Analyzing the Attack Landscape of Zigbee-enabled IoT Systems and Reinstating Users' Privacy

Weicheng Wang
wang3623@purdue.edu
Purdue University

Fabrizio Cicala
fcicala@purdue.edu
Purdue University

Syed Rafiul Hussain
hussain1@purdue.edu
Purdue University

Elisa Bertino
bertino@purdue.edu
Purdue University

Ninghui Li
ninghui@cs.purdue.edu
Purdue University

Abstract

Zigbee network security relies on symmetric cryptography based on a pre-shared secret. In the current Zigbee protocol, the *network coordinator* creates a *network key* while establishing a network. The coordinator then shares the network key securely, encrypted under the pre-shared secret, with devices joining the network to ensure the security of future communications among devices through the network key. The pre-shared secret, therefore, needs to be *installed* in millions or more devices prior to deployment, and thus will be inevitably leaked, enabling attackers to compromise the confidentiality and integrity of the network. To improve the security of Zigbee networks, we propose a new certificate-less Zigbee *joining* protocol that leverages low-cost public-key primitives. The new protocol has two components. The first is to integrate Elliptic Curve Diffie-Hellman key exchange into the existing association request/response messages, and to use this key both for link-to-link communication and for encryption of the network key to enhance privacy of user devices. The second is to improve the security of the installation code, a new joining method introduced in Zigbee 3.0 for enhanced security, by using public key encryption. We analyze the security of our proposed protocol using the formal verification methods provided by ProVerif, and evaluate the efficiency and effectiveness of our solution with a prototype built with open source software and hardware stack. The new protocol does not introduce extra messages and the overhead is as lows as 3.8% on average for the *join* procedure.

CCS Concepts

• **Computer systems organization** → **Embedded systems**; *Redundancy*; Robotics; • **Networks** → Network reliability.

Keywords

IoT, Zigbee, Attacks, Privacy, ECDH, Key Management, Formal Analysis

ACM Reference Format:
Weicheng Wang, Fabrizio Cicala, Syed Rafiul Hussain, Elisa Bertino, and Ninghui Li. 2020. Analyzing the Attack Landscape of Zigbee-enabled IoT Systems and Reinstating Users' Privacy. In *13th ACM Conference on Security and Privacy in Wireless and Mobile Networks (WiSec '20), July 8–10, 2020, Linz (Virtual Event), Austria.* ACM, New York, NY, USA, 11 pages. https://doi.org/10.1145/3395351.3399349

1 Introduction

In recent years, Internet-of-things (IoT) devices have become ubiquitous. According to SafeAtLast statistics [5], in 2019, there were about 26.6 billion active IoT devices, and the number is expected to reach 30 billion within 2020. The constant spreading of IoT technologies and the increasing demand led to the development of several IoT standards. Among these, Zigbee [3] is one of the earliest and most widespread standards. According to the official documentation, Zigbee [2] was developed as a "very low-cost, very low-power-consumption, two-way, wireless communications standard", designed to be embedded in building automation and industrial controls. The standard is implemented in a large number of different devices that range from home automation products to industry-related devices, such as smart lights, door locks, cameras, smart sensors and detectors (e.g., smoke and fire detectors). As this continuous growth is transforming our houses and work environments into highly connected networks of smart devices, the security and privacy aspects of the Zigbee protocol are becoming more important and urgent. Indeed, as the connectivity increases, so does the risk of vulnerability points that can be exploited by malicious users.

Problem and scope. Zigbee deploys cryptographic protocols to achieve security and privacy objectives. However, existing protocols are based on a faulty adversary model in which all benign devices share some secret master key, not accessible to the attacker. To guarantee interoperability among devices from different vendors and manufacturers, every device must be provided with the pre-shared secret. Any secret key that needs to be shared by a large number of devices will inevitably be reconstructed and become public knowledge. An attacker with the secret key can destroy the security of any Zigbee network it aims to attack. While a new security feature in Zigbee 3.0 called installation code helps alleviate this problem; installation code still uses symmetric cryptography. As a result, installation code is vulnerable to attackers who control devices that can read the installation code. In this paper, we propose enhancements to the current version of the Zigbee protocol to address these vulnerabilities.

Challenges. While Zigbee allows the creation of complex IoT networks with low-cost smart devices, the limited capabilities and resources, and the automation requirements present stringent constraints over the security protocols. In the following, we discuss two main challenges in designing a solid enhancement to the current Zigbee protocol.

(1) Limited capabilities. Low power consumption is a fundamental requirement for IoT devices, as it leads to extended durability and reduced energy usage, which allows customers to save money and avoid buying new devices frequently. To achieve low cost and low power consumption, Zigbee devices (e.g., light bulbs or electrical sockets) have limited storage and computational capabilities, and, in most cases, lack input interfaces. The lack of computational power poses constraints on the cryptographic primitives that suit the Zigbee environment, as it is necessary to choose efficient solutions. Additionally, given the restrained input capabilities, security and authentication models based on user-inserted passwords, such as the WiFi model, are not feasible, as they require an input interface.

(2) Efficiency. One of the main benefits of the IoT networks, such as Zigbee, is communication efficiency. IoT devices are designed to transmit small-size packets that only contain short and specific commands, and the number of messages exchanged between devices is kept low (IEEE 802.15.4 limits each message to 127 bytes). Therefore, our goal is to design a mechanism that improves the security of the current protocol without adding any additional procedure or message, while staying under the packet size limit.

Approach. To address the vulnerabilities of the Zigbee protocol while meeting the constraints imposed by the IoT environment, we propose to enhance the Zigbee protocol through two main changes. First, we propose to use the Elliptic Curve Diffie Hellman (ECDH) [23] algorithm in the key exchange procedure to establish a link key between each pair of devices that directly communicate with each other. This eliminates the need for a pre-shared secret among the devices, and allows each device-to-device communication link to be secured with a different encryption key established from two ephemeral secrets. In other words, no other device, besides those involved in the communication, has access to the ECDH key. This approach secures the Zigbee network against passive adversaries, but remains vulnerable to active attackers (e.g., man-in-the-middle). Second, we propose to improve the installation code mechanism introduced in Zigbee 3.0 by using a public key-based scheme, enabling secure authentication of specific devices joining a user's network, even against active adversaries.

Contribution. In summary, the paper has the following contribution:

(1) We analyze the attack landscape for the current Zigbee standard.
(2) We propose enhancements to the protocol to address the attacks. The main goal is to improve the security of the Zigbee standard, without drastically impacting the performances.
(3) We evaluate the proposed enhancements in two ways. First, we use ProVerif to verify the correctness of the proposed protocols. Second, we implement and deploy the new protocol to evaluate and compare it with the Zigbee standard implementations in terms of time delay, memory usage, and message size increment.

2 Background

Zigbee is a low-cost, low-power-consumption protocol that supports communication among devices via radio transmission. In this section, we will introduce the roles of devices in Zigbee network, the architecture of Zigbee and security related protocols in Zigbee.

2.1 Zigbee devices

A Zigbee network contains three types of devices with specific roles in the network as described below.

Coordinator. Each Zigbee network has a single coordinator, which establishes the network and allows other devices to join. The coordinator assigns a unique identifier called Personal Area Network ID (PAN ID) to the network and runs an application called the *Zigbee Trust center*, which manages the routing tables of the network. The coordinator has an out-of-band channel (e.g., Wiress LAN) to communicate with the cloud, the user, or both.

