TCP	1304	1443 → 56487 [ACK] Seq=1693983 Ack=575 Win=
7697	107.248544	52.142.230.230	10.42.0.13	TCP	1150	1443 → 56487 [PSH, ACK] Seq=1695221 Ack=575
7698	107.248582	52.142.230.230	10.42.0.13	TLSv1.2	1304	Application Data [TCP segment of a reassemb
7699	107.248593	52.142.230.230	10.42.0.13	TCP	374	[TCP Window Full] 1443 → 56487 [ACK] Seq=16
7700	107.281533	10.42.0.13	52.142.230.230	TCP	66	[TCP ZeroWindow] 56487 → 1443 [ACK] Seq=575
7701	107.310220	10.42.0.13	52.142.230.230	TCP	66	[TCP Window Update] 56487 → 1443 [ACK] Seq=
7702	107.310907	10.42.0.13	52.142.230.230	TCP	66	[TCP Window Update] 56487 → 1443 [ACK] Seq=
7703	107.316877	52.142.230.230	10.42.0.13	TCP	1304	1443 → 56487 [ACK] Seq=1697851 Ack=575 Win=
7704	107.316899	52.142.230.230	10.42.0.13	TCP	600	1443 → 56487 [ACK] Seq=1699089 Ack=575 Win=

Figure 19: TLSv1.2 traffic from, but mostly to the gateway at `10.42.0.13`

A.2 UDP heartbeat

As we describe in Section 6.1.1, the gateway sends out heartbeat packets. Many consecutive packets have the same payload. The only difference between them is the Identification number in the UDP header and consequently the header checksum. All other header fields and the content are identical in those packet.

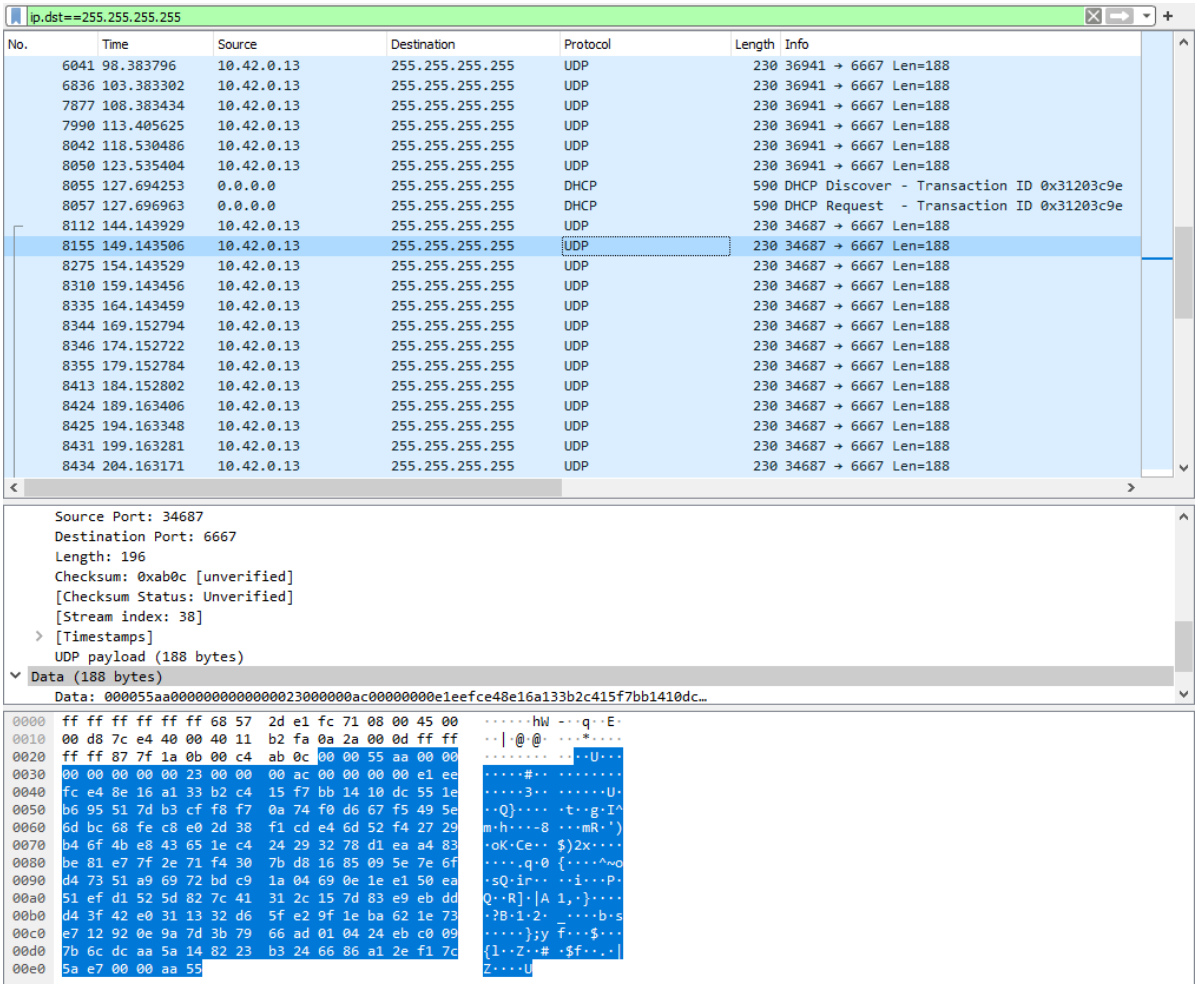


Figure 20: Screenshot of heartbeat packets in Wireshark

A.2.1 UDP heartbeat payloads

The exact data that is sent in every broadcast packet until packet no. 8050. From packet no. 8112 on, the content is changed to a new fixed value.

