A Formal Security Analysis of ZigBee (1.0 and 3.0)

Li Li[*]
Syracuse University
Syracuse, NY
lli101@syr.edu

Proyash Podder[*]
Florida International University
Miami, FL
ppodd002@fiu.edu

Endadul Hoque
Syracuse University
Syracuse, NY
enhoque@syr.edu

ABSTRACT

The rapid increase in the number of IoT devices in recent years indicates how much financial investment and efforts the tech-industries and the device manufacturers have put in. Unfortunately, this aggressive competition can give rise to poor quality IoT devices that are prone to adversarial attacks. To make matter worse, these attacks can compromise not only security but also safety, since an IoT device can directly operate on the physical world. Many recently reported attacks are due to the insecurity present in the underlying communication protocol stacks, and ZigBee is one of them. Considering the emergence and adoption of ZigBee 3.0 and the current market share of ZigBee 1.0, it is essential to study and analyze these protocol stacks at their specification level so that any insecurity at the specification level should be identified and fixed before they go into production.

With that goal in mind, in this paper, we develop a model for ZigBee (1.0 and 3.0) and reason about its security properties using a security protocol verification tool (named Tamarin). Our model of ZigBee closely follows the ZigBee specification, and the security properties are derived from the ZigBee specification. We use Tamarin to verify these properties on our model and report our findings on ZigBee 1.0 and ZigBee 3.0.

CCS CONCEPTS

• **Security and privacy → Formal security models**; **Logic and verification**; **Network security**.

KEYWORDS

ZigBee, protocol verification, symbolic analysis

ACM Reference Format:
Li Li, Proyash Podder, and Endadul Hoque. 2020. A Formal Security Analysis of ZigBee (1.0 and 3.0). In *Hot Topics in the Science of Security Symposium (HotSoS '20), April 7–8, 2020, Lawrence, KS, USA.* ACM, New York, NY, USA, 11 pages. https://doi.org/10.1145/3384217.3385617

[*]Both authors contributed equally and their names are in alphabetic order.

1 INTRODUCTION

The rapid growth in the number of IoT devices [5, 6, 23] in recent years indicates the amount of financial investment and efforts from the tech-industries and the device manufacturers toward widespread adoption of IoT devices [3, 4, 19, 25]. Unfortunately, this aggressive competition can give rise to poor quality IoT devices that are prone to adversarial attacks and exploitation [2, 7, 11, 12, 17, 18, 24, 32, 33]. These attacks can compromise not only security [31, 43] but also safety [21, 44], since an IoT device can directly operate on the physical world. Many recently reported attacks [9, 26, 29, 34, 35] are due to the insecurity of the underlying communication protocol stacks. Hence, it is imperative to analyze the existing as well as emerging communication protocols to reason about their security guarantees and limitations, prior to their wide adoption.

IoT devices rely on one or more communication technologies (*e.g.*, ZigBee, Z-Wave, BLE, WiFi), some of which are well-suited for resource-constrained devices to operate in low-power and lossy networks (LLNs) such as ZigBee and Z-Wave. Among them, ZigBee-based IoT devices are predominant in the IoT landscape (Half a billion ZigBee chips have been sold to date, which is projected to reach 1 billion chips by 2023 [46]). Security is one of the main concerns of ZigBee communication. A newer version of ZigBee emerged in 2015 dubbed ZigBee 3.0, which has been designed to improve the performance and security limitations of the prior version, ZigBee 1.0. Considering the current market share of ZigBee and the gradual adoption of ZigBee 3.0, we aim to focus on ZigBee specification and analyze its security guarantees. While prior work [16, 20, 27, 39, 42] have studied ZigBee enabled devices for finding vulnerabilities at the implementation level, those findings do not necessarily reflect the insecurity in the specification. Furthermore, their methodologies are often ad hoc and geared toward discovering low-level vulnerabilities, and cannot effectively be transported to formal analysis of the protocol stack.

In this paper, we aim to study and *formally* analyze ZigBee protocol stacks (ZigBee 1.0 and ZigBee 3.0) at their specification level, with the goal of verifying the security guarantees that ZigBee is supposed to provide. Any violation of these security guarantees will indicate underlying insecurity at the specification level, which must be fixed before that insecurity loop-hole creeps into many ZigBee devices. For our formal analysis, we first build a symbolic model of ZigBee by closely following the ZigBee specifications [49, 50]. This model primarily captures the important elements related to ZigBee security, for instance, key sharing, device joining, key update and so on. Next, we derive the security properties – based on ZigBee specifications as well as some common security assumptions – and check them against the model.

We utilize a security protocol verification tool (called **Tamarin** [37, 38]) to reason about the security properties against this model.

Tamarin is a powerful tool for the symbolic modeling and analysis of security protocols. It is already a proven tool for analyzing complex protocols like TLS 1.3 [14], 5G Authentication [8], and DNP3 [13]. Our model is compact as we were able to encode both ZigBee 1.0 and ZigBee 3.0 in a single model by leveraging the branching support in Tamarin. Our security analysis discovers some known security vulnerabilities in ZigBee 1.0 and demonstrates the absence of those vulnerabilities in ZigBee 3.0. We believe our formal model can be leveraged by other researchers in the community to reason about additional security properties of ZigBee 3.0. Our model and findings are available at [51].

To summarize our contribution:

- We develop a symbolic model of ZigBee by following the ZigBee specifications using the modeling language of Tamarin, a security protocol verification tool. The model primarily captures the important aspects of ZigBee security.
- We present the security properties derived from the ZigBee specifications and encode them for Tamarin so that it can check if the model satisfies these properties or not.
- Our analysis shows how ZigBee 3.0 overcomes the vulnerabilities found in ZigBee 1.0 by discovering the security vulnerabilities in ZigBee 1.0 and proving their absence in ZigBee 3.0.
- We believe our model and analysis can facilitate future research on ZigBee security.

The rest of the paper is organized as follows: the background on ZigBee and its security properties in Section 2; our model and the properties of ZigBee encoded for Tamarin and the threat model in Section 3; our analysis results in Section 4; a discussion on our analysis in Section 5; and relevant prior work in Section 6. Finally, we conclude in Section 7.

2 BACKGROUND ON ZIGBEE

In this section, we describe the architecture of ZigBee and some key concepts as these are necessary to understand our model correctly. Then, we also discuss the security properties of ZigBee.

2.1 Overview

We will describe ZigBee primarily based on the official ZigBee specifications [47, 49]. Besides, we have gathered information from other documentations, blogs, research papers too [28, 41, 48, 50].

2.1.1 ***Architecture.*** The ZigBee stack architecture (Figure 1) consists of multiple layers. Each layer performs a specific set of services for the upper layer. The two lower layers - physical (PHY) layer and the medium access control (MAC) layer are defined by the IEEE 802.15.4 standard. Above them, the ZigBee Alliance defined the network (NWK) layer and the application (APL) layer. The bottom of the application layer is the application support (APS) sub-layer, which provides services to the ZigBee device objects (ZDO) and manufacture-defined objects.

2.1.2 ***Actors.*** There can be three types of actors in ZigBee protocol: coordinator, router and end device.