End device. End devices are the end points on the network. They provide functions, such as detecting the sounds, monitoring the temperature, locking or unlocking the doors, etc.

Router. Routers forward the messages between the end devices and the coordinator. In order to do that, they maintain routing tables and forward messages. End devices can also perform the role of routers, forwarding messages between other devices and the coordinator.

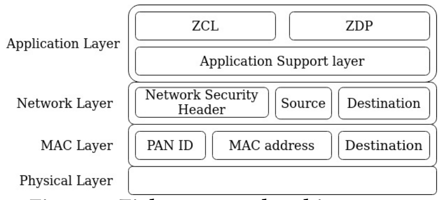

Figure 1: Zigbee protocol architecture.

2.2 Zigbee Network Architecture

Zigbee protocol stack is a layered stack. From the bottom to the top, the layers are Physical, Media Access Control (MAC), Network and Application as shown in Figure 1. Physical and MAC layers are defined by the IEEE 802.15.4 standard. Similar to other protocols, the aim of these two layers is to support packets transmission and the payloads/contents added by these two layers are unencrypted. The network layer is responsible for network layer packets forwarding, whereas the application (support) layer handles the concrete application-level commands, such as open/close door lock. The contents of these two layers are encrypted using AES-CCM* algorithm.

2.3 Zigbee Security Keys

There are several types of keys in the Zigbee network. The Network Key (NWK) is a 128-bit secret randomly generated by the coordinator when it forms the network. When a device joins the network, the device receives the NWK, stores it in a specific key table, and updates it as needed. Devices use the NWK to achieve message secrecy and authenticate the network layer payload through the AES-CCM* cryptographic algorithm.

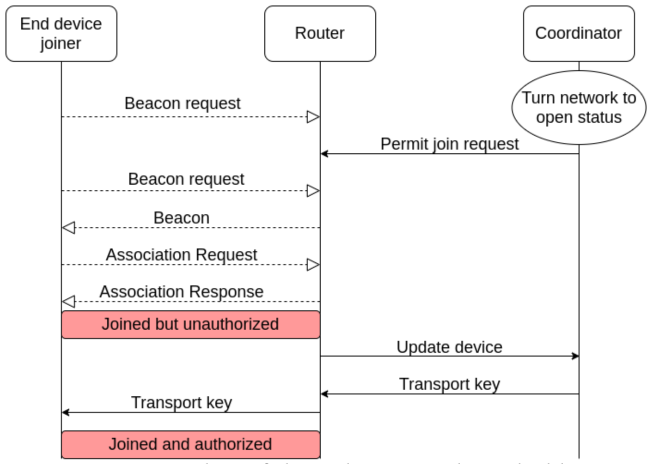

Figure 2: Join procedure of the Zigbee Network. Dashed lines represent unencrypted messages, whereas solid lines represent encrypted messages.

Each device uses one of the Trust Center Link Keys (TCLKs) to communicate with the coordinator during the initial joining procedure. The NWK is delivered to the device under the encryption of a TCLK. All Zigbee devices uses a Global Master Key (GMK), whose value is derived from the string "ZigbeeAlliance09", as one of its TCLKs. Since GMK is public knowledge, some vendors choose another key and use it as an additional TCLK for its devices. Some of these keys have also been leaked to the public.

The Zigbee protocol provides the possibility to secure the communication between a pair of devices individually using a key specific to the connection (called Link Key) instead of the NWK. If such a key is employed, other nodes cannot decrypt the content of the messages. In the current Zigbee stack profile, the usage of link keys is optional, as the NWK can be used both for network layer and for application layer encryption. Currently, most commercial Zigbee devices only use link key in NWK exchanging process and then use NWK to encrypt all the messages in default [32] [12].

2.4 Join Procedure

When a user wants to add a new end device to an existing Zigbee network, the user sends a command (through an out-of-band channel) to the coordinator, which turns the network's state from closed to open. In closed state, no new device is allowed to join the network, whereas, after the network is switched to open state, the coordinator broadcasts a permit_join_response message, which notifies all devices that the network is in open state and new joining requests are accepted.

When an end device wants to join a Zigbee network, it keeps broadcasting an unencrypted beacon_request message, until it receives a beacon message as response. When a router receives a beacon_request, if the network is in the open state, it broadcasts a beacon message with the `association permit` sub-field set to 1 (`permit association`). Router's MAC address and Personal Area Network (PAN) ID are also included in the beacon message.

After receiving a `permit association` from the router, the end device starts establishing a connection by sending an association_request message to the router, which responds with an association_response message. Both are unencrypted unicast messages. Then the router sends an update_device message to the coordinator for authorizing the new device to join the network. If the coordinator authorizes the joining, it transmits a transport_key message, which encrypts the NWK under the TCLK, via the router to the end device. After receiving and decrypting the message, the end device joins the network and can use the NWK in future communications.

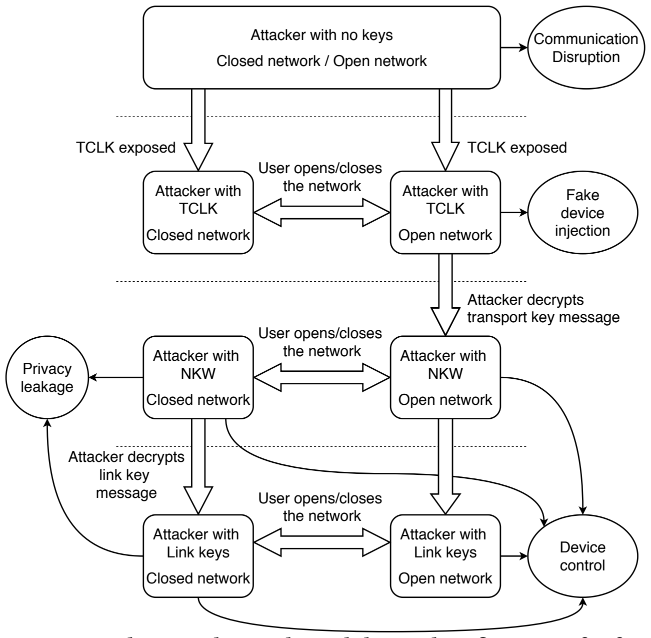
Attacker with no keys
Closed network / Open network
Communication Disruption
TCLK exposed
TCLK exposed
User opens/closes the network
Attacker with TCLK Closed network
Attacker with TCLK Open network
Fake device injection
Attacker decrypts transport key message
User opens/closes the network
Privacy leakage
Attacker with NKW Closed network
Attacker with NKW Open network
Attacker decrypts link key message
User opens/closes the network
Attacker with Link keys Closed network
Attacker with Link keys Open network
Device control

Figure 3: Zigbee attacks graph. Each layer identifies a set of information possessed by the adversary and the type of attack it can perform. An adversary with stronger information can perform attacks that require less information.

3 Characterizing Attack Landscape of Zigbee Network

As discussed in Section 2.3, messages between coordinators and end devices are encrypted under the NWK. As a result, an attacker that possesses the NWK can eavesdrop on every message transmitted within the network, as well as disrupt the functionalities of the network by sending forged messages. In the rest of this section, we define the adversary model and explore the attack landscape within such a model. We organize our discussions based on whether the attacker can obtain the NWK or not. In Figure 3 we provide a graphic representation of the overall attack landscape.