Payload 1:

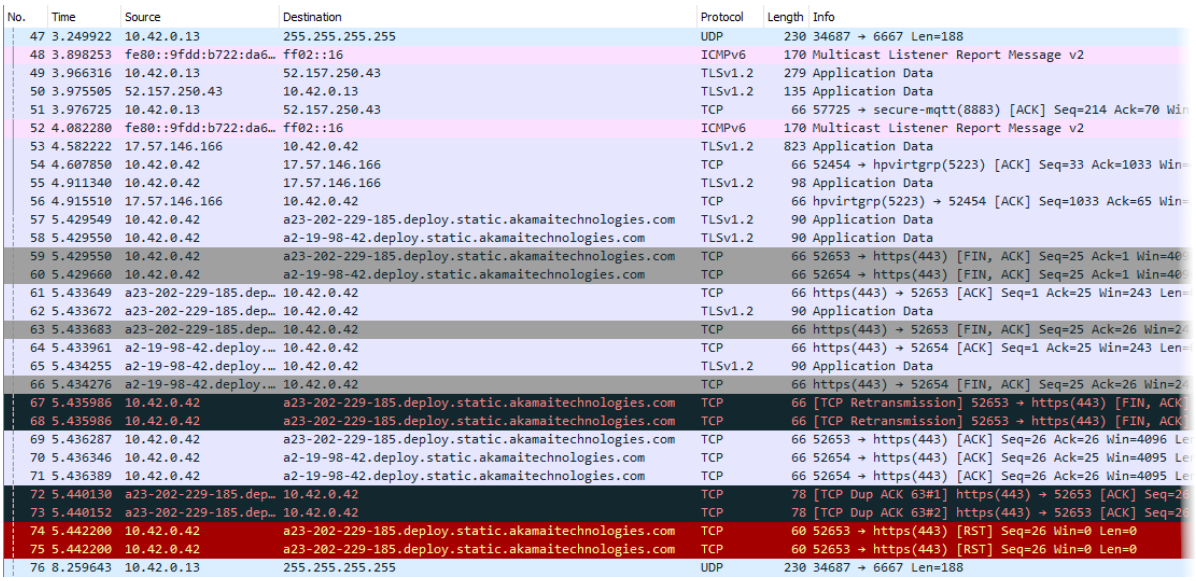
```
0000 00 00 55 aa 00 00 00 00 00 00 00 13 00 00 00 ac
0010 00 00 00 00 e1 ee fc e4 8e 16 a1 33 b2 c4 15 f7
0020 bb 14 10 dc a5 d8 91 d4 3e 51 86 96 65 b5 38 65
0030 7e 6c ef 1f b1 ed 5b d6 37 4c 0a 72 3b 87 d4 f9
0040 a1 86 1f 77 f2 02 f0 c8 03 cc 92 ea 0b a0 33 b9
0050 85 2f 70 7c d9 8a 20 83 cc 86 3e 53 24 9c df ed
0060 5e c9 31 22 b4 b2 bb ee ce 58 82 c4 5b 23 69 7e
0070 1d aa c6 fc 58 ae 2f e3 4d 45 df 46 52 0d 97 ce
0080 35 fe ee 70 c1 5f 4a be 41 10 07 52 06 5e 17 e4
0090 55 8a eb 2f 64 db ae fb cb a2 c8 7b 25 fb 8c 80
00a0 43 0f ec dd 2d d3 a7 95 42 4f 28 79 a4 49 2a 51
00b0 a7 d9 0f 07 7a 5c 21 48 00 00 aa 55
```

Payload 2:

```
0000 00 00 55 aa 00 00 00 00 00 00 00 13 00 00 00 ac
0010 00 00 00 00 e1 ee fc e4 8e 16 a1 33 b2 c4 15 f7
0020 bb 14 10 dc 55 1e b6 95 51 7d b3 cf f8 f7 0a 74
0030 f0 d6 67 f5 49 5e 6d bc 68 fe c8 e0 2d 38 f1 cd
0040 e4 6d 52 f4 27 29 b4 6f 4b e8 43 65 1e c4 24 29
0050 32 78 d1 ea a4 83 be 81 e7 7f 2e 71 f4 30 7b d8
0060 16 85 09 5e 7e 6f d4 73 51 a9 69 72 bd c9 1a 04
0070 69 0e 1e e1 50 ea 51 ef d1 52 5d 82 7c 41 31 2c
0080 15 7d 83 e9 eb dd d4 3f 42 e0 31 13 32 d6 5f e2
0090 9f 1e ba 62 ff 4e 78 fc 06 e2 37 98 64 ca 88 34
00a0 3f 17 f8 ba c0 09 7b 6c dc aa 5a 14 82 23 b3 24
00b0 66 86 a1 2e 77 9b 95 45 00 00 aa 55
```

A.3 Communication after pressing doorbell

These are all packets (without filter) on the local ethernet network that are sent/received after the doorbell is being pressed on time 3.9s. The gateway has IP address `10.42.0.13` and sends a TLS encrypted MQTT packet to a Microsoft server (`52.157.250.43`) in packet 49, receives a packet back (50) and sends an ACK as response (51). 0.6 seconds after that, the phone with IP address `10.42.0.42` gets a TLS encrypted packet from an Apple server (`17.57.146.166`) that results in the phone displaying the push notification that the doorbell has been pressed. After getting the notification from Apple, the phone contacts an Akamai server.