Coordinator is the primary and basic part of a ZigBee network. Each ZigBee network must have one coordinator that is basically responsible for establishing, executing, and managing the overall

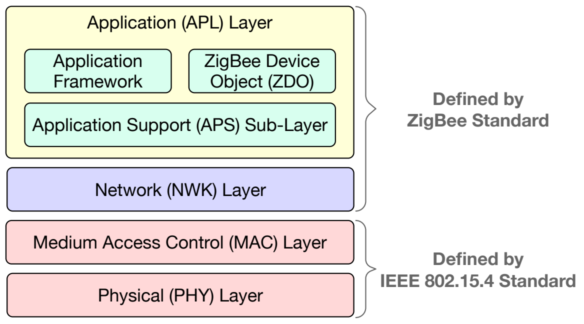
Application (APL) Layer
Application Framework
ZigBee Device Object (ZDO)
Application Support (APS) Sub-Layer
Network (NWK) Layer
Medium Access Control (MAC) Layer
Physical (PHY) Layer
Defined by ZigBee Standard
Defined by IEEE 802.15.4 Standard

Figure 1: ZigBee Stack Architecture

ZigBee network. Each device must have to get the coordinator's permission before joining the network. It is also responsible for the security management of the network. The coordinator manages all the keys that are used for encryption to build secure communication and distribute these keys to other devices.

Router behaves as an intermediate device between the coordinator and the end device. An end device can communicate with a coordinator directly or it can communicate via a router. A router can just route data between a coordinator and end device. Usually, a router is used when an end device cannot communicate with the coordinator directly. The router is an optional part of a ZigBee network since it is just used for routing data. It does not perform any type of key establishment or any significant task. So, for simplicity and without loss of generality, we will confine our model to an end device to coordinator communication.

End Devices are usually the devices we are familiar with (*e.g.*, smart switch, motion sensor, *etc.*). They do not have the ability to route data. They usually have limited computational capabilities and can only communicate with their parent (either a router or a coordinator). To become a part of the network, they have to get through the join process first with the coordinator.

2.1.3 ***Keys.*** ZigBee uses the symmetric-key cryptography. To be specific, it uses 128-bit AES-based encryption system [49]. Many keys are used in different sectors of communication in the ZigBee protocol. Here, we will introduce some of them.

Network key is used by a ZigBee device to secure outgoing NWK frames, and that is available for use to process incoming NWK frames. It is generated by the coordinator and distributed to all the devices. All the devices in a single ZigBee network share the same network key.

Link keys are used in unicast communication and applied by the APS (application support sub-layer) of the ZigBee stack [48]. Link keys can be of different types. Each end device has a pre-configured link key that is used to encrypt the network key when it is transmitted from coordinator to an end device. The pre-configured link key can be global or unique. After the sharing of the network key between the coordinator and device, the pre-configured link key might be replaced by a new trust center link key in the latest versions of

ZigBee. Two devices other than coordinator can use an application link key for securing their communication at the application level.

2.2 Sub-Protocols

In order to model and analyze the standard security mode of the ZigBee protocol, we will divide the whole protocol into some sub-protocols for better understanding.

2.2.1 ***Initial Key Generation.*** Every ZigBee network must have one coordinator, and the coordinator must generate a network key to initiate the network. This network key would be the same over the network, and the coordinator should send it to all the devices in the network. So, the first phase of a ZigBee network is the generation of a network key.

Then, when an end device joined the network, the coordinator sends the network key to it. But the key will not be sent in plain text. The coordinator encrypts it with the pre-configured link key. This key is possessed by the device and need to share with the coordinator. Now, there can be multiple situations. In ZigBee, a pre-configured global key (5A 69 67 42 65 65 41 6C 6C 69 61 6E 63 65 30 39) is used as the default trust center link key. It is simple because no key sharing is needed in this process. Both coordinator and device know it as it is global. But surely it is vulnerable as an adversary also knows it.

To provide protection against this vulnerability, in ZigBee 3.0, a so-called *Install Code* is used for sharing a unique link key. *Install Codes* are 128-bit of random data and a 16-bit cyclic redundancy check (CRC) codes which are passed through a Matyas-Meyer-Oseas (MMO) hash function to generate a trust center link key. This derived key would be used instead of the well-known global trust center link key. Generally, *Install Code* derived trust center link keys are hard-coded into joining devices during the manufacturing process. The corresponding *Install Code* is then included with the device and sends it to the coordinator through an out-of-band method such as a user interface [41]. The process of using *Install Code* is simple. For the end device, we create the *Install Code* and derive the pre-configured link key from it using the Matyas-Meyer-Oseas hash function. Then we configure this link key into the device. For the coordinator, the *Install Code* is sent out of band. And the coordinator derives the pre-configured link key from it also using the Matyas-Meyer-Oseas hash function. Then, the link key is installed in the Trust Center.

2.2.2 ***Join a Secured Network.*** In a ZigBee network, all the devices should possess the network key after fully joined in the network, and the coordinator is responsible for the distribution of the key. The network key is usually shared when an end device has been permitted to join a network. Here we describe the message communication for this sub-protocol (shown in Figure 2): The joiner device broadcasts a **beacon request** for network discovery. A coordinator that listens to the **beacon request** sends **beacon** to the joiner which contains some information about the coordinator including PAN (personal area network) ID, protocol version, *etc.* Particularly, the coordinator will use the *Association Permit* field in the **beacon** to indicate whether it allows other devices to join. Upon receiving the **beacon**, and if the value of the *Association Permit* field is *True*, the joiner sends an **association request** to the coordinator, and it will receive **association response** with *Association Status* field enclosed to indicate whether the association attempt was successful. Now, the device is in "joined but unauthorized" [50] state. After the coordinator decides to let the joining device into the network, the coordinator will send the network key (encrypted by a pre-configured link key) to the device using **transport key** command. When the device receives the network key, it becomes an authorized device for that particular network. A much more detailed description of joining a secured network presented in Appendix A.

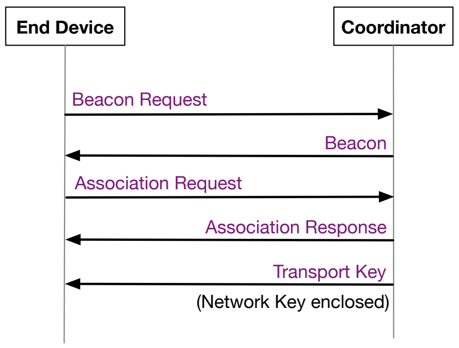

Figure 2: Join a Secured Network

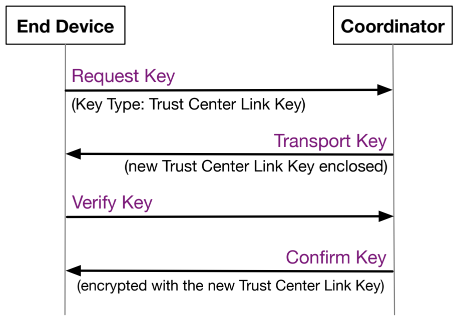
End Device
Coordinator
Request Key
(Key Type: Trust Center Link Key)
Transport Key
(new Trust Center Link Key enclosed)
Verify Key
Confirm Key
(encrypted with the new Trust Center Link Key)

Figure 3: Request New Trust Center Link Key

In ZigBee, an extra layer of security is added by updating the trust center link key. When a device successfully joins a network, it shall request a new trust center link key that is unique for the coordinator and that particular device. The coordinator uses this key to encrypt all the messages sending to the device at the application layer and vice-versa. Here we describe the message communication for this sub-protocol (shown in Figure 3): After a device joins a network, it requests for a new trust center link key by sending the coordinator a **request key** command with corresponding key type. The coordinator generates a trust center link key for that device and sends it to the device via **transport key** command that encrypted with the previous link key (pre-configured link key). The device then sends APS **verify key** to the coordinator for the verification of the new trust center link key. The coordinator then confirms the key by encrypting the **confirm key** command with the new trust center link key and sending it to the device.