3.1 Adversary Model

The adversary model we consider is compliant with the Dolev-Yao model [10]. Namely, the adversary can read, modify, delete, and inject messages on the communication channel, and impersonate legitimate devices in the network. However, the adversary adheres to all cryptographic assumptions.

Moreover, we make two assumptions over the devices in the network: (1) the devices controlled by the user are benign (i.e., the devices are not compromised with malware or backdoor that can be exploited by an attacker); (2) the adversary can bring malicious devices into the Zigbee network area, and send and receive packets arbitrarily.

Given such assumptions, we consider two types of attackers: *passive* and *active*. A passive attacker can set up a sniffer in the target's Zigbee network that can collect all the Zigbee packets transmitted over the air. The attacker can then analyze the packets and retrieve information about the devices in the network. An active attacker can bring a malicious device that can send Zigbee packets. The malicious device can act as an end device to join the network, or act as a coordinator and set up another Zigbee network. It can also spoof existing devices by forging the public identity (like MAC address, PAN ID) to deceive the other devices in the network.

In this paper, we do not consider an adversary that exploits other security aspects of Zigbee systems, such as compromising a device's pre-shared secret or the installation code through physical or remote access during manufacturing or post-manufacturing time, and denial-of-service or resource exhaustion by jamming the device's communication with arbitrary physical or network layer messages.

3.2 Attackers with no NWK

We first consider attackers who are unable to obtain the NWK of a Zigbee network. This happens in two situations. The first is when the coordinator of the Zigbee network uses a TCLK that the attacker does not have. For instance, when the coordinator uses a vendor-specific TCLK that the attacker does not know. This means that the attacker cannot decrypt the transport_key message and obtain the NWK. The second is when the NWK is never transmitted while the attacker can eavesdrop on the communication. For instance, the network can be established before any attacker-controlled device is present, or the network may never be in open state after the attacker is present.

Without access to the NWK, the attacker can still retrieve addressing information of the devices in the network. In fact, through passive sniffing of regular Zigbee packets, the attacker can obtain MAC and network addresses of the devices as well as the PAN ID. The attacker can also retrieve the Extended PAN ID (EPID) of the network. For instance, the attacker can broadcast beacon requests and wait for the beacon response, which contains the EPID, along with the notification related to the state of the network. Exploiting this information, an adversary can try to impersonate one of the devices, or even the coordinator, and send forged messages in order to disrupt the normal communication flow.

3.3 Attacker with the NWK

An attacker can obtain the NWK if the following two conditions are both satisfied. First, the attacker knows the TCLK used by the coordinator, either because the coordinator uses the GMK (which is public knowledge) as the TCLK, or because the vendor-specific TCLK used by the coordinator has already been exposed. Indeed, in many inter-manufacturer device communication, the devices still employ the GMK to perform the join procedure. In addition, an attacker could retrieve a private TLCK by reverse engineer any authorized Zigbee device or by external sources if the key is leaked (for instance, the ZLL Master key was leaked in March 2015 [22]). Hence, we can assume that an adversary with reasonable means can obtain either the GMK or the TCLK. Second, the attacker is present when the network is in the open state, accepting new devices to join the network. If an attacker appears when the network is in the closed state, the attacker can either monitor the network state and wait until it is changed to the open state by the user, or try to perform some actions (through some out-of-band channel) to cause the user to open the network. When the above two conditions are satisfied, an active adversary can exploit the knowledge of both the TCLK and the NWK to perform several attacks that we categorize into three main categories:

- **A1** User privacy leakage
- **A2** User device control
- **A3** Fake device injection

In the following, we discuss these classes of attacks separately.

A1. User privacy breaches. The attacker decrypts every message transmitted inside the victim's Zigbee network with the NWK and accesses their content. This may include sensitive information related to door locks, cameras and sensors, security tools such as gas/smoke detectors, alarms. Thus, the adversary has access to a broad set of data that could be exploited for further attacks. Nonetheless, since different end devices, manufactured by different vendors, use the same NWK key to communicate with the coordinator in the same Zigbee network, this potentially allows an end device (e.g., smart light) to learn the secure communication between the controller and other devices, such as smart camera and door lock. This violates device's data privacy and thus enables a compromised or honest but curious device to expose other devices' communication to an adversary.

A2. User device control. The attacker forges and spreads valid packets through the network. Knowing the NWK, the attacker can impersonate the coordinator of the network, by setting up a fake device with the same MAC and network addresses as the coordinator, and send specially crafted commands to a target device. Since the messages are correctly encrypted, the target device accepts them as valid and executes the commands. To understand the severity of this vulnerability, we provide two example scenarios. In the first scenario, the adversary takes control over a smart door lock and can open/close the device at any time. In this case, the attacker can lock a victim out of the house or open the main door while no one is inside. In the second scenario, the attacker takes control of security devices such as sensors and cameras. In this case, the attacker has access to sensitive information about the victim that could be further exploited for monetary purposes. As the attacker can gain control over single devices, he can also extend his attack scope and aim for the entirety of the Zigbee network. If such an event were to happen, the victim's network would be utterly compromised.

A3. Fake device injection. For this type of attack, the attacker only needs the TCLK, but requires to be present when the network is in open state. Since the coordinator does not verify the identity of a joining device, an attacker that is present in an open state network can join the network with a fake device. The knowledge of the TCLK allows the attacker to correctly complete the join procedure and receive the NWK transport message which he can decrypt with the TCLK. Once the attacker has joined the victim's network, he can silently monitor the network and obtain sensitive information.

Moreover, after obtaining the NWK, the attacker can perform the attacks discussed in **A1** and **A2**.

4 Analyzing the Solution Space

In order to enhance the security of the Zigbee protocols, the Zigbee Alliance has introduced additional security features (e.g., installation code), and previous researchers have also proposed cryptographic enhancements to the protocols. However, such solutions either fail to adequately defend against the attacks we discussed in the previous section or exceed the resource constraints of typical Zigbee devices. In the following, we discuss some of the current solutions and their limitations.

4.1 Security features by Zigbee Alliance

We first discuss the security features outlined by the Zigbee Alliance and their limitations.

Zigbee Link keys. According to the Zigbee specification [2], an end device can request the coordinator to generate a link key to be used between the requesting device and a neighboring end device. The coordinator will send the link key encrypted under the NWK to both devices, which can use the link key to encrypt communications between them. However, since the message that carries the link key is encrypted with the NWK, an attacker who has access to the NWK can easily obtain the link key and perform attacks, such as sensitive data theft (**A1**) and device control through forged messages (**A2**). Thus, although unique link keys allow encrypting each device-to-device communications, the key exchanging mechanism fails to prevent third devices in the network, both benign and malicious, from retrieving them. In summary, link keys fail to improve the security of the protocol.

Installation code. A key challenge in Zigbee security is how to authenticate a device that is about to join a network. When a user intends to add a device to a network, how does the coordinator distinguish that device from other (potentially malicious) devices that are also within the communication range. To solve this problem, Zigbee 3.0 introduces a new join mechanism called *installation code* to enable more control over the joining devices. Zigbee installation code, also referred to as install code, is an 18-byte value (a 16-byte random key + 2-byte CRC) put on the outside of a device (e.g., in the form of a QR code). The code is also stored inside the device. The user can scan the code using a smartphone application, which then sends the code to the coordinator. According to the specifications ([18]), the installation code is used as a pre-configured secret, from which a link key is derived through the AES-MMO algorithm. The link key is then used to encrypt the initial transport_key message from the coordinator to the device during the join procedure. Only devices that know the code can decrypt the transport_key message and join the network.