No.	Time	Source	Destination	Protocol	Length	Info
47	3.249922	10.42.0.13	255.255.255.255	UDP	230	34687 → 6667 Len=188
48	3.898253	fe80::9fdd:b722:da6…	ff02::16	ICMPv6	170	Multicast Listener Report Message v2
49	3.966316	10.42.0.13	52.157.250.43	TLSv1.2	279	Application Data
50	3.975505	52.157.250.43	10.42.0.13	TLSv1.2	135	Application Data
51	3.976725	10.42.0.13	52.157.250.43	TCP	66	57725 → secure-mqtt(8883) [ACK] Seq=214 Ack=70 Wi
52	4.082280	fe80::9fdd:b722:da6…	ff02::16	ICMPv6	170	Multicast Listener Report Message v2
53	4.582222	17.57.146.166	10.42.0.42	TLSv1.2	823	Application Data
54	4.607850	10.42.0.42	17.57.146.166	TCP	66	52454 → hpvirtgrp(5223) [ACK] Seq=33 Ack=1033 Win
55	4.911340	10.42.0.42	17.57.146.166	TLSv1.2	98	Application Data
56	4.915510	17.57.146.166	10.42.0.42	TCP	66	hpvirtgrp(5223) → 52454 [ACK] Seq=1033 Ack=65 Win
57	5.429549	10.42.0.42	a23-202-229-185.deploy.static.akamaitechnologies.com	TLSv1.2	90	Application Data
58	5.429550	10.42.0.42	a2-19-98-42.deploy.static.akamaitechnologies.com	TLSv1.2	90	Application Data
59	5.429550	10.42.0.42	a23-202-229-185.deploy.static.akamaitechnologies.com	TCP	66	52653 → https(443) [FIN, ACK] Seq=25 Ack=1 Win=40
60	5.429660	10.42.0.42	a2-19-98-42.deploy.static.akamaitechnologies.com	TCP	66	52654 → https(443) [FIN, ACK] Seq=25 Ack=1 Win=40
61	5.433649	a23-202-229-185.dep…	10.42.0.42	TCP	66	https(443) → 52653 [ACK] Seq=1 Ack=25 Win=243 Len=
62	5.433672	a23-202-229-185.dep…	10.42.0.42	TLSv1.2	90	Application Data
63	5.433683	a23-202-229-185.dep…	10.42.0.42	TCP	66	https(443) → 52653 [FIN, ACK] Seq=25 Ack=26 Win=2
64	5.433961	a2-19-98-42.deploy.…	10.42.0.42	TCP	66	https(443) → 52654 [ACK] Seq=1 Ack=25 Win=243 Len=
65	5.434255	a2-19-98-42.deploy.…	10.42.0.42	TLSv1.2	90	Application Data
66	5.434276	a2-19-98-42.deploy.…	10.42.0.42	TCP	66	https(443) → 52654 [FIN, ACK] Seq=25 Ack=26 Win=2
67	5.435986	10.42.0.42	a23-202-229-185.deploy.static.akamaitechnologies.com	TCP	66	[TCP Retransmission] 52653 → https(443) [FIN, ACK
68	5.435986	10.42.0.42	a23-202-229-185.deploy.static.akamaitechnologies.com	TCP	66	[TCP Retransmission] 52653 → https(443) [FIN, ACK
69	5.436287	10.42.0.42	a23-202-229-185.deploy.static.akamaitechnologies.com	TCP	66	52653 → https(443) [ACK] Seq=26 Ack=26 Win=4096 Le
70	5.436346	10.42.0.42	a2-19-98-42.deploy.static.akamaitechnologies.com	TCP	66	52654 → https(443) [ACK] Seq=26 Ack=25 Win=4095 Le
71	5.436389	10.42.0.42	a2-19-98-42.deploy.static.akamaitechnologies.com	TCP	66	52654 → https(443) [ACK] Seq=26 Ack=26 Win=4095 Le
72	5.440130	a23-202-229-185.dep…	10.42.0.42	TCP	78	[TCP Dup ACK 63#1] https(443) → 52653 [ACK] Seq=2
73	5.440152	a23-202-229-185.dep…	10.42.0.42	TCP	78	[TCP Dup ACK 63#2] https(443) → 52653 [ACK] Seq=2
74	5.442200	10.42.0.42	a23-202-229-185.deploy.static.akamaitechnologies.com	TCP	60	52653 → https(443) [RST] Seq=26 Win=0 Len=0
75	5.442200	10.42.0.42	a23-202-229-185.deploy.static.akamaitechnologies.com	TCP	60	52653 → https(443) [RST] Seq=26 Win=0 Len=0
76	8.259643	10.42.0.13	255.255.255.255	UDP	230	34687 → 6667 Len=188

Figure 21: Ethernet packets after the doorbell has been pressed (see Section 6.1.1)

Appendix B ZigBee captures

B.1 ZCL Sequence numbers

As explained in Section 5.2.4, sequence numbers can have multiple instances. Here we show different instances for the ZCL sequence number and the APS counter. We see that the packets 285, 331 and 355 use the same counter, just like packets 289, 333, 369 and 372 have their own. Packet 393 uses yet another counter for the ZCL sequence number and also has an unexpected value (227 instead of 240) for the APS counter.