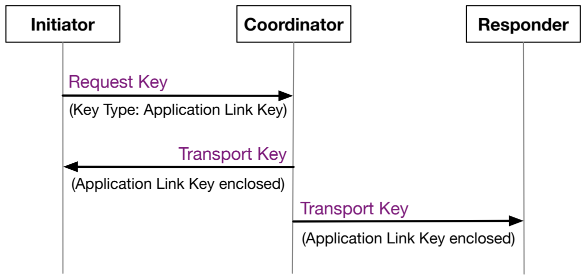
Initiator
Coordinator
Responder
Request Key
(Key Type: Application Link Key)
Transport Key
(Application Link Key enclosed)
Transport Key
(Application Link Key enclosed)

Figure 4: Establish Application Key

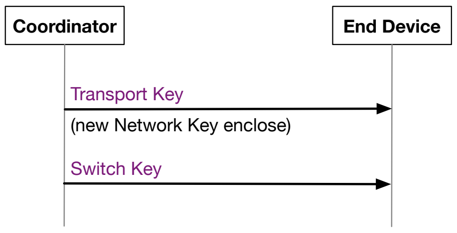

Figure 5: Update Network Key

After all these message exchanges, the device starts to use the new trust center link key, replacing the pre-configured link key for all the subsequent communications.

2.2.3 ***Application Key Establishment.*** ZigBee uses the so-called application link key to support end-to-end application security. To establish a link key between initiator and responder devices, a coordinator (Trust Center) needs to be involved. The message sequence is the following (also shown in Figure 4): The initiator device first sends a **request key** command to the coordinator. In this message, the initiator must set the key type (to application link key) and the partner responder's address. The coordinator generates a link key for the pair and sends it to the initiator and responder devices via **transport key** command respectively. From then, those two devices can use the newly generated application link key to achieve secured communication.

2.2.4 ***Network Key Update.*** ZigBee updates its network key periodically or when required. To prevent replay attack, ZigBee network contains a 32-bit frame counter that increments at every packet transmission [48]. The maximum value that a frame counter can be 0xFFFFFFFF. So, when this value reaches, no packet can be transmitted anymore, then a network key update happens, which resets the frame counter. As shown in Figure 5, the steps are: The coordinator generates a new network key and broadcasts it to all the devices through **transport key** command. When all the devices receive the new network key, the coordinator again broadcasts a **switch key** command. Upon receiving the command, all the devices start to use the new network key instead of the previous one.

2.3 Security Properties

ZigBee uses some security measures for key establishment, key transport, frame protection, and device management. These services are basically implemented in the network (NWK) and application support sub-layer (APS) layers. The ZigBee protocol is based on an "open trust" model. This means all protocol stack layers trust each other. Therefore cryptographic protection only occurs between devices. Every layer is responsible for the security of their respective frames [52].

According to the ZigBee specifications, *the level of security provided by the ZigBee security architecture depends on the safekeeping of the symmetric keys, on the protection mechanisms employed, and on the proper implementation of the cryptographic mechanisms and associated security policies involved. It is assumed that secret keys do not become available outside the device in an unsecured way. That is, a device will not intentionally or inadvertently transmit its keying material to other devices unless the keying material is protected, such as during key-transport* [49]. So, our first security goal will be to figure out whether the keys maintain the secrecy property or not.

We already know that the ZigBee protocol uses lots of keys. For the network layer encryption, ZigBee uses the network key. For the application layer encryption between the coordinator and device, it uses the trust center link key, and between two end devices, it uses the application link key. Besides, for the protection of the network key, while it transports from the coordinator towards the newly joining device, ZigBee uses a pre-configured link key (globally known or derived from the *Install Code*). So, we have to ensure that all the keys remain secret in a network in the presence of an active adversary.

Apart from the confidentiality of the keys, another critical security property is authenticity. We will also try to ensure the authenticity of the ZigBee protocol. In this case, we will follow the authenticity properties mentioned by Lowe [22], and briefly discuss the properties that Lowe mentioned in his paper. For example, two actors A and B is running the protocol. Now, the following properties should be fulfilled for authenticity: (i) *Aliveness*: a protocol guarantees to an initiator A *aliveness* of another agent B if, whenever A (acting as initiator) completes a run of the protocol, apparently with responder B, then B has previously been running the protocol. (ii) *Weak agreement*: Besides fulfilling the aliveness property, here we also have to include that B has been running the protocol with A. They might not run the protocol over the same data. (iii) *Non-injective agreement*: To satisfy this property, B has to run the protocol with A over the same data. (iv) *Injective agreement*: Finally, *injective agreement* adds the fact that, besides maintaining a non-injective agreement, it will also have to follow the fact that, for each run of A corresponds to a unique run of B.

3 MODELING ZIGBEE

In this section, we will first describe Tamarin, a tool that we use to model and verify the security properties of ZigBee (§ 3.1). Then we will briefly discuss about how we model ZigBee using Tamarin (§ 3.2), along with the threat model. Finally, we will present the encoding of the security properties that we intend to check using Tamarin (§ 3.3).

3.1 Tamarin

The Tamarin prover [37, 38] has been proven to be a powerful tool for symbolic modeling and security analysis of complex protocols like TLS 1.3 [14] and 5G Authentication [8]. At a high level, Tamarin

```
rule Key_Generation:
[Fr(~k)] --[Key_generated(~k)]-> [!Key(~k)]

rule A_Send_Message:
[!Key(k), Fr(~m)] --[A_Sent(~m)]-> [Out(senc(~m,k))]

rule B_Receive_Message:
[In(senc(m,k))]  --[B_Received]-> []
```

Figure 6: An Example of Protocol Modeling

```
lemma message_secrecy:
  "All m #i. A_sent(m) @ i ==> not Ex #j. K(m) @ j"
```

Figure 7: Message Secrecy Lemma

takes a security protocol model as the input, which specifies the protocol agents, the adversary, and the protocol's properties. To elaborate, it includes (i) the actions to be taken by the agents that run the protocol but in different roles (say, a client, a server, and a trusted third party), (ii) the capability of the adversary, and (iii) a description of the desired properties. Tamarin checks whether the given model fulfills the desired properties and can automatically construct a proof to support its findings: either *satisfaction* or *violation*.

Tamarin Modeling Language. Tamarin provides an expressive language to specify protocol agents and adversaries. The language is based on multiset rewriting rules [10] which can be used to define the protocol as a labeled transition system (LTS). Each state of this LTS symbolically encodes the protocol' state, the in-transit network messages, and the current knowledge of the adversaries. Protocol agents interact by exchanging network messages, which directly correspond to the transitions of the LTS. Tamarin supports a good number of cryptographic primitives out of the box, such as hashing, symmetric-encryption, asymmetric-encryption, and Diffie-Hellman. It allows arbitrary instances of the protocol agents to interleave in parallel. The properties are specified in Tamarin using first-order logic formulas, which allows quantification over messages and (logical) time instants.

Example Scenario. Suppose there are two agents A and B running a protocol and they share a key `k` between themselves for symmetric encryption. A will send a message `m` encrypting it with `k` and B will receive it. Figure 6 shows how to model this simple protocol using three rules. The notations and the rules are explained below.

Tamarin Rules. A rule has three parts: *premises* (left-hand side), *action facts* (middle) and *conclusions* (right-hand side) (see Figure 6). Each rule must be defined with a unique name (*e.g.*, `Key_Generation`). Recall that these rules are used to define the protocol as a label transition system (LTS). The global state of this LTS is essentially a multiset of facts (*e.g.*, `In(senc(m,k))`), where is the initial state is the empty multiset. The protocol makes progress as follows: If the LTS is currently in $\mathcal{S}_i$ and all the facts of the premises of a rule are present/active in $\mathcal{S}_i$, then the rule will be executed. When a rule executes, the facts of the premises are usually removed from $\mathcal{S}_i$ and the facts of the conclusion (right-hand side) are added to $\mathcal{S}_i$, which in turn means the LTS transitioned from $\mathcal{S}_i$ to $\mathcal{S}_j$ (possibly, $\mathcal{S}_j \neq \mathcal{S}_i$). The action facts (the middle part) are usually used to label the transition, and hence they are not part of the global state. However, they appear on the trace of the model and thus play an important role in writing security properties.