While enabling the identification of a joining device, the installation code mechanism, however, still suffers from several vulnerabilities. First, as the installation code is printed on the outside of a device, an attacker who has a malicious device in the same environment may use a camera or scanner to obtain the installation code, and derive the secret key. The attacker is then able to decrypt the transport key message and obtain the NWK, which leads to **A1** and **A2**. Second, even if the attacker is not present at the joining time, the mechanism fails to provide forward secrecy, which means that an adversary can log the encrypted communication and later retrieve the installation code and decrypt the messages (for instance, to perform behavioral analysis). In conclusion, the installation code in the current standard does not provide adequate protection in terms of Zigbee security and privacy vulnerabilities.

Touchlink. Touchlink [2] is another authentication procedure that is included in the current Zigbee standard. In Touchlink, the user places a new connecting end device within a short distance from the coordinator so that the latter can verify its identity. In other words, Touchlink provides an authentication scheme based on vicinity. Although this approach seems reasonable in the context of domestic networks, it is not as acceptable in the context of industrial networks in which the end device could require to be located at a considerable distance from the controller.

4.2 PKI-based Proposals

In the following, we discuss why the existing proposals are inadequate and fail to mitigate the vulnerabilities in the Zigbee protocol.

Certificate-based model. It is obvious that the root cause of some of the vulnerabilities in Zigbee is that only symmetric cryptography is employed. As a result, adding a trusted device to the network while preventing untrusted devices in the environment to join is challenging. Some researchers have proposed to solve this problem using a public key infrastructure (PKI) [24]. In this case, each device has a pair of public/private key, and a certificate signed by the manufacturer. During the joining process, the coordinator verifies the end devices certificate to check whether it is authorized.

The problem of using PKI is that it fails to distinguish the device a user wants to add to the network, from the billions of other devices that could be added to the network. In an open ecosystem with many vendors, the number of devices certified through a valid certificate chain is vast, and it is easy for attackers to have malicious devices that are certified through certificate chains, either by modifying the software on a device or by becoming a vendor. Also, there are challenges in installing the root certificates of all vendors in resource constrained Zigbee devices at the production phase. This seriously affects the ubiquity and the re-usability of these devices when a device wants to communicate with another device but is not provisioned with the vendor's root certificate of the other device.

Certificate-less model. Choi *et al.* [8] and Hassan *et al.* [15] proposed to use ECDH for secure key distribution and subMAC (constructed with selective bits of an HMAC) to prevent MitM attacks. These proposals, however, have several limitations. First, the attacker can drop the original packets and send the fake ones to carry out MitM attacks. Second, the attacker can still perform **A3** attacks with their solutions in place since neither the end device nor the coordinator can identify when the attacker injects a fake device into the target network. Finally, it is not clear how the secret keys can play the role of NWK as suggested by their solutions, because NWK needs to be shared with all devices in a network so that broadcast messages can be decrypted by every device. If all devices in the network share the same key, then the protocol is still under attack of **A1, A2** and **A3**.

Unlike certificate-based schemes, Tedeschi et al. [29] proposed a certificate-less scheme dubbed *Like* that uses ECDH to create the session link key, and Domain Authority (DA) to authenticate the devices. Compared to the previous scheme, LiKE maintains NWK for broadcast messages, and can prevent attackers' fake devices joining the network. This approach, however, fails to identify an adversary-controlled device impersonating a benign device. In addition, the effectiveness of this solution depends on the centralized Domain Authority which is responsible for assigning the initial secret to all Zigbee devices regardless of vendors. Centralized Domain Authority is, however, impractical due to deployment challenges and single point of failure.

5 Proposed Defense

Based on the vulnerabilities related to NWK and TCLK, we need to reconsider how devices in the network share and employ secret keys. In addition, we also need to consider the rigid constraints that the Zigbee environment poses over the possible solution space due to limited devices' resources and capabilities and efficiency requirements, although some of them can be relaxed without risking to impact the protocol drastically. Our proposal tries to effectively respond to the three categories of vulnerabilities **A1**, **A2**, and **A3**.

A1 and **A2** are vulnerabilities related to the use of a single shared NWK to encrypt all the communication in the network. Therefore, we aim at minimizing the usage of the NWK and introducing link keys related to specific coordinator-to-device communication. Furthermore, secure agreement on the shared link key should not depend on the secrecy of the NWK and should provide forward secrecy. To achieve this goal, we propose to use a key exchange scheme based on Elliptic Curve Diffie-Hellman (ECDH) during the join procedure to establish a shared key between each pair of devices that direct communicate with each other.

Finally, to overcome **A3**, we propose to improve the installation code mechanism, already included in the latest Zigbee protocol, introducing a public/private key scheme, in which the installation code encodes the end device's public key. This avoids the risk of undesired devices to join the network, as the coordinator only allows devices for which the user has scanned the code. In what follows, we describe our proposed defense in detail.

ECDH Link keys. Our proposed defence is motivated by two main flaws of the current protocol:

(1) link keys are exchanged under the encryption of the NWK, which defeats the purpose of using link keys in the first place.
(2) the NWK is encrypted with a pre-installed TCLK, which can be exposed and compromise the entire network;

In order to fix the first flaw, we change the order in which the keys are exchanged. That is, the link key is exchanged first, and then it is used to encrypt the NWK in the transport key message. This approach avoids having private information, such as a link key, being accessible by virtually every device in the network. However, it raises the problem of the second point, which is how to exchange the link keys securely. To achieve such a goal, we eliminate the use of pre-installed keys, and we introduce ECDH in the key exchange scheme.

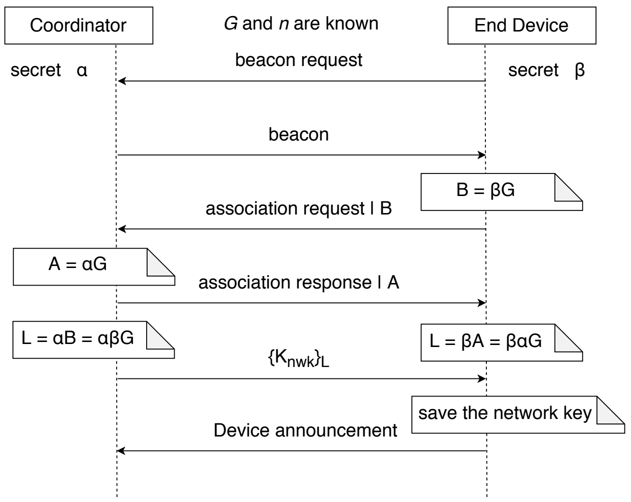

Figure 4: Join procedure with Elliptic Curve Diffie-Hellman key exchange protocol.

Network key. Link keys ensure that each end-to-end communication privacy is preserved, against either an external or an internal threat. However, the Zigbee protocol also involves broadcast messages. Using link keys to encrypt the same message for each device individually would be too expensive in terms of both resources and time. Hence, in our model, we maintain the NWK, but, differently from the current protocol, the key is used only to encrypt/decrypt broadcast messages. As before, the coordinator provides the NWK to each new device during the join procedure, but in our proposal the NWK is encrypted with the link key rather than with the TCLK. Note that broadcast messages cannot carry commands for a specific device (e.g., open/close door lock), but only general network-related information (e.g., changing the network state from open to closed and vice versa, and device announcement).