wpan.src16==0x912b

No.	Time	MAC Src	NWK Src	MAC Dst	NWK Dst	MAC SN	NWK SN	Sec Ctr	APS Ctr	ZCL TSN	Length	Info
283	136.259397	0x912b	0x912b	0x0000	0x0000	110	30	12319	231		45	APS: Ack, Dst Endpt: 0, Src Endpt: 0
285	136.267327	0x912b	0x912b	0x0000	0x0000	111	32	12320	233	21	49	ZCL: Configure Reporting Response, Seq: 21
287	136.274931	0x912b	0x912b	0x0000	0x0000	112	33	12321	233		45	APS: Ack, Dst Endpt: 1, Src Endpt: 1
289	136.282511	0x912b	0x912b	0x0000	0x0000	113	35	12322	234	6	52	ZCL: Report Attributes, Seq: 6
291	137.116556	0x912b		0x0000		114					12	Data Request
295	137.123308	0x912b		0x0000		115					12	Data Request
299	137.134204	0x912b		0x0000		116					12	Data Request
303	137.146420	0x912b		0x0000		117					12	Data Request
307	138.089717	0x912b		0x0000		118					12	Data Request
311	138.115092	0x912b		0x0000		119					12	Data Request
315	138.126119	0x912b		0x0000		120					12	Data Request
319	138.133026	0x912b		0x0000		121					12	Data Request
323	138.155408	0x912b		0x0000		122					12	Data Request
327	138.173285	0x912b		0x0000		123					12	Data Request
331	138.229360	0x912b	0x912b	0x0000	0x0000	124	37	12323	235	22	60	ZCL: Read Attributes Response, Seq: 22
333	139.112050	0x912b	0x912b	0x0000	0x0000	125	39	12324	236	7	52	ZCL: Report Attributes, Seq: 7
335	139.136577	0x912b		0x0000		126					12	Data Request
339	139.184948	0x912b		0x0000		127					12	Data Request
343	139.196903	0x912b		0x0000		128					12	Data Request
347	139.215414	0x912b		0x0000		129					12	Data Request
351	139.225643	0x912b		0x0000		130					12	Data Request
355	139.252669	0x912b	0x912b	0x0000	0x0000	131	41	12325	237	23	58	ZCL: Read Attributes Response, Seq: 23
357	139.261282	0x912b	0x912b	0x0000	0x0000	132	42	12326	238		45	APS: Ack, Dst Endpt: 1, Src Endpt: 1
359	140.148674	0x912b		0x0000		133					12	Data Request
363	140.317931	0x912b		0x0000		134					12	Data Request
367	141.157364	0x912b		0x0000		135					12	Data Request
369	141.188714	0x912b	0x912b	0x0000	0x0000	136	44	12327	238	8	54	ZCL IAS Zone: Zone Status Change Notification, Seq: 8
372	142.140438	0x912b	0x912b	0x0000	0x0000	137	46	12328	239	9	52	ZCL: Report Attributes, Seq: 9
374	142.167814	0x912b		0x0000		138					12	Data Request
378	142.180155	0x912b		0x0000		139					12	Data Request
382	142.204045	0x912b		0x0000		140					12	Data Request
386	143.184674	0x912b		0x0000		141					12	Data Request
390	144.193233	0x912b		0x0000		142					12	Data Request
393	144.497470	0x912b	0x912b	0x0000	0x0000	143	47	12329	227	20	59	ZCL: Read Attributes Response, Seq: 20
397	145.204060	0x912b		0x0000		144					12	Data Request
403	145.492166	0x912b		0x0000		145					12	Data Request
408	146.214228	0x912b		0x0000		146					12	Data Request

Figure 22: ZigBee traffic with a variety of sequence number instances

B.2 Add device to network

This is a screenshot of Wireshark that captured the ZigBee communication between the gateway (`0x0000`) and the motion sensor (`0x912b`) whilst it is being added to the network. We see that during the process of adding the device to the network, the NWK Key is sent twice (packets 38 and 69).

No.	Time	MAC Src	NWK Src	MAC Dst	NWK Dst	MAC SN	NWK SN	Sec Ctr	APS Ctr	ZCL TSN	Length	Info
20	117.123643	0x0000				221					28	Beacon, Src: 0x0000, EPID: 91:cd:63:05:36:2a:a4:63
21	119.078219			0xffff		6					10	Beacon Request
22	119.144993	0x0000				222					28	Beacon, Src: 0x0000, EPID: 91:cd:63:05:36:2a:a4:63
23	121.010427	0x0000	0x0000	0xffff	0xfffc	231	32	42525	206		48	Permit Join Request
24	121.473198	0x0000	0x0000	0xffff	0xfffc	232	32	42526	206		48	Permit Join Request
25	121.699263			0xffff		24					10	Beacon Request
26	121.803489	0x0000				223					28	Beacon, Src: 0x0000, EPID: 91:cd:63:05:36:2a:a4:63
27	121.840873			0xffff		25					10	Beacon Request
28	121.921089	0x0000	0x0000	0xffff	0xfffc	233	32	42527	206		48	Permit Join Request
29	121.958318	0x0000				224					28	Beacon, Src: 0x0000, EPID: 91:cd:63:05:36:2a:a4:63
30	121.982207			0x0000		26					21	Association Request, RFD
31	121.983262					26					5	Ack
32	122.185108			0x0000		27					18	Data Request
33	122.186068					27					5	Ack
34	122.187779					234					27	Association Response, PAN: 0x1a3e Addr: 0x912b
35	122.189026					234					5	Ack
36	122.201130	0x912b		0x0000		28					12	Data Request
37	122.201898					28					5	Ack
38	122.203121	0x0000	0x0000	0x912b	0x912b	235	33	3	207		73	Transport Key
39	122.205841					235					5	Ack
40	122.402414	0x912b		0x0000		29					12	Data Request
41	122.403182					29					5	Ack
42	122.605425	0x912b		0x0000		30					12	Data Request
43	122.606193					30					5	Ack
44	122.806723	0x912b		0x0000		31					12	Data Request
45	122.807490					31					5	Ack
46	123.011606	0x912b		0x0000		32					12	Data Request
47	123.012374					32					5	Ack
48	123.212383	0x912b		0x0000		33					12	Data Request
49	123.213151					33					5	Ack
50	123.416396	0x912b		0x0000		34					12	Data Request
51	123.417163					34					5	Ack
52	123.619042	0x912b		0x0000		35					12	Data Request
53	123.619810					35					5	Ack
54	123.820479	0x912b		0x0000		36					12	Data Request
55	123.821247					36					5	Ack
56	127.074078	0x0000	0x0000	0xffff	0xfffc	236	34	42528			47	Link Status
57	127.302576			0xffff		38					10	Beacon Request
58	127.398331	0x0000				225					28	Beacon, Src: 0x0000, EPID: 91:cd:63:05:36:2a:a4:63
59	127.443636			0xffff		39					10	Beacon Request
60	127.492367	0x0000				226					28	Beacon, Src: 0x0000, EPID: 91:cd:63:05:36:2a:a4:63
61	127.585022			0x0000		40					21	Association Request, RFD
62	127.586077					40					5	Ack
63	127.787599			0x0000		41					18	Data Request
64	127.788559					41					5	Ack
65	127.790275					237					27	Association Response, PAN: 0x1a3e Addr: 0x912b
66	127.791523					237					5	Ack
67	127.802775	0x912b		0x0000		42					12	Data Request
68	127.803543					42					5	Ack
69	127.804387	0x0000	0x0000	0x912b	0x912b	238	35	4	208		73	Transport Key