Tamarin support two types of facts: *linear* and *persistent*. While linear facts are ephemeral (produced and deleted) during the execution of the rule (*e.g.*, `In(senc(m,k))`), persistent facts (prefixed with a bang `!`) are never removed from the global state (*e.g.*, `!Key(k)`). Three built-in facts in Tamarin are: `Fr` to denote a freshly generated name/identifier (*e.g.*, `~m`, where `~` indicates freshness of `m`), `In` and `Out` to exchange messages over an untrusted network.

Security Properties. The security properties in Tamarin are expressed as *lemma*. In the previous example, A wants to send B a message `m` such that only B can learn `m`. To achieve this objective, A encrypts `m` with a secret key `k` that only A and B are supposed to know. Now we can check whether `m` remains secret or not while in-transit through the untrusted network at the presence of an active adversary (K), by writing a lemma as shown in Figure 7. The lemma reads as follows: for all `m` and temporal instant `i`, once the rule `A_send(m)` is executed at `i`, there exists no subsequent temporal instant `j` (`i < j`, as per Tamarin notation) such that the adversary K gains the knowledge of `m` at `j`. In other words, once A sends the encrypted `m`, the adversary K can never know `m` (the plaintext).

For each of the specified security properties, Tamarin checks the property against the model. Tamarin explores all the paths of the model to find out a counterexample of the negated formula. If found, Tamarin presents the counterexample (*i.e.*, a trace), demonstrating how the model violates the property; otherwise, Tamarin ensures that the property holds.

3.2 Modeling ZigBee Modules in Tamarin

We use the Tamarin modeling language to model ZigBee. Before presenting our ZigBee model, we first outline our threat model and some cryptographic assumptions. Note that these assumptions are common in formal analysis of security protocols using symbolic models.

3.2.1 *Threat Model.*

The threat model that we will consider is Dolev-Yao(DY) model [15]. It is a built-in adversary model in Tamarin. The adversary has full control over the untrusted network such that it can delete, modify and intercept in-transit messages or even inject fabricated messages. It can also replay or combine messages that it learns from previous the network messages. However, the adversary will not have physical access to any device. In addition, we consider that the user is honest and always properly installs devices. The installed devices are certified by the ZigBee Alliance and never become rogue or compromised during their lifetime.

3.2.2 *Assumptions.*

We assume that the cryptographic primitives (*e.g.*, encryption, signature, hashing) are secure even at the presence of an adversary in the network. For example, an adversary can decrypt a message only if she has obtained the corresponding key. Without the key, she can never learn the message by using other techniques (*e.g.*, brute force, frequency analysis). If any party (*e.g.*, the coordinator) needs to generate some cryptographic keys, the party utilizes Tamarin's random number generator to produce fresh keys. Each random number generated by Tamarin is assumed to have the property of a cryptographic *nonce*.

3.2.3 *Modeling.*

We closely follow the ZigBee specifications [49, 50] to model ZigBee. In our model we also follow the traditional

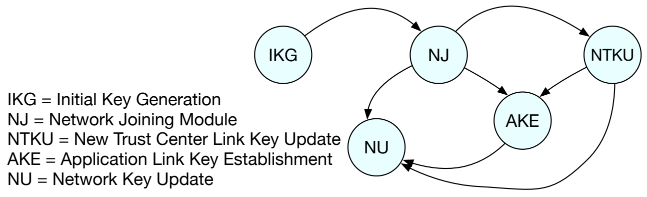
IKG
NJ
NTKU
AKE
NU
IKG = Initial Key Generation
NJ = Network Joining Module
NTKU = New Trust Center Link Key Update
AKE = Application Link Key Establishment
NU = Network Key Update

Figure 8: Relationship between different modules of ZigBee protocol

```
rule Net_key_gen:
[Fr(~nk)] --[SecretNK(~nk)]-> [!NwkKey($C,~nk)]
```

Figure 9: Network Key Generation Rule

naming convention. Since we aim to model different versions of ZigBee in a single symbolic model, we divide ZigBee security module into multiple sub-modules and chain them accordingly as appropriate for each version. To do so, we leverage the branching support available in Tamarin; as a result, Tamarin can decide at runtime which path of the model (*i.e.*, which version of ZigBee) it should follow depending on an incoming message. Although ours is a single model, we will now describe them individually as sub-modules to increase readability. For completeness, Figure 8 illustrates the relationship among these sub-modules.

Initial Key Generation (IKG). As a pre-requisite, we first need to generate some keys. We model all those key generations and their association with corresponding agent. For example, Figure 9 shows the rule of network key generation. In the premise part of the rule `Net_key_gen`, we generate a key using `Fr` which generates a fresh random number that we assume as the network key of the coordinator. In the conclusion part, the fact `[!NwkKey($C, ~nk)]` shows the association between the freshly generated key (`~nk`) with the coordinator, `$C`. The `$` sign is used to express a public agent. In addition, this fact is a persistent fact as denoted by `!` sign. The action fact `[SecretNK(~nk)]` is used to label the rule, which will appear on the trace and be used while writing security properties (as *lemma*) in Tamarin (§ 3.3).

Network Joining Module (NJ). To model each module, we rely on their state diagram that we created based on the specifications. The advantage of creating such state diagrams is that each transition in a state diagram directly corresponds to a rule in the symbolic model. Figure 10 shows the exchange of messages between a new end device and the coordinator when the end device wishes to join the network. It also shows how each agent makes transitions between their internal states. All ZigBee versions follow this primary module. The only difference is that the earlier versions use a *global key* as pre-configured link key and the later versions use a *unique key* that is only shared between the device and coordinator. In real life, for sharing the unique pre-configured key, the device uses an out-of-band channel (*e.g.*, the user enters the PIN/QR/bar code).

Modeling an out-of-band channel in Tamarin is challenging. To overcome this issue, we create an additional secure channel in Tamarin that mimics this out-of-band mechanism of sharing the

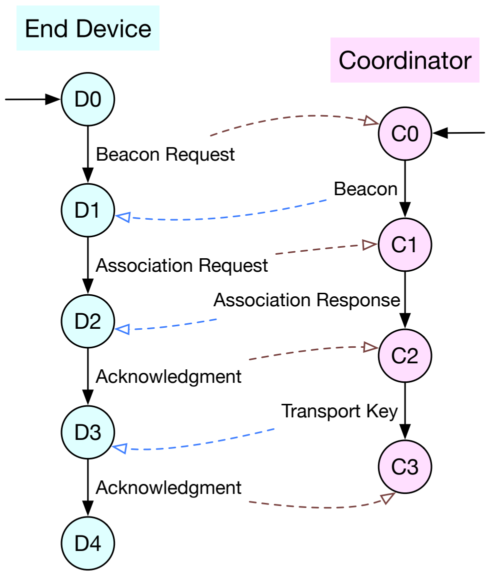

Figure 10: State diagram of Network Joining

```
rule ChanOut_S:
  [ Out_S($A,$B,x) ]
    --[ ChanOut_S($A,$B,x) ]->
      [ !Sec($A,$B,x) ]

rule ChanIn_S:
  [ !Sec($A,$B,x) ]
    --[ ChanIn_S($A,$B,x) ]->
      [ In_S($A,$B,x) ]
```

Figure 11: A Secure Channel

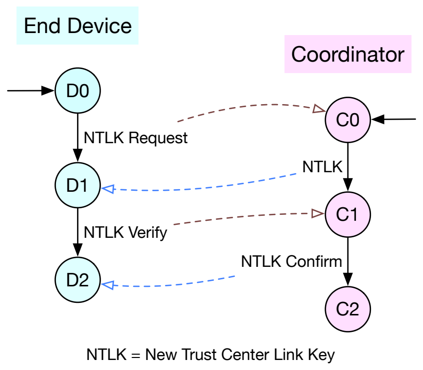

Figure 12: State diagram of New Trust Center Link Key Update

unique key. Since the objective of sharing the key using an out-of-band channel is to ensure that the key remains secret (*i.e.*, the adversary no knowledge of it), we can achieve the same objective by using this secure channel in Tamarin (see Figure 11).