5.1 ECDH Key Exchange in Zigbee

ECDH allows two parties to agree on a shared key without transmitting the actual key. Accordingly to the protocol, two Zigbee devices d_1 and d_2, can share a secret key as follows:

(1) d_1 and d_2 have the parameter of the elliptic curve pre-installed, including the generator point G, and n, an integer associated with the curve;
(2) d_1 generates a random secret integer α in the range $[1,\ n-1]$ and computes $A = \alpha G$;
(3) d_2 generates a random secret integer β in the range $[1,\ n-1]$ and computes $B = \beta G$;
(4) d_1 and d_2 exchange A and B;
(5) d_1 and d_2 compute $L_1 = \alpha B = \alpha\beta G$ and $L_2 = \beta A = \beta\alpha G$, respectively;
(6) since α and β are integers, $\alpha\beta = \beta\alpha$, hence, for the properties of the elliptic curves, $L_1 = L_2 = L$. As a result, d_1 and d_2 now share a secret key L.

The shared secret L is used as the link key to secure every end-to-end communication between two devices d_1 and d_2 in the network.

In Figure 4, we show the updated join procedure with ECDH. The coordinator and the connecting device exchange the values $A = \alpha G$ and $B = \beta G$. Specifically, A and B can be included in the association_response association response message and the association_request, respectively so that no additional message is required during the join procedure. Once the coordinator and the device have the other party's value, they can generate the shared secret key L. At this point, the coordinator can transmit the NWK in the transport_key message symmetrically encrypted under L. An adversary capable of eavesdropping on the conversation has no means to retrieve the value of the key L and, therefore, cannot decrypt the NWK. In our new model, the key L is employed as link key, thus, every device in the network (i.e., coordinator, router, or end device) must generate a different encryption key for each of the other devices with which it is communicating. This approach provides the following advantages:

(1) An adversary with the TCLK cannot retrieve the NWK;
(2) An adversary with the NWK cannot compromise any device since messages are expected to be encrypted with link keys;
(3) Devices inside the network cannot eavesdrop on the communications between other devices.

Security considerations. The key exchange protocol with ECDH successfully defends the system against **A1** and **A2** discussed in Section 3. However, the updated join procedure is still vulnerable to an adversary that can carry out a man-in-the-middle attack (MitM) between a coordinator and a joining device. This attack would allow the adversary to eavesdrop on all the communication between the two devices, send forged messages, and disrupt the connection. Given the ECDH key exchange scheme, an MitM attacker, however, has several limitations and constraints:

- As the communication is over the air, the attacker has to constantly jam the end device, which would otherwise detect the presence of a different coordinator in the same network;
- Even if an attacker successfully performs MitM, he can only compromise one single device;
- The adversary can only hope to perform the attack when the network is in open state, and it needs to schedule the messages accurately during the join procedure.

Nonetheless, the described scenario represents an actual vulnerability. Moreover, the new scheme based on ECDH does not provide any defense against **A3**, which means that an adversary can still inject fake devices into the network. Thus, we further improve our solution by leveraging the installation code mechanism.

5.2 Installation code and ECDH key exchange

Given the discussed MitM vulnerability and the lack of an authentication scheme, we propose to modify the current installation code mechanism and integrate a public key scheme as follows: each Zigbee device is provided with a pair of Elliptic Curve public and private keys, referred to as pk and sk respectively, and the installation code on its exterior encodes pk (note that with Zigbee 3.0, every compliant device must support installation code). Hence, during the join procedure, the coordinator can authenticate each connecting device with the Elliptic Curve Digital Signature Algorithm (ECDSA) through the following steps:

(1) The user scans the code on the end device's exterior through the smartphone application used to control the coordinator device. The coordinator receives the code pk from the former and stores in its memory.
(2) The end device forwards a message digitally signed with sk to prove its identity.
(3) The coordinator verifies the signature with pk. If the signature is valid, the coordinator authenticates the end device and continues with the join procedure. Otherwise, the procedure is interrupted and the end device is rejected.

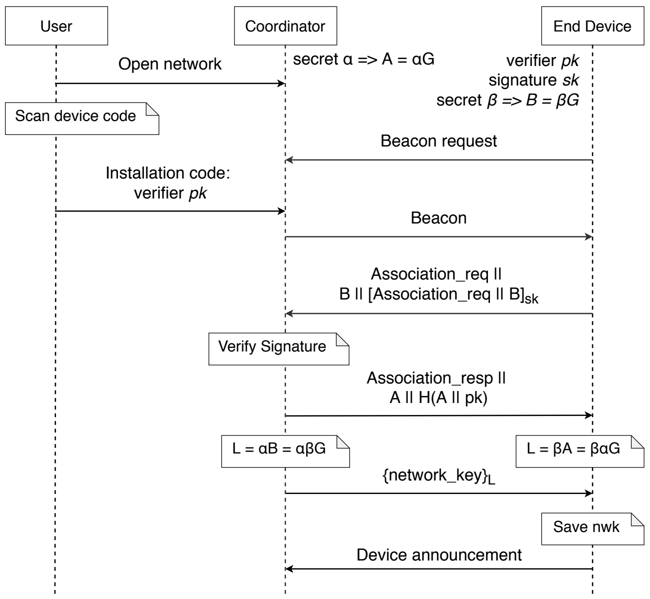

Figure 5: Zigbee join procedure with ECDH key exchange protocol and Public-Key installation code.

This public key-based model can prevent an adversary from injecting a fake end device in the Zigbee network (**A3**). However, the model itself, when combined with the current Network key-based security scheme proposed by Zigbee Alliance, cannot guarantee perfect forward secrecy. Moreover, an attacker who possesses the NWK can steal sensitive user data (**A1**) and take control over target devices (**A2**).

Therefore, we propose to combine this new installation code scheme with our proposed ECDH key exchange procedure.

Public-key installation code and ECDH Link Key exchange. In order to combine the two solutions, we employ the ECDSA authentication procedure together with the ECDH key exchange within the Zigbee join procedure. Specifically, we propose to leverage the installation code to authenticate the association_request message and the end device's ECDH part of the key. Moreover, the end device can verify that no one tampered with the coordinator's association_response message by exploiting the fact that the latter must know the end device's EC public key (pk). We present the entire new join procedure in Figure 5. The two crucial steps that differ from the current version of Zigbee are (1) the association_request message and (2) the association_response message.
(1) Along with the association_request, the end device also sends its part of the ECDH key ($B = \beta G$). As we want the coordinator to authenticate the device, the latter must also compute the Hash of the entire message ($association\_request || B$), sign it with its private

key sk, and concatenate the signature with the request. Thus, the coordinator can verify the signature with pk and authenticate the end device.
(2) Similar to the previous case, the coordinator forwards its part of the ECDH key ($A = \alpha G$) along with the association_response message. Since this time we want the end device to verify the coordinator, the latter must also compute the Hash function of A concatenated to the end device's private key pk, and send the result along with the rest of the message. Hence, the end device can verify the Hash and ensure that the key A was not altered, and the message belongs to the coordinator that possesses pk.