Figure 23: The motion is being added to the network and it is being sent the NWK Key.

Appendix C Frame Control Fields

Here we describe the Frame Control Fields for the MAC Layer and the the NWK Layer.

C.1 MAC Layer Frame Control Field

All packets we capture during our research that contain a ZigBee NWK Layer, have a Frame Control Field with the value `0x8861` or `0x8841` on the MAC Layer, defined in IEEE 802.15.4 [6]. `0x8861` is used by the device originally sending the message, `0x8841` is used by routers forwarding a message and only differs in that it does not ask for an ACK response. The separate fields and flags of that FCF are shown in the table below.

Bits	Field	Value
.... .001	Frame Type	Data (0x1)
.... 0...	Security Enabled	False
....0	Frame Pending	False
....1.	Acknowledge Request	True
.... .1..	PAN ID Compression	True
.... 0...	Reserved	False
....0	Sequence Number Suppression	False
.... ..0.	Information Elements Present	False
.... 10..	Destination Addressing Mode	Short/16-bit (0x2)
..00	Frame Version	IEEE Std 802.15.4-2003
10..	Source Addressing Mode	Short/16-bit (0x2)

Table 3: MAC Layer Frame Control Field

C.2 ZigBee NWK Layer Frame Control Field

The most occurring FCF on the NWK Layer is `0x2248`. It is the carrier for ZigBee Data, which is mainly used for ZCL frames. The separate fields and flags of that FCF are shown in the table below.

Bits	Field	Value
....01	Frame Type	Data (0x0)
....00 10..	Protocol Version	2
.... 01..	Discover Router	Enabled
....0	Multicast	False
.... ..1.	Security	True
.... .0..	Source Route	False
.... 0...	Extended Destination	False
...0	Extended Source	False
..1.	End Device Initiator	True

Table 4: ZigBee NWK Layer Frame Control Field

Appendix D IEEE 802.15.4 / ZigBee Packets

In this section we show specific important packets. The colors are used to display the layers a value belongs to. The ∗ after a field value indicates that options are declared bit-wise instead of byte-wise.

Color scheme:

- IEEE 802.15.4
- ZigBee NWK Layer
- ZigBee APS Layer
- ZigBee Security Header as part of the NWK Layer or the APS Layer
- Zigbee Cluster Library Frame
- Zigbee Device Profile

D.1 Beacons

These are examples of a Beacon Request and a Beacon Response. When a Beacon Request is sent to the broadcast address (`0xffff`), all devices in the network respond with a Beacon Response. In Section 6.1.2 we explain how we learn more about the devices in the network and the network structure by sending a Beacon Request. In Section 6.3 we describe how a PAN ID conflict can be generated by sending a Beacon Response (without it being asked for with a Beacon Request).

Beacon Request

Description	Payload (hex, LE)
Frame Control Field	`0308*`
Sequence number	`4e`
Dest. PAN	`ffff`
Dest. addr.	`ffff`
Command Identifier *Beacon Request*	`07`
Frame Control Sequence	`4b12`

Table 5: Beacon Request

Beacon Response

Description	Payload (hex, LE)
Frame Control Field	`0080*`
Sequence number	`00`
Source PAN	`3e1a`
Source addr.	`0000`
Superframe specification	`ff4f*`
GTS	`00`
Pendig addr.	`00`
Protocol ID	`00`
Beacon Stack Profile	`2284*`
Extended PAN ID	`63a42a360563cd91`
Tx Offset	`ffffff`
Update ID	`00`
Frame Control Sequence	`4deb`

Table 6: Beacon Response

D.2 Turn on/off power sockets

This packet is used to control the power socket, as described in Section 6.2, Attack scenario A5. This packet is the command for turning **on** the socket number **1**. The alternative options are shown next to **Dest. Endpoint** and **Command** and separated with ||. By changing these values, updating the sequence numbers and recalculating the MIC and FCS, the other sockets can be turned *on* or *off*. What we find particularly interesting, is that the socket number (dest. endpoint) is defined in the APS Layer, whereas the command (on or off) is set in the Cluster Library Frame.