New Trust Center Link Key Update (NTKU). The New Trust Center Link Key Update module (see Figure 12) is applicable only for ZigBee 3.0. The pre-configured link key is replaced by a new trust

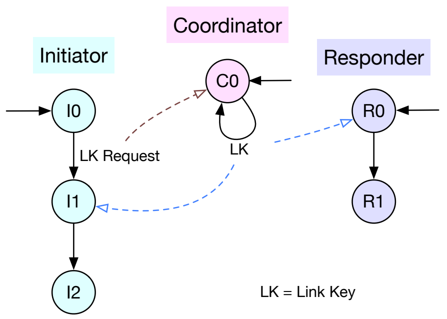

Figure 13: State diagram of Application Link Key Establishment

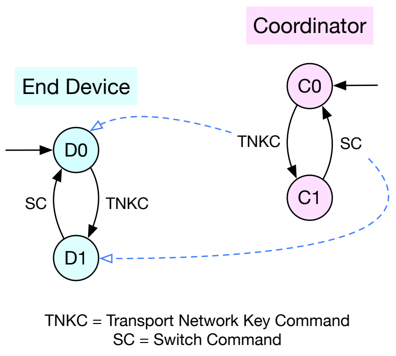

Figure 14: State diagram of Network Key Update

center link key. We model this module and utilizes the branching support in Tamarin so that it only applies to ZigBee 3.0.

Application Link Key Establishment (AKE). All ZigBee versions have this module (Figure 13), but it is optional. If two devices want to communicate secretly between themselves, then this module is applied to establish a unique key for these devices.

Network Key Update (NU). This module (Figure 14) is used to prevent replay attack. Each device has a frame counter. When the frame counter reaches a certain value, the coordinator must update the network key for all the devices. The coordinator transports the new network key using TNKC, and later it commands all the devices to switch to the new network key using SC. Tamarin allows us to include user-defined functions in the model. We wrote a user-defined function to model the increment of the frame counter.

3.3 ZigBee Security Properties in Tamarin

Previously in Section 2.3, we have discussed the security properties of ZigBee, and in this section, we will discuss our effort to model those properties (known as *lemmas* in Tamarin).

Secrecy of Keys. As we mentioned earlier in Section 2.3 that ZigBee mainly focuses on the secrecy of its keys. So we construct *lemmas* to prove whether the model maintains the secrecy of these keys or not. Here, secrecy means that a piece of information should only be known to the authorized parties. If any unauthorized party (*e.g.*, an adversary) learns a piece of information that is not intended for him/her, we consider it as a *violation* of secrecy.

```
lemma secrecy_NK:
"All x #i. SecretNK(x) @ i ==> (not(Ex #j. K(x) @j))"
```

Figure 15: Network Key Secrecy Lemma (simple)

```
    lemma secrecy_NK:
    "All x #i.
    SecretNK(x) @ i ==> (not(Ex #j. K(x) @j))
    | (Ex C D #r. RevPCK(D,C) @ r)"
```

Figure 16: Network Key Secrecy Lemma (Updated)

```
lemma injective_agreement:
  " All c d x #i.
    NwkKeyRecv(d,c,x) @ #i
    ==>
    (Ex #j. Send_Network_Key(c,d,x) @ #j
      &  j < i)
    |(Ex #r. RevNK(c) @ r)"
```

Figure 17: Injective-Agreement Lemma

In ZigBee all the keys (*e.g.*, network key, pre-configured link key, trust center link key, application link key) should only be known to the coordinator and the corresponding device(s). To handle a simple case of the secrecy of a key, we can write a lemma as shown in Figure 15. It says that once a key is generated, the adversary can never learn it in future. When Tamarin checks this lemma, it explores all possible execution paths to find out if the adversary can learn the key. If the adversary somehow learns the key, then Tamarin provides a trace of the model (aka, counterexample) demonstrating how the adversary learns the key and thus proving that the secrecy of the key does not hold. Otherwise, in case of no counterexample, Tamarin proves that the lemma holds.

However, this simpler lemma is not enough to accurately model the secrecy property of a key used in ZigBee. Since the keys are not necessarily independent of each other, the secrecy of one key may depend on the secrecy of another. Furthermore, in reality, it is not impossible that some keys are exposed to the adversary in the long run. Therefore our model needs to capture any underlying dependency. For instance, the network key is encrypted with the (unique/global) pre-configured link key and then transported by the coordinator. If the pre-configured link key is exposed to the adversary and the adversary manages to capture the encrypted packets of transporting the network key, then the adversary can easily obtain the network key by decrypting the traffic. We updated the secrecy lemma of the network key as shown in Figure 16. It says that once a network key is generated, an adversary cannot learn the key unless the pre-configured key has (somehow) been revealed. Similarly, we updated the secrecy lemma for the other keys.

Authentication properties. As we mentioned earlier in Section 2.3 about Lowe's authentication properties, we have tried to prove whether our ZigBee models satisfies those properties or not. We consider 4 lemmas to check all the 4 properties of authentication (*i.e.*, aliveness, weak agreement, non-injective agreement, injective agreement).

Table 1: Secrecy of Keys in ZigBee 1.0

Security Property	Result
Network key Secrecy	violated
Pre-Configured Link Key Secrecy	violated
Application Link Key Secrecy	violated

Table 2: Secrecy of Keys in ZigBee 3.0

Security Property	Result
Network key Secrecy	verified
Pre-Configured Link Key Secrecy	verified
New Trust Center Link Key Secrecy	verified
Application Link Key Secrecy	verified

Table 3: Results of Lowe's Authentication Properties

Security Property	Result
Aliveness	verified
Weak Agreement	verified
Non-injective agreement	verified
Injective agreement	verified

For instance, the proof of injective agreement of network key sharing, we construct a lemma as shown in Figure 17. It says that when a device receives a network key at time `i`, a coordinator must have sent the network key at time `j`, where `j < i`. In other words, if `x` is received by a device `d` at `i`, there must be a coordinator `c` that sends the network key `x` at `j` (`j < i`) or the key is previously revealed.

4 RESULTS FROM OUR ANALYSIS

In this section, we will discuss our findings on whether our ZigBee model satisfies the security properties or not. We invoke Tamarin to verify the security properties against our symbolic model for both ZigBee 1.0 and ZigBee 3.0.

Secrecy of keys. Recall that ZigBee 1.0 does not have the install-code based mechanism to generate and share a unique pre-configured link key; instead, it only uses a global key as the pre-configured link key. In the case of ZigBee 1.0, we incorporated a constant global key in our model. Our analysis (see Table 1) demonstrates that the secrecy of keys is violated because Tamarin is able to find a trace denoting how the adversary can learn each of these keys.

On the contrary, in ZigBee 3.0, the pre-configured link key is shared in a secured way. An install-code is shared using an out-of-band channel between the device and the coordinator. Then they generate the pre-configured key based on the shared code. So when we analyze the secrecy of keys in ZigBee 3.0, we have found that the properties hold (see Table 2).