Fake coordinator scenario. So far, when we consider an attacker who is able to inject fake devices into a target network, we have assumed that the injected device is an end device, and the coordinator of the network is always under control of the user. We now consider the scenario in which the attacker controls a fake coordinator and tricks a user's end devices into joining a network setup by it, rather than the network of the real coordinator. We assume that our defense has been deployed and is operative, and the adversary knows the public key pk of the target end device. The adversary sets up a fake coordinator and lets the target device to join in. Since he knows pk, he can correctly perform the join procedure so that the end device happily joins the fake coordinator. However, for this attack, the adversary requires strong capabilities, as he needs to set up a controller with the appropriate configuration and be close enough for the device to detect it. Moreover, the adversary must intercept the target device during the join procedure and hope for the device to join the fake network rather than the legitimate one. Given these constraints, the attacker with the needed capabilities can potentially perform the attack. Therefore, in order to protect end devices from such an event, we exploit the *factory state*. According to the standard, every device can get back to the original "factory state". Hence, we can force the end device to use the installation code only during the very first join. After the device has performed the first join procedure, it switches to "*user state*" and performs every consecutive join using the last generated link key. If the join is not possible, the user can reset the device to factory mode and re-perform the first installation. Such a mechanism forces the device to use the installation code only in factory mode, that is, an adversary who knows the public key of the target device can take control over it only during the very first join procedure, as every other join will be performed with the secret link key.

6 Formal Security Analysis

In order to verify the correctness of our proposed protocol, we use ProVerif [6], the state-of-the-art tool for automatic cryptographic protocol verification. First, we assume that the cryptographic primitives that we leverage are inherently robust. Specifically, we assume the correctness of four primitives:

- Elliptic curve asymmetric cryptography;
- ECDSA signature-verifier scheme;
- AES-CCM* symmetric encryption;
- Hash function (i.e., pre-image resistance and collision resistance properties).

Given such assumptions, we evaluate our defense with ProVerif under two scenarios: an adversary that does not possess the target device's installation code, and an adversary that possess the target device's installation code. The attacker model provided by ProVerif is compliant with the Dolev-Yao model [10], that is, the attacker can read, modify, delete, and inject messages on the communication channel. Therefore, the attacker can obtain all the public values, such as A and B (i.e., the partial key of the coordinator and the end device) and the hash values that are sent in clear. We model all the messages involved in the new protocol, and we run ProVerif. In both the scenarios, the attacker can correctly retrieve A and B but has no means to access either the link key L or the NWK. On the other hand, the end device receives the NWK correctly. In conclusion, we formally prove that the protocol we propose is robust against both a passive or an active adversary that aims at obtaining either the link key or the NWK.

For comparison, we also model the current Zigbee protocol under the assumption that the attacker knows the TCLK. As expected, ProVerif output shows that the attacker can retrieve the NWK and, subsequently, any link key.

7 Evaluation

In this section, we discuss the implementation and evaluation of the new Zigbee protocol.

7.1 Environment setup

Beyond computers, we use four devices to experiment with sending, receiving, and capturing Zigbee packets. We use two Universal Software Radio Peripheral 210 (USRP210) [25] devices. One device acts as the coordinator, and the other as an end device. The USRP210 platform provides an AD9361 RFIC direct-conversion transceiver with up to 56MHz of real-time bandwidth and an open and re-programmable Spartan6 FPGA. This device is capable of radio communication with frequency coverage from 70 MHz to 6 GHz, which covers the frequencies used by Zigbee protocols. We also use an ApiMote v4 Beta [26] device (which is less powerful than USRP210) as a sniffer to capture the Zigbee packets for further analysis. Last, we use a Raspberry Pi 3B to simulate the performance of Zigbee devices. It has Quad-Core 1.2GHz Broadcom BCM2837 64bit CPU and 1GB RAM. We use this device instead of a PC or laptop to measure the time it takes for cryptography-related operations, such as Elliptic curve point multiplication, ECDSA signature verification, and hash function calculation. This device is 10 times slower than a PC with an i7-3770 CPU and 16GiB memory for Elliptic curve point multiplication. All the devices are shown in Figure 6.

The USRP devices are connected to a computer via USB and controlled by Z3Sec [27], which is a penetration testing software framework designed by Philipp Morgner of Friedrich–Alexander University Erlangen–Nürnberg in 2017. Z3sec uses the capabilities of GnuRadio to send and receive ZigBee packets via the USRP. Z3Sec was initially designed for analyzing the Zigbee light link protocol, which is similar to Zigbee Home automation but is only suitable for Smart Light join procedure. We leverage the physical layer and MAC layer designs provided by Z3Sec, whereas for the network and application layers, we implement the new Zigbee protocol thoroughly. Z3Sec invokes GNU radio to provide signal processing blocks for USRP to send and receive packets.

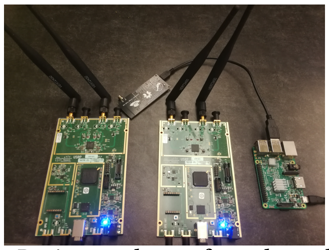
Figure 6: Devices used to perform the evaluation.

7.2 Performance of the new features

We have three new features for the improved Zigbee protocol, which are responsible for key exchange, device authentication and device verification. We explain the details of the performances of these new features below and the results are in Table 1.

Implementation of the ECDH. We employ the Elliptic Curve Diffie-Hellman algorithm (ECDH) to generate a secret, which is used as a link key in the new Zigbee protocol. We pre-store the curve information (g, p, n), the private values (α and β), and the public point (A and B) for further use. Hence, during the join procedure, both the coordinator and the end device only have to perform the point multiplication after receiving the other party's public point. The length of the ECDH point is decided by the curve. In Table 1, we show the performances of the main operations involved in the new protocol. We implemented the feature with Python based on TinyEC library. We evaluated the performances on five curves, and chose the brainpoolIP256 curve, which has the lowest overhead and can produce 128-bit long secret.

Implementation of Hash function. The new protocol requires Hash functions to check the ECDH public point against manipulation. In our new protocol, the association_response message requires the Hash function to bind the sender information with the public key. We select the hashlib library and evaluate the performances of three hash functions implemented with Python. The three functions are SHA224, SHA256,and SHA384 [11]. In Table 1, we can see that the SHA hash function takes quite a short time of computation compared to the time required for ECDH and ECDSA. We select SHA256 for our experiments, because it can produce the suitable Hash length (32 Bytes).

Implementation of ECDSA. In our new protocol, we change the value of the installation code from the pre-configured secret to the verifier of the device. The signature and verifier pair is generated by the Elliptic Curve Digital Signature Algorithm (ECDSA) [17]. In the implementation, we select Warner's ECDSA library and test six curves implemented in Python. We test the time cost for both signing and verifying. In Table 1, we notice that the verifying time is longer than the signature. Since ECDSA and ECDH are both using Elliptic Curve, and if we select the same curve, we do not need to store multiple constants. In our experiments, we select brainpoolIP256 curve.

7.3 New protocol overhead

We evaluate computational, memory, and communication overhead of our new Zigbee protocol. We consider both the case with installation code and the case without installation code. We run our evaluation employing the brainpool256r1 curve for ECDH and ECDSA, and SHA256 for the hash function.