Description	**Payload (hex, LE)**
Frame Control Field	`6188*`
Sequence number	`6f`
Dest. PAN	`3e1a`
Dest. addr.	`a99f`
Source addr.	`0000`
Frame Control Field	`0802*`
Dest. addr.	`a99f`
Source addr.	`0000`
Radius	`1e`
Sequence number	`58`
Security Control Field	`28*`
Frame counter	`91c60000`
Extended source	`aacad1feff818e58`
Key Sequence number	`00`
Encrypted payload (11 bytes)	`2abf0654195c1d90498c71`
Decrypted:	
Frame Control Field	`00*`
Dest. Endpoint	`01 || 02 || 03`
Cluster *On/Off*	`0600`
Profile *Home Automation*	`0401`
Source Endpoint	`01`
Counter	`4c`
Frame Control Field	`11*`
Sequence number	`43`
Command *On* || *Off*	`01 || 00`
MIC	`81d5dbd9`
Frame Control Sequence	`d012`

Table 7: Packet sent by the coordinator to turn on or off a power socket

D.3 Remote control: press the “O”-button or “I”-button

This packet is sent by the remote control whenever the “O” or “I” buttons are pressed. The functionality of these buttons (and thus of this packet) is to turn on or off a set of lamps. We send this packet to control a lamp as described in Section 6.2.
To change the meaning of the command to *on* or *off*, the **Command** value has to be set to `0x00` (off) or `0x01` (on).
This packet is sent by the remote control (`0xabd8`). It sends the packet physically to the gateway (`0x0000`) only, but mentions in the NWK layer that it is to be broadcasted (`0xfffd`). The gateway and the power socket both act as router and thus forward the packet. These packets have both the MAC Layer (`0xffff`) and NWK Layer (`0xfffd`) address set to a broadcast address. This confirms that it is possible to extend the reach of a network (in terms of physical distance) by placing more network sockets at strategic locations.

Description	**Payload (hex, LE)**
Frame Control Field	`6188*`
Sequence number	`98`
Dest. PAN	`3e1a`
Dest. addr.	`0000`
Source addr.	`d8ab`
Frame Control Field	`4822*`
Dest. addr.	`fdff`
Source addr.	`d8ab`
Radius	`0c`
Sequence number	`f3`
Security Control Field	`28*`
Frame counter	`15000000`
Extended source	`046555feff142e84`
Key Sequence number	`00`
Encrypted payload (12 bytes)	`96f0ce1e2e99f3dcecdb8046`
Decrypted:	
Frame Control Field	`0c*`
Group	`0000`
Cluster	`0600`
Profile *Home Automation*	`0401`
Source endpoint	`01`
Counter	`bc`
Frame Control Field	`01*`
Sequence number	`06`
Command *Off* || *On*	`00 || 01`
MIC	`2f732c9f`
Frame Control Sequence	`9be6`

Table 8: Packet sent by the remote control after pressing ”O” or ”I”

D.4 Leave command

We observe that this packet is used to deauthenticate a device from the network, as described in Section 6.3, Attack scenario A9. This packet is also the basis for the forged unsecured version in Appendix D.9.

Description	Payload (hex, LE)
Frame Control Field	`6188*`
Sequence number	`93`
Dest. PAN	`3e1a`
Dest. addr.	`0000`
Source addr.	`ee69`
Frame Control Field	`0912*`
Dest. addr.	`fdff`
Source addr.	`ee69`
Radius	`01`
Sequence number	`e9`
Extended source	`4c49f9feff142e84`
Security Control Field	`28*`
Frame counter	`1e800000`
Extended source	`4c49f9feff142e84`
Key Sequence number	`00`
Encrypted payload (2 bytes)	`1707`
Decrypted:	
Command Identifier *Leave*	`04`
Flags: rejoin, request, remove children	`00*`
MIC	`468bd726`
Frame Control Sequence	`7eb2`

Table 9: Packet sent by the doorbell after the inside button is pressed three times

D.5 IAS Zone Status Update

An IAS Zone Status Update packet is sent by the doorbell and the motion sensor to inform the gateway that it has been pressed or motion was detected respectively. This process is explained in Section 3.4.1 and the results are described in Section 6.1.2. It is also used by both devices to inform the gateway that the device has been opened or closed.
In Section 6.2 we describe how we use this package to trigger arbitrary many push notifications being sent to the user.

Description	**Payload (hex, LE)**
Frame Control Field	`6188*`
Sequence number	`7b`
Dest. PAN	`3e1a`
Dest. addr.	`0000`
Source addr.	`ee69`
Frame Control Field	`4822*`
Dest. addr.	`0000`
Source addr.	`ee69`
Radius	`0c`
Sequence number	`e4`
Security Control Field	`28*`
Frame Counter	`1b800000`
Extended source	`4c49f9feff142e84`
Key Sequence number	`00`
Encrypted payload (17 bytes)	`8b1b083546e45428e486c27132d81fbc15`
Decrypted:	
Frame Control Field	`40*`
Dest. endpoint	`01`
Cluster *Intruder Alarm System Zone*	`0005`
Profile *Home Automation*	`0401`
Source endpoint	`01`
Counter	`7c`
Frame Control Field	`09*`
Sequence number	`18`
Command *Zone Status Change Notification*	`00`
ZoneStatus	`0500*`
Extended status	`00`
Zone ID	`00`
Delay in quarterseconds	`0000`
MIC	`35c33d51`
Frame Control Sequence	`95a3`

Table 10: Packet sent by the doorbell to inform about being pressed upon

*The **ZoneStatus** field is further explained on the next page.*

These are the exact bits of the ZoneStatus field:

Bits	Field	Value
`.... .... .... ...1`	Alarm 1	Opened or alarmed
`.... .... .... ..0.`	Alarm 2	Closed or not alarmed
`.... .... .... .1..`	Tamper	Tampered
`.... .... .... 0...`	Battery	OK
`.... .... ...0 ....`	Supervision reports	Does not report
`.... .... ..0. ....`	Restore reports	Does not report restore
`.... .... .0.. ....`	Trouble	OK
`.... .... 0... ....`	AC (mains)	AC/Mains OK

Table 11: ZoneStatus field bit specification

For the doorbell and motion sensor the different bit combinations are:

- Device opened:
  `Alarm 1: 0, Tamper: 1`
- Device closed:
  `Alarm 1: 0, Tamper: 0`
- Pressed/triggered while closed:
  `Alarm 1: 1, Tamper: 0`
- Pressed/triggered while open:
  `Alarm 1: 1, Tamper: 1`

For the doorbell all other bit values are always `0`. For the motion sensor all other values are also always `0`, except the `Restore reports` bit, which is always `1`.