Authentication. We checked the Lowe's authentication properties against our symbolic model. We found that all these properties hold (see Table 3), which means both ZigBee versions maintain all the authentication properties.

5 DISCUSSION

In our analysis, we have discovered the known vulnerabilities in ZigBee 1.0 and their absence in ZigBee 3.0, which demonstrates the protective measures taken by ZigBee 3.0 against those vulnerabilities are effective. In other words, ZigBee 3.0 maintains the confidentiality of the keys.

In the battle between usability and (effective) security, usability wins most of the time, if not always. While ZigBee 3.0 is a more secured option, it turns out that most of the manufacturer still depends on the earlier versions of ZigBee because of their simplicity compared to ZigBee 3.0. In our testbed experiments with smart-home IoT devices and coordinators (*e.g.*, Samsung SmartThings Hub (Ver. 3) [36], OpenHAB 2.4 [1] using Telegesis ETRX357USB ZigBee USB Stick), we observed that the devices and the hubs were still using the global key for transporting the network key. As a matter of fact, we were able to decrypt the whole communication simply by using the global key to extract the transported network key. Since the coordinators and devices did not use ZigBee 3.0, no new trust center link key was generated and/or shared between them. If an adversary had captured (or overheard) the encrypted traffic between these coordinators and the devices, it could have easily broken the confidentiality with little to no effort.

6 RELATED WORK

A significant amount of research has already been conducted about the security issues of ZigBee [16, 20, 27, 30, 39, 40, 42, 52], most of them focused on vulnerabilities at the implementation level.

Tobias Zillner *et al.* [52] described the actually applied security measures in ZigBee and highlighted some weaknesses in the implementation. They pointed out while the ZigBee specifications offer good measures, the ZigBee implementations lack those security measures. Their findings include that many ZigBee implementations still use the default TC link key for initial key exchange, never use link keys, and never update the network key even after a long time (11 months).

Joshua Wright has developed a Python-based framework named KillerBee [45] that can be used to exploit the security of ZigBee devices. KillerBee includes several tools designed to attack ZigBee and IEEE 802.15.4 networks. His experiments revealed that the network key was sent in plain text and there was no reply protection mechanism in ZigBee back then. The Zigbee specification that published in 2012 successfully addressed those problems and provided solutions.

Fan *et al.* [16] demonstrated how to sniff the network key of a ZigBee network using a combination of association flooding and replay attack. Vidgren *et al.* [40] showed two attacks named ZigBee End-Device Sabotage Attack and ZigBee Network Key Sniffing Attack. Olawumi *et al.* [30] presented three practical attacks against ZigBee security. Philipp *et al.* [27] analyzed the touchlink commissioning procedure of ZigBee and presented some novel attacks showing that this commissioning procedure is insecure by design. Their work has demonstrated how a passive eavesdropper can extract key material from a distance of 130 meters and an active attacker is able to take over devices from a distance of 190 meters.

Prior work [8, 13, 14] on formal analysis of security protocols using Tamarin motivated us to use Tamarin to conduct a formal security analysis of ZigBee. Tamarin has already been used as a powerful tool to analyze lots of protocols. Among them, Cremers *et al.* [14] did a comprehensive symbolic analysis of TLS 1.3, Basin *et al.* [8] performed a formal analysis of 5G authentication, and Cremers *et al.* [13] also made a formal analysis of DNP3.

7 CONCLUSION

To formally analyze security of ZigBee, we presented a symbolic model to represent both ZigBee 1.0 and ZigBee 3.0. The model closely follows the ZigBee specifications. We also derived some security properties from its specifications. Using a security protocol verification tool, Tamarin, we checked whether our model complies with these security properties. Our analysis demonstrates how ZigBee 1.0 violates the security properties by exposing the cryptographic keys to an unauthorized party (say, an adversary) and how ZigBee 3.0 maintains the security properties since those vulnerabilities were fixed in ZigBee 3.0.

REFERENCES

[1] [n. d.]. OpenHAB. https://www.openhab.org/.

[2] 2015. Vulnerability note VU#792004. CERT Vulnerability Notes Database. https://www.kb.cert.org/vuls/id/792004.

[3] 2019. 5 Key Developments in IoT for Transportation and Logistics. https://www.iotforall.com/real-time-tracking/.

[4] 2019. Double-Digit Growth Expected in the Smart Home Market, Says IDC. https://www.idc.com/getdoc.jsp?containerId=prUS44971219.

[5] 2019. Internet of Things (IoT) in Healthcare Market Size, Trends, Drivers, Restraints, Opportunities, and Challenges. https://www.ccsentinel.com/business/internet-of-things-iot-in-healthcare-market-size-trends-drivers-restraints-opportunities-and-challenges/.

[6] 2019. IoT in Intelligent Transportation Systems Market 2019 Size, Share, Trends, Growth, Types, Application, Industry Overview, Opportunity, Future Technology and Forecast to 2026. https://marketsgazette24.com/2019/11/11/iot-in-intelligent-transportation-systems-market-2019-size-share-trends-growth-types-application-industry-overview-opportunity-future-technology-and-forecast-to-2026/.

[7] Omar Alrawi, Chaz Lever, Manos Antonakakis, and Fabian Monrose. 2019. SoK: Security Evaluation of Home-Based IoT Deployments. In *2019 IEEE Symposium on Security and Privacy (S&P)*. IEEE.

[8] David Basin, Jannik Dreier, Lucca Hirschi, Saša Radomirovic, Ralf Sasse, and Vincent Stettler. 2018. A formal analysis of 5G authentication. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*. 1383–1396.

[9] Xianghui Cao, Devu Manikantan Shila, Yu Cheng, Zequ Yang, Yang Zhou, and Jiming Chen. 2016. Ghost-in-zigbee: Energy depletion attack on zigbee-based wireless networks. *IEEE Internet of Things Journal* 3, 5 (2016), 816–829.

[10] Iliano Cervesato. 2001. Typed Multiset Rewriting Specifications of Security Protocols. *Electronic Notes in Theoretical Computer Science* 40 (2001).

[11] E. Chiel. 2016. Here are the sites you can't access because someone took the internet down. https://splinternews.com/here-are-the-sites-you-cant-access-because-someone-took-1793863079.

[12] Catalin Cimpanu. 2019. Microsoft: Russian state hackers are using IoT devices to breach enterprise networks. https://www.zdnet.com/article/microsoft-russian-state-hackers-are-using-iot-devices-to-breach-enterprise-networks/.

[13] Cas Cremers, Martin Dehnel-Wild, and Kevin Milner. 2017. Secure Authentication in the Grid: A Formal Analysis of DNP3: SAv5. In *Computer Security – ESORICS 2017*, Simon N. Foley, Dieter Gollmann, and Einar Snekkenes (Eds.). Springer International Publishing, Cham, 389–407.

[14] Cas Cremers, Marko Horvat, Jonathan Hoyland, Sam Scott, and Thyla van der Merwe. 2017. A comprehensive symbolic analysis of TLS 1.3. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*. 1773–1788.

[15] Danny Dolev and Andrew Yao. 1983. On the security of public key protocols. *IEEE Transactions on information theory* 29, 2 (1983), 198–208.

[16] Xueqi Fan, Fransisca Susan, William Long, and Shangyan Li. 2017. *Security analysis of zigbee*. Technical Report. Massachusetts Institute of Technology.

[17] Sarthak Grover and Nick Feamster. 2016. The internet of unpatched things. In *Proc. PrivacyCon*.