Function	Parameters	Time (ms)	memory (code)	memory (const)	communication	energy overhead
ECDH	brainpool160r1	111.5	10.1 KB	120 Bytes	11 Bytes	around 0.57mW
	brainpool192r1	154.4		144 Bytes	13 Bytes	
	brainpool224r1	215.7		168 Bytes	15 Bytes	
	brainpool256r1	279.4		192 Bytes	17 Bytes	
	secp256r1	283.2		192 Bytes	17 Bytes	
SHA	SHA224	0.009	7.8 KB	288 Bytes	24 Bytes	N/A
	SHA256	0.009		288 Bytes	32 Bytes	
	SHA384	0.009		288 Bytes	48 Bytes	
ECDSA	brainpool160r1	4.32+17.25	23.2 KB	120 Bytes	20 Bytes	around 0.8mW
	brainpool192r1	5.81+23.55		144 Bytes	48 Bytes	
	brainpool224r1	7.67+31.75		168 Bytes	56 Bytes	
	brainpool256r1	9.98+42.50		192 Bytes	64 Bytes	
	secp256k1	9.51+39.15		192 Bytes	64 Bytes	
	NIST256P	9.97+40.41		192 Bytes	64 Bytes	

Table 1: The performance of three functions in the new protocol. The key components contain time of operation, code and constant memories, communication overhead, and energy overhead.

Computational overhead. In the new protocol without install code, we only require to add an ECDH secret key in the association request message for the end device and a ECDH secret key in the association response message for the coordinator. The computation time for the ECDH point multiplication is about 279.4 *ms*. Note that the coordinator and the end device can compute the ECDH point multiplication in parallel, that is, we consider the time overhead for ECDH only once. In the new protocol with install code, we need two more hash calculations and a signing & verifying for two nodes. The end device requires to sign the public secret, and the coordinator verifies it. For the hash function, we select the SHA256 function, which costs about less than 0.2 *ms* for two operations. We need approximately 52.5 *ms* for signature and verifier, and a total 331.9 *ms* for the whole join procedure. To provide a comparison, in the original Zigbee protocol, the entire join procedure consists of the user opening the network, the coordinator scanning the channels, two devices establishing the communication, and, finally, the coordinator sending the NWK. By the standard joining procedure, the coordinator first scans the total sixteen channels, each takes 1/3 to 1 seconds [13]. Then it will cost around 0.5156 seconds for the secured key exchange [7]. The total joining time for the standard protocol is around 5.89-16.51 seconds. Our protocol overhead is between 1.69% - 5.62%, and 3.8% on average.

Memory overhead. In the new protocol without installation code, we need to save the constants and the codes for ECDH only. The constants are 192 Bytes and the codes are 10.1 KB (Table 1). With the installation code, we store the extra codes for the hash function and the ECDSA (23.2 KB), and constants for the hash functions (7.8 KB). The constants for ECDSA are the same as those for ECDH and we do not need to save them twice. We notice that the size of the current stack varies due to the implementation. As a reference, the total size of Texas Instrument Z-stack is larger than 4MB, and our memory overhead is very small.

Communication overhead. In the new protocol without installation code, we send ECDH public keys in the association request and association response messages. The total overhead for the communication is 17 Bytes. When we add the installation code, we have 81 Bytes communication overhead for the association request, including one ECDH public key and one ECDSA signature, whereas we have 49 Bytes overhead for the association response, including one public key and one hash result. In total, the communication overhead for the new protocol is within the maximum limit of 127

	w/o install code	w/ install code
total operation overhead	1 ECDH	1 ECDH + 2 Hash + 1 ECDSA
total time overhead	279.4ms	331.9ms
memory overhead(code)	10.1 KB	41.1 KB
memory overhead(consts)	192 Bytes	480 Bytes
communication overhead	17 + 17 Bytes	81 + 49 Bytes

Table 2: The overall performance of the new protocol. We compare operation overhead, total overhead, code and constant memory overhead, and communication overhead with and without install (or installation) code.

Properties	This paper w/o install code	This paper w/ install code	[29]	[8]	[24]
Extra messages	0	0	4	3	New protocols
CA / DA required	Neither	Neither	DA	Neither	CA
A3 defense	No	Yes	No	No	Yes
Operation overhead	2 ECDH	2 ECDH + 2 Hash + 1 ECDSA	3 ECDH+ 5 Hash + 2 ECDSA	2 ECDH	PKI online verification
Memory overhead (Code + Constants)	10.1 KB + 192 Bytes	41.1 KB + 480 Bytes	42KB + 13.594 KB	10.2 KB + 192 Bytes	
Communication overhead	17 + 17 Bytes	81 + 49 Bytes	400 Bytes	300 Bytes	

Table 3: The comparison among the new protocol presented in this paper and the state of art schemes of Zigbee protocol.

Bytes supported by Zigbee. Therefore, the increased size packet results to be a much more suitable solution compared to introducing additional messages. As the standard data rate for Zigbee is 250 Kbit/sec and the whole join process will send out more than 517 Bytes, the communication overhead is rather small.

Energy overhead. The standard Zigbee joining procedure contains beacon scanning and key exchanging. With a standard Zigbee device using a 5 voltage battery, the average energy consumption for the joining procedure is around 60mW [31]. In the new protocol without installation code, we need to calculate two more ECDH multiplications, which cost 0.57mW [9, 14]. The overhead is less than 1%. For the protocol with install code, the ECDSA operation will take about 0.8mW for the signature generator and verifier [1]. The total energy overhead is less than 2.5%, thus, it is negligible when compared to the Lithium CR2477 Battery with 850-1000mAh used in Zigbee devices, which usually runs for 3-7 years [4].

7.4 Comparison with existing proposals

In the following, we compare our new protocol with the proposals discussed in Section 4. We consider the following properties: the number of extra messages required for the new protocol; whether it requires a CA or DA; whether **A3** can be prevented by the new protocol; operation overhead; memory overhead; and communication overhead. The details of the result are in Table 3.

Extra messages. Our new protocol does not require extra messages compared to the original protocol. As such, our new protocol has less communication overhead than other existing solutions. Indeed, some other approaches require extra messages, which introduce heavier communication overhead, and require many changes to existing protocol. For instance, [29] and [8] have 4 and 3 extra messages correspondingly compared with the current protocol.

CA/DA requirement. [29] and [24] require a centralized CA or DA, whereas our new protocol does not, as it is based on a certificate-less model. A CA/DA-based protocol would substantially slow down the join procedure, and would not prevent an unauthorized device from joining the network. If an adversary can compromise a device or the CA/DA, it can join the network and obtain the NWK. Finally, a centralized CA/DA for Zigbee networks with devices from different vendors is hard to establish, as it requires mutual trust among all the vendors.

*Defense **A3**.* To prevent **A3**, the coordinator has to distinguish between the real devices trying to join the network and other devices near the environments. The devices can be attackers' compromised devices or other unintended free devices. Based on the limitations of Zigbee devices, Zigbee join procedure is totally automatic and the user has no control over which device can join the network. As a result, only improving the encryption scheme [8, 24, 29] cannot solve **A3**, and it is necessary to introduce an authentication mechanism that leverages user interaction.

8 Related Work

Authenticating and verifying a new device when it tries to join a network and securely delivering the network key from the coordinator to the new device are challenging tasks. In the following, we discuss some of the most relevant solutions proposed in prior literature.

Cryptographic solutions. Previous researchers proposed to improve the security of the join procedure by implementing a more secure key exchange algorithm. [16, 24, 28–30] designed new key exchange protocols that involve either certificate-less public key cryptography models or certificate-based key exchange models. Although they can defend against a passive adversary sniffing private information, the encryption scheme they propose cannot distinguish between real and fake devices, which means that an active adversary can impersonate an end device or a coordinator.

Password-based solutions. In order to solve the problem of malicious device injection, previous work introduced password-based schemes to authorize the devices in the network [19–21]. Such schemes require the device to provide a user interface for entering the password. However, due to their limitations, many Zigbee devices lack such an interface.