D.6 Transport NWK Key

This is a transport key packet, which is used to transport the NWK Key, as explained in Section 6.1.2. It is sent by the coordinator (`0x0000`) to the newly added device, in this case the motion sensor (`0x912b`). We find it remarkable that the Key value is the only field that is sent in Big Endian. All other fields are sent in Little Endian.

Description	**Payload (hex, LE)**
Frame Control Field	`6188*`
Sequence number	`ee`
Dest. PAN	`3e1a`
Dest. addr.	`2b91`
Source addr.	`0000`
Frame Control Field	`0800*`
Dest. addr.	`2b91`
Source addr.	`0000`
Radius	`1e`
Sequence number	`23`
Frame Control Field	`21*`
Counter	`d0`
Security Control Field	`30*`
Frame Counter	`04000000`
Extended source	`aacad1feff818e58`
Encrypted Payload (35 bytes)	`2cda099dcddf89909a8f41423c7321f521c9`
	`ffc1c22872dd1b34afd071d8fd06f9dc67`
Decrypted:	
Command ID: Transport Key	`05`
Key Type: Standard NWK Key	`01`
Key value	`fed3ca371d15665ed99bb0ddaa5964d4` (BE)
Key Sequence number	`00`
Extended dest.	`f9f4e0feff142e84`
Extended source	`aacad1feff818e58`
Message Integrity Code	`9d805661`
Frame Control Sequence	`108b`

Table 12: Packet sent by the coordinator that holds the NWK Key

D.7 Request Key

This Key Request packet is used by newly added devices to request a Trust Center Link Key, as described in Section 6.1.2. It is sent by the newly added device, in this case the motion sensor (`0x912b`), to the coordinator (`0x0000`).
This packet and the Transport Trust Center Link Key (see Appendix D.8) are encrypted twice. This packet is Encrypted on the NWK Layer with the NWK Key and encrypted on the APS Layer with the Global Key 'ZigBeeAlliance09'.

Description	**Payload (hex, LE)**
Frame Control Field	`6188*`
Sequence number	`3f`
Dest. PAN	`3e1a`
Dest. addr.	`0000`
Source addr.	`2b91`
Frame Control Field	`0822*`
Dest. addr.	`0000`
Source addr.	`2b91`
Radius	`1e`
Sequence number	`fa`
Security Control Field	`28*`
Frame Counter	`0a300000`
Extended source	`f9f4e0feff142e84`
Key Sequence number	`00`
Encrypted payload (21 bytes)	`24ec2f0466a54e5dfd86ba1f48b30ab5a634f2ba02`
Decrypted:	
Frame Control Field	`21*`
Counter	`da`
Security Control Field	`20*`
Frame Counter	`00100000`
Extended source	`f9f4e0feff142e84`
Encrypted payload (2 bytes)	`aa7a`
Decrypted:	
Command Identifier *Request Key*	`08`
Key Type *Trust Center Link Key*	`04`
MIC	`06a4f0fe`
MIC	`a7efc7a1`
Frame Control Sequence	`21f4`

Table 13: Packet sent by the motion sensor to request a Trust Center Link Key

D.8 Transport Trust Center Link Key

This packet is used to transport the Trust Center Link key from the coordinator (`0x0000`) to the newly added device, in this case the motion sensor (`0x912b`), as described in Section 6.1.2. It is encrypted twice, just like the Request Key packet (see Appendix D.7): once at the NWK Layer and once at the APS Layer. The NWK Layer uses the NWK Key for encryption, the APS Layer uses the globally known Global Key. Effectively, the confidentiality of the Trust Center Link Key is thus only dependant on knowledge of the NWK Key.
Like the Key value in the Transport NWK Key packet (see Appendix D.6), the Key value is sent Big Endian instead of Little Endian.

Description	**Payload (hex, LE)**
Frame Control Field	`7188*`
Sequence number	`00`
Dest. PAN	`3e1a`
Dest. addr.	`2b91`
Source addr.	`0000`
Frame Control Field	`0802*`
Dest. addr.	`2b91`
Source addr.	`0000`
Radius	`1e`
Sequence number	`36`
Security Control Field	`28*`
Frame counter	`32a60000`
Extended source	`aacad1feff818e58`
Key Sequence number	`00`
Encrypted payload (53 bytes)	`8b54141bf197fa3144a201617e58737a7f00 0ff329c2c3b50d9d2c17187215cd8ef17fdc fe4c38cbda37b387ba4503f2a7183791ae`
Decrypted:	
Frame Control Field	`21*`
Counter	`d7`
Securit Control Field	`38`
Frame counter	`05000000`
Extended source	`aacad1feff818e58`
Encrypted payload (34 bytes)	`1f02ad7929d8cf3fec10bb3114008227791e 76908990350c24390f7b451cc03ceb32`
Decrypted:	
Command ID *Transport Key*	`05`
Key Type *Trust Center Link Key*	`04`
Key	`22714a372c76f6f278815ee5ae1474cb` (BE)
Extended dest.	`f9f4e0feff142e84`
Extended source	`aacad1feff818e58`
MIC	`5cb1090b`
MIC	`c2bf0a9f`
Frame Control Sequence	`1dba`

Table 14: Packet sent by the coordinator that holds the Trust Center Link Key

D.9 Unsecured leave

This is a crafted unsecured leave command. In Section 6.2 we explain how we try to deauthenticate a device from the network by sending this packet. This packet only has a MAC Layer and NWK Layer.