[18] Isobel Hamilton. 2019. Siri, Google Assistant, and Amazon Alexa can be hijacked with a $14 laser pointer to open garage doors, start cars, and shop online. https://www.businessinsider.com/hackers-hijack-smart-speakers-apple-google-facebook-amazon-14-laser-2019-11.

[19] Kevin Harris. 2019. IoT Technology Is Changing Healthcare, But Not Without Risk. https://inpublicsafety.com/2019/10/iot-technology-is-changing-healthcare-but-not-without-risk/.

[20] Salam Khanji, Farkhund Iqbal, and Patrick Hung. 2019. ZigBee Security Vulnerabilities: Exploration and Evaluating. In *2019 10th International Conference on Information and Communication Systems (ICICS)*. IEEE, 52–57.

[21] Heather Landi. 2019. 82% of healthcare organizations have experienced an IoT-focused cyberattack, survey finds. https://www.fiercehealthcare.com/tech/82-healthcare-organizations-have-experienced-iot-focused-cyber-attack-survey-finds.

[22] Gavin Lowe. 1997. A hierarchy of authentication specifications. In *Proceedings 10th Computer Security Foundations Workshop*. IEEE, 31–43.

[23] Knud Lasse Lueth. August 8, 2018. State of the IoT 2018: Number of IoT devices now at 7B - Market accelerating. https://iot-analytics.com/state-of-the-iot-update-q1-q2-2018-number-of-iot-devices-now-7b/.

[24] Marianne Kolbasuk McGee. 2019. FDA Issues Alert on Medical Device IPnet Vulnerabilities. https://www.careersinfosecurity.com/fda-issues-alert-on-medical-device-ipnet-vulnerabilities-a-13164.

[25] Brian McHale. 2019. Asset tracking that works out of the box from AWS IoT, Verizon, and Domo. https://www.verizon.com/about/news/aws-iot-verizon-and-domo.

[26] Francesca Meneghello, Matteo Calore, Daniel Zucchetto, Michele Polese, and Andrea Zanella. 2019. Iot: Internet of threats? a survey of practical security vulnerabilities in real iot devices. *IEEE Internet of Things Journal* 6, 5 (2019), 8182–8201.

[27] Philipp Morgner, Stephan Mattejat, Zinaida Benenson, Christian Müller, and Frederik Armknecht. 2017. Insecure to the touch: attacking ZigBee 3.0 via touchlink commissioning. In *Proceedings of the 10th ACM Conference on Security and Privacy in Wireless and Mobile Networks*. ACM, 230–240.

[28] NXP: ZigBee 3.0 Devices User Guide 2016. https://www.nxp.com/docs/en/user-guide/JN-UG-3114.pdf

[29] Olayemi Olawumi, Keijo Haataja, Mikko Asikainen, Niko Vidgren, and Pekka Toivanen. 2014. Three practical attacks against ZigBee security: Attack scenario definitions, practical experiments, countermeasures, and lessons learned. In *2014 14th International Conference on Hybrid Intelligent Systems*. IEEE, 199–206.

[30] Olayemi Olawumi, Keijo Haataja, Mikko Asikainen, Niko Vidgren, and Pekka Toivanen. 2014. Three practical attacks against ZigBee security: Attack scenario definitions, practical experiments, countermeasures, and lessons learned. In *2014 14th International Conference on Hybrid Intelligent Systems*. IEEE, 199–206.

[31] Danny Palmer. 2019. This aggressive IoT malware is forcing Wi-Fi routers to join its botnet army. https://www.zdnet.com/article/this-aggressive-iot-malware-is-forcing-wi-fi-routers-to-join-its-botnet-army/.

[32] Jon Porter. 2019. Security researchers expose new Alexa and Google Home vulnerability. https://www.theverge.com/2019/10/21/20924886/alexa-google-home-security-vulnerability-srlabs-phishing-eavesdropping.

[33] Jingjing Ren, Daniel J. Dubois, David Choffnes, Anna Maria Mandalari, Roman Kolcun, and Hamed Haddadi. 2019. Information Exposure for Consumer IoT Devices: A Multidimensional, Network-Informed Measurement Approach. In *Proc. of the Internet Measurement Conference (IMC)*.

[34] Eyal Ronen and Adi Shamir. 2016. Extended functionality attacks on IoT devices: The case of smart lights. In *2016 IEEE European Symposium on Security and Privacy (EuroS&P)*. IEEE, 3–12.

[35] 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.

[36] Samsung SmartThings Hub V3 [n. d.]. Retrieved June 30, 2019 from https://www.samsung.com/us/smart-home/smartthings/hubs/samsung-smartthings-hub--2018--gp-u999sjvlgda/

[37] Benedikt Schmidt, Simon Meier, Cas Cremers, and David Basin. 2012. Automated analysis of Diffie-Hellman protocols and advanced security properties. In *2012 IEEE 25th Computer Security Foundations Symposium*. IEEE, 78–94.

[38] Tamarin Prover [n. d.]. Github Repository. https://tamarin-prover.github.io/

[39] Ivan Vaccari, Enrico Cambiaso, and Maurizio Aiello. 2017. Remotely Exploiting AT Command Attacks on ZigBee Networks. *Security and Communication Networks* 2017 (2017).

[40] Niko Vidgren, Keijo Haataja, Jose Luis Patino-Andres, Juan Jose Ramirez-Sanchis, and Pekka Toivanen. 2013. Security threats in ZigBee-enabled systems: vulnerability evaluation, practical experiments, countermeasures, and lessons learned. In *2013 46th Hawaii International Conference on System Sciences*. IEEE, 5132–5138.

[41] What's New in Zigbee 3.0 2019. http://www.ti.com/lit/an/swra615a/swra615a.pdf

[42] Lindsey N Whitehurst, Todd R Andel, and J Todd McDonald. 2014. Exploring security in ZigBee networks. In *Proceedings of the 9th Annual Cyber and Information Security Research Conference*. ACM, 25–28.

[43] Zack Whittaker. 2019. Amazon Ring doorbells exposed home Wi-Fi passwords to hackers. https://techcrunch.com/2019/11/07/amazon-ring-doorbells-wifi-hackers/.
[44] Zack Whittaker. 2019. Security flaws in a popular smart home hub let hackers unlock front doors. https://techcrunch.com/2019/07/02/smart-home-hub-flaws-unlock-doors/.
[45] Joshua Wright. 2009. Killerbee: practical zigbee exploitation framework. In *11th ToorCon conference, San Diego*, Vol. 67.
[46] Zigbee Alliance 2018. Analysts Confirm Half a Billion Zigbee Chipsets Sold, Igniting IoT Innovation; Figures to Reach 3.8 Billion by 2023. https://www.marketwatch.com/press-release/analysts-confirm-half-a-billion-zigbee-chipsets-sold-igniting-iot-innovation-figures-to-reach-38-billion-by-2023-2018-08-07.
[47] Zigbee Base Device Behavior Specification Version 1.0 2016. http://www.zigbee.org/wp-content/uploads/2014/10/docs-13-0402-13-00zi-Base-Device-Behavior-Specification-2.pdf
[48] Zigbee Security Basics 2017. https://research.kudelskisecurity.com/2017/11/08/zigbee-security-basics-part-2/
[49] ZigBee Specifications 2012. http://www.zigbee.org/wp-content/uploads/2014/11/docs-05-3474-20-0csg-zigbee-specification.pdf
[50] ZigBee Specifications 2015. https://zigbeealliance.org/wp-content/uploads/2019/11/docs-05-3474-21-0csg-zigbee-specification.pdf
[51] ZigBee Tamarin Modeling [n. d.]. Github Repository. https://github.com/Li-Syr/zigbee-tamarin/
[52] Tobias Zillner. 2015. ZIGBEE EXPLOITED: The good, the bad and the ugly. https://www.blackhat.com/docs/us-15/materials/us-15-Zillner-ZigBee-Exploited-The-Good-The-Bad-And-The-Ugly-wp.pdf

A DETAILS ON HOW AN NEW IOT DEVICE JOINS A ZIGBEE NETWORK

In the paper, we analyzed the ZigBee protocol based on the specifications. Here we provide a more detailed diagram (Figure A1) based on the real communications captured by the Wireshark software. The coordinator is the Samsung SmartThings Hub and the end device is a Samsung SmartThings Outlet.