9 Conclusion and Future Work

In this paper, we introduce two cryptographic enhancements to Zigbee security protocols. We have implemented the proposed protocols, and evaluated them. Our evaluations provide substantial evidence that these enhancements are feasible, and are effective in defending against passive and active attackers.

In the future, we plan to deploy the enhanced protocol in the wild and evaluate its performances in complex environments. Also, we will extend our solution to provide effective defenses against the attacks that we left out of the scope of this work.

Acknowledgement

We thank the anonymous reviewers for their suggestions. This work was supported by NSA's Science of Security Lablet program through North Carolina State University.

References

[1] Mishall Al-Zubaidie, Zhongwei Zhang, and Ji Zhang. 2019. Efficient and Secure ECDSA Algorithm and its Applications: A Survey. *arXiv preprint arXiv:1902.10313* (2019).

[2] ZigBee Alliance. 2015. *ZigBee Document 05-3474-21.* Technical Report. ZigBee Alliance.

[3] Zigbee Alliance. 2020. *WHY Leading companies choose Zigbee.* https://zigbeealliance.org/

[4] batteryequivalents.com. 2020. *Lithium CR2477 Battery - Equivalents and Replacements.* https://www.batteryequivalents.com/lithium-cr2477-battery-equivalents-and-replacements.html

[5] Ana Bera. 2019. *80 IoT Statistics (Infographic).* https://safeatlast.co/blog/iot-statistics/

[6] Bruno Blanchet. 2001. An Efficient Cryptographic Protocol Verifier Based on Prolog Rules. In *14th IEEE Computer Security Foundations Workshop (CSFW-14).* IEEE Computer Society, Cape Breton, Nova Scotia, Canada, 82–96.

[7] Kyung Choi, Mihui Kim, and Kijoon Chae. 2013. Secure and Lightweight Key Distribution with ZigBee Pro for Ubiquitous Sensor Networks. *International Journal of Distributed Sensor Networks* 9, 7 (2013), 608380.

[8] Kyung Choi, Minjung Yun, Kijoon Chae, and Mihui Kim. 2012. An enhanced key management using ZigBee Pro for wireless sensor networks. In *The International Conference on Information Network 2012.* IEEE, 399–403.

[9] Nilanjan Dey, Parikshit N Mahalle, Pathan Mohd Shafi, Vinod V Kimabahune, and Aboul Ella Hassanien. [n.d.]. Internet of Things, Smart Computing and Technology: A Roadmap Ahead. ([n. d.]).

[10] D. Dolev and A. Yao. 1983. On the security of public key protocols. *IEEE Transactions on Information Theory* 29, 2 (March 1983), 198–208. https://doi.org/10.1109/TIT.1983.1056650

[11] Donald Eastlake and Paul Jones. 2001. US secure hash algorithm 1 (SHA1).

[12] GE. 2017. *ZigbeeFAQ.* https://products.gecurrent.com/sites/products.currentbyge.com/files/documents/document_file/DT302-daintree-zigbee-security.pdf

[13] Drew Gislason. 2008. *Commissioning ZigBee Networks.* https://www.sciencedirect.com/science/article/pii/B9780750685979000082

[14] Tarun Kumar Goyal and Vineet Sahula. 2016. Lightweight security algorithm for low power IoT devices. In *2016 International Conference on Advances in Computing, Communications and Informatics (ICACCI).* IEEE, 1725–1729.

[15] Nabaa A Hassan and Alaa K Farhan. 2019. Security Improve in ZigBee Protocol Based on RSA Public Algorithm in WSN. *Engineering and Technology Journal* 37, 3 B (2019), 67–73.

[16] Debiao He, Sahadeo Padhye, and Jianhua Chen. 2012. An efficient certificateless two-party authenticated key agreement protocol. *Computers & Mathematics with Applications* 64, 6 (2012), 1914–1926.

[17] Don Johnson, Alfred Menezes, and Scott Vanstone. 2001. The elliptic curve digital signature algorithm (ECDSA). *International journal of information security* 1, 1 (2001), 36–63.

[18] Silicon Labs. 2019. AN1089: Using Installation Codes with Zigbee Devices. Available at https://www.silabs.com/documents/public/application-notes/an1089-using-installation-codes-with-zigbee-devices.pdf.

[19] Leslie Lamport. 1981. Password authentication with insecure communication. *Commun. ACM* 24, 11 (1981), 770–772.

[20] Albert Levi, Erhan Çetintaş, Murat Aydos, Çetin Kaya Koç, and M Ufuk Çağlayan. 2004. Relay attacks on bluetooth authentication and solutions. In *International Symposium on Computer and Information Sciences.* Springer, 278–288.

[21] Zhiwei Li, Warren He, Devdatta Akhawe, and Dawn Song. 2014. The emperor's new password manager: Security analysis of web-based password managers. In *23rd {USENIX} Security Symposium ({USENIX} Security 14).* 465–479.

[22] @MayaZigBee. 2015. https://twitter.com/mayazigbee?lang=en Twitter.

[23] Victor S. Miller. 1986. Use of Elliptic Curves in Cryptography. In *Advances in Cryptology — CRYPTO '85 Proceedings*, Hugh C. Williams (Ed.). Springer Berlin Heidelberg, Berlin, Heidelberg, 417–426.

[24] Sudip Misra, Sumit Goswami, Chaynika Taneja, and Anandarup Mukherjee. 2016. Design and implementation analysis of a public key infrastructure-enabled security framework for ZigBee sensor networks. *International Journal of Communication Systems* 29, 13 (2016), 1992–2014.

[25] Ettus Research. 2019. *USRP B210.* https://www.ettus.com/all-products/UB210-KIT/

[26] riverloopsecurity. 2019. *APIMOTE.* https://www.riverloopsecurity.com/projects/apimote/

[27] Eyal Ronen, Adi Shamir, Achi-Or Weingarten, and Colin O'Flynn. 2017. IoT goes nuclear: Creating a ZigBee chain reaction. In *2017 IEEE Symposium on Security and Privacy (SP).* IEEE, 195–212.

[28] Jian Shen, Ziyuan Gui, Sai Ji, Jun Shen, Haowen Tan, and Yi Tang. 2018. Cloud-aided lightweight certificateless authentication protocol with anonymity for wireless body area networks. *Journal of Network and Computer Applications* 106 (2018), 117–123.

[29] Pietro Tedeschi, Savio Sciancalepore, Areej Eliyan, and Roberto Di Pietro. 2019. LiKe: Lightweight Certificateless Key Agreement for Secure IoT Communications. *IEEE Internet of Things Journal* (2019).

[30] Shengbao Wang, Zhenfu Cao, and Haiyong Bao. 2008. Efficient Certificateless Authentication and Key Agreement (CL-AK) for Grid Computing. *IJ Network Security* 7, 3 (2008), 342–347.

[31] Atmel MCU Wireless. 2015. *AT03663: Power Consumption of ZigBee End Device.* https://www.digikey.nl/en/pdf/a/atmel/power-consumption-of-zigbee-end-device

[32] Tobias Zillner and S Strobl. 2015. ZigBee exploited: The good, the bad and the ugly. *Black Hat–2015: https://www. blackhat. com/docs/us-15/materials/us-15-Zillner-ZigBee-Exploited-The-Good-The-Bad-And-The-Ugly. pdf* (2015).