Description	**Payload (hex, LE)**
Frame Control Field	`4188*`
Sequence number	`25`
Dest. PAN	`3e1a`
Dest. addr.	`0000`
Source addr.	`ca9e`
Frame Control Field	`0910*`
Dest. addr.	`fdff`
Source addr.	`ca9e`
Radius	`1e`
Sequence number	`2a`
Exented source	`4c49f9feff142e84`
Command Identifier *Leave*	`04`
Flags: *rejoin, request, remove children*	`00*`
Frame Control Sequence	`3393`

Table 15: Forged 'Leave' packet, based on Appendix D.4

D.10 Unsecured rejoin

This is a crafted unsecured rejoin request. In Section 6.2 we explain how we try to enter a network in an unsecured manner. This packet only has a MAC Layer and NWK Layer.

Description	Payload (hex, LE)
Frame Control Field	`6188*`
Sequence number	`1d`
Dest. PAN	`3e1a`
Dest. Addr.	`0000`
Source addr.	`2b91`
Frame Control Field	`0910*`
Dest. addr.	`0000`
Source addr.	`2b91`
Radius	`1e`
Sequence number	`26`
Extended source	`f9f4e0feff142e84`
Command Identifier *Rejoin Request*	`06`
Capability Information	`80*`
Frame Control Sequence	`8030`

Table 16: Forged 'Rejoin' packet

Appendix E Screenshot of a manipulation warning

This is a screenshot of a push notification notifying the user that a device, in this case the doorbell ("Deurbel"), has been removed, which is the result of opening the device. This is explained in Sections 3.4.1, 6.1.1 and 6.1.2. See also Appendix D.5 for the ZigBee packet (IAS Status Zone update) that results in this push notification.

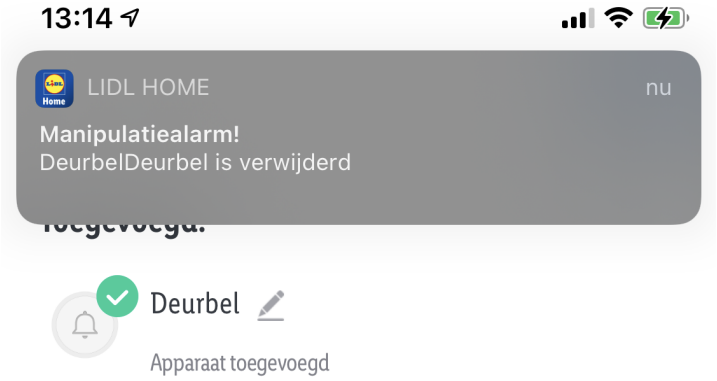


Figure 24: Push notification that the device "Deurbel" (doorbell) has been 'removed' (opened).

Appendix F Scapy extension: IAS Zone Status Update

As explained in Sections 3.4.3 and 6.1, the IAS Zone Status update packet was not implemented in Scapy [56] at the time of our experiments. Therefore we present our own implementation of it here. It is an extension to `path/to/Scapy/layers/zigbee.py` and should therefore be pasted into that file. It can be used inside Scapy to create a new instance directly by calling `ZCLIASZoneStatus(Packet: bytes)` or by calling a lower level class such as `Dot15d4(Packet: bytes)`, which then parses the byte string and automatically chooses the `ZCLIASZoneStatus` if appropriate.

```
# Based on captures of the Lidl Smart Home product line. No warranty!

# To be inserted between the `ZCLPricePublishPrice` class
# and `ZigbeeClusterLibrary` class.

_zcl_alarm = {
    0: "Closed or not alarmed",
    1: "Opened or alarmed"
}

class ZCLIASZoneStatus(Packet):
    name = "ZCL Intruder Alarm System Zone Status Frame"
    fields_desc = [
        # ZoneStatus
        BitEnumField("ac_mains", 0, 1, {0: "OK", 1: "Not OK"}),
        BitEnumField("trouble", 0, 1, {0: "OK", 1: "Not OK"}),
        BitField("restore_reports", 0, 1),
        BitField("supervision_reports", 0, 1),
        BitEnumField("battery", 0, 1, {0: "OK", 1: "Not OK"}),
        BitField("tamper", 0, 1),
        BitEnumField("alarm2", 0, 1, _zcl_alarm),
        BitEnumField("alarm1", 0, 1, _zcl_alarm),
        BitField("reserved", 0, 8),
        BitField("extended_status", 0, 8),
        BitField("zone_id", 0, 8),
        BitField("delay", 0, 16)
    ]

# To be inserted in the function 'guess_payload_class()' from the class
# 'ZigbeeClusterLibrary', just before the final return statement

        elif self.zcl_frametype == 0x01 and self.command_identifier == 0x00 \
                and self.command_direction == 1 and self.manufacturer_specific == 0:
            return ZCLIASZoneStatus

# To be inserted right after the other 'bind_layers()' function
# calls which connect the 'ZigbeeClusterLibrary' class with the
# 'ZCLGeneralReadAttributes' and 'ZCLGeneralReadAttributesResponse' classes.

bind_layers(ZigbeeClusterLibrary, ZCLIASZoneStatus,
            zcl_frametype=0x01, command_identifier=0x00)
```