A.0.1 Network Discovery. Assume we have a new end device and we want to join this device into a ZigBee network. The very first thing for the end device to do is to detect all coordinators within its transmission range, and that is to discover existing ZigBee networks around it. As shown in Figure A1, the end device first broadcasts **beacon request** command and receives **beacon** as a reply. If the beacon frame with its *Association Permit* field set to *True*, the end device will initiate the association process. In addition, when the coordinator permits for other devices to join, it will also broadcast **permit joining request** to all the coordinators and routers in the current PAN (note that this command is encrypted with the network key) to enable association in the whole PAN.

A.0.2 Join into the PAN. Now the end device knows how many coordinators are around it and which one allows new devices to join its network. The next step for the end device is to try to associate with the coordinator by sending **association request** to the coordinator it wants to join. It will finally receive **association response** with *Association Status* field enclosed to indicate whether the association attempt was successful. However, one can notice that there is a **data request** command lies between the association request and response. This is because the association decision is designed to be available at the coordinator within a time of *aResponseWaitTime*. After this time, the end device attempts to forcefully pull the **association response** from the coordinator using the **data request** command.

A.0.3 Setup Security. After a device joined into a PAN, the security of this PAN is achieved by encrypting commands at Network (NWK)

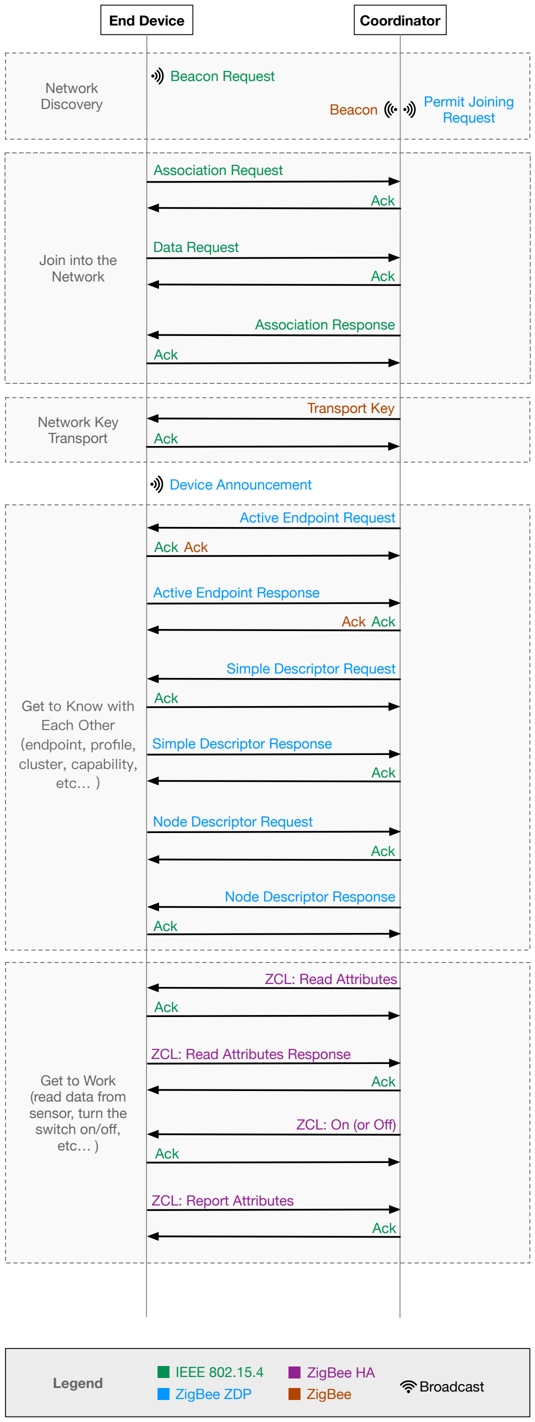

Figure A1: Exchanged network messages between a new end device and a ZigBee coordinator

layer or Application Support (APS) layer within this PAN. This will prevent eavesdropping as long as the encryption key remains secret. The (symmetric) encrypting key used in the network is the so-called *network key*. The coordinator generates this key and sends to newly joined devices over the **transport key** command. One would argue that sending a key via an untrusted communication channel is pointless since this would leak the key; and that is correct. The common solution is to use asymmetric key encryption scheme to transport the symmetric key. Due to the complexity concern, the ZigBee protocol uses a naive approach instead of public-key encryption. The **transport key** command is sent over the APS layer, and this layer is encrypted by a pre-configured trust center link key.

A.0.4 Device Announcement. Just like the IP address of a host (*i.e.*, computer), after the ZigBee device joined to a network, it will be assigned a network address (Nwk Addr). The **device announcement** command is broadcasted by the newly joined device to see if any other devices already use this address. If so, a new address will be assigned.

A.0.5 Get to Know with Each Other. Until now, the coordinator does not know the services supported by this device, as well as whether the device is a light bulb or a motion sensor.

Intuitively, if the device is a light bulb or a switch, the coordinator can send a message to turn it on or off. In ZigBee, this On/Off functionality is defined by *cluster*, be more specific, the *OnOff Cluster*. Clusters such as *Scenes Cluster*, *Power Configuration Cluster*, and *Temperature Configuration Cluster* are common clusters used in smart home applications. On the other hand, ZigBee is designed to be general, not only for smart home devices. ZigBee uses *profile* as domain space to define related applications and devices, *e.g.*, *Home Automation* (HA) is a public profile about lights, outlets, switches, and thermostats.

The first command in this section is the **active endpoint request**. Literally, this command is to query active endpoints of the end device, but what is an endpoint? We know from the previous discussion that if a coordinator wants to control the end device, it has to know the application profile of the device and what are the clusters it supports in this profile. However, a physical device may have several independent things connected to it or support multiple irrelevant services; *e.g.*, a power outlet may have five sockets and a motion sensor may support lumen sensing and temperature sensing. As one can think of, every socket of an outlet is an endpoint, every sensing functionality of a motion sensor is also an endpoint. This design makes different devices and application profiles can exist within one physical device (or a *node*). After the coordinator knows the IDs of each endpoint of the end device, it will continue to query the *profile* of each endpoint as well as *clusters* in the profile by **simple descriptor request**.

The end device also wants to know more information about the coordinator, such as what radio-frequency bands it supports, what is the max buffer size. This can be done by the **node descriptor request** and **response**.

Note that there are two different **Ack**s for **active endpoint request** and **response** – one for IEEE 802.15.4 protocol and the other for ZigBee Application Support (APS) layer. The APS layer ACK is a more sophisticated retry and acknowledgment mechanism. The APS layer ACK in Figure A1 is just an example; those ACK commands happen widely in the communication, and the usages differ from manufacturer's implementations.

A.0.6 Get to Work. To make the end device to provide actual service is easy. So far the coordinator already learns the endpoints of the end device (node), the profile of a specific endpoint, and clusters supported in this profile. Now the coordinator can control a light or get the temperature just by sending a specific command defined in a specific cluster of a profile to the destination endpoint. These commands are all enclosed by ZigBee Cluster Library (ZCL) frames.