Formal Security Analysis of ZigBee Protocol

Sebastiano Morson

September 4, 2025

Contents

1 Introduction

The Internet of Things (IoT) ecosystem consists of a wide variety of devices with different capabilities and needs. To enable communication among them, communication protocols are required that take into account the heterogeneity of the network, both in terms of the resources demanded by the network nodes and their placement. Among these protocols, Zigbee stands out for its energy efficiency and its ability to support secure and scalable mesh networks. However, the growing adoption of Zigbee has necessitated a thorough analysis of its vulnerabilities and the security properties it must satisfy [9]. Formal verification serves as an essential tool to ensure the security of protocols, allowing the identification of potential flaws before they can be exploited by malicious actors.

In this work, I examine the security of the Zigbee protocol using Tamarin-Prover, a tool for the formal verification of cryptographic protocols [4]. Tamarin-Prover enables the modeling of the protocol's behavior and the verification of fundamental security properties, such as confidentiality and authentication. Through this analysis, the goal is to assess whether Zigbee meets the required security standards.

The document is structured into several sections. First, I provide an overview of the Zigbee protocol, describing its stack architecture, the types of supported devices, network topologies, and cryptographic key management. Next, I analyze the main security threats to the protocol and the properties it must guarantee. I then introduce Tamarin Prover, explaining how protocols are formalized within the tool. Subsequently, I present the modeling of the Zigbee protocol in Tamarin. Following that, I formalize the security properties and describe the rules used for their verification. Finally, I discuss the results of the analysis, highlighting any vulnerabilities that were identified.

2 ZigBee protocol overview

The Zigbee stack architecture is based on distinct layers that handle specific functionalities [2]. Figure 1 shows the scheme of the various layers. In this context, we will focus on the Network and Application layers, leaving out the details of the physical layers (IEEE 802.15.4).

Network Layer (NWK): This layer is responsible for network formation, management and maintenance. It handles device addressing, packet routing and implementation of network-level security policies. The NWK layer enables the creation of network topologies such as star, tree and mesh.

Application Layer (APL): The Application layer includes several key components:

1. **Application Support Sublayer (APS):** Provides services for managing communications between applications, including handling binding tables that determine which devices can communicate with each other.

2. **Zigbee Device Object (ZDO):** Manages device functionalities, such as service discovery and management, and coordinates device association and disassociation operations in the network.

3. **Application Framework:** Allows developers to define application profiles and specific clusters for application needs, facilitating interoperability between devices from different manufacturers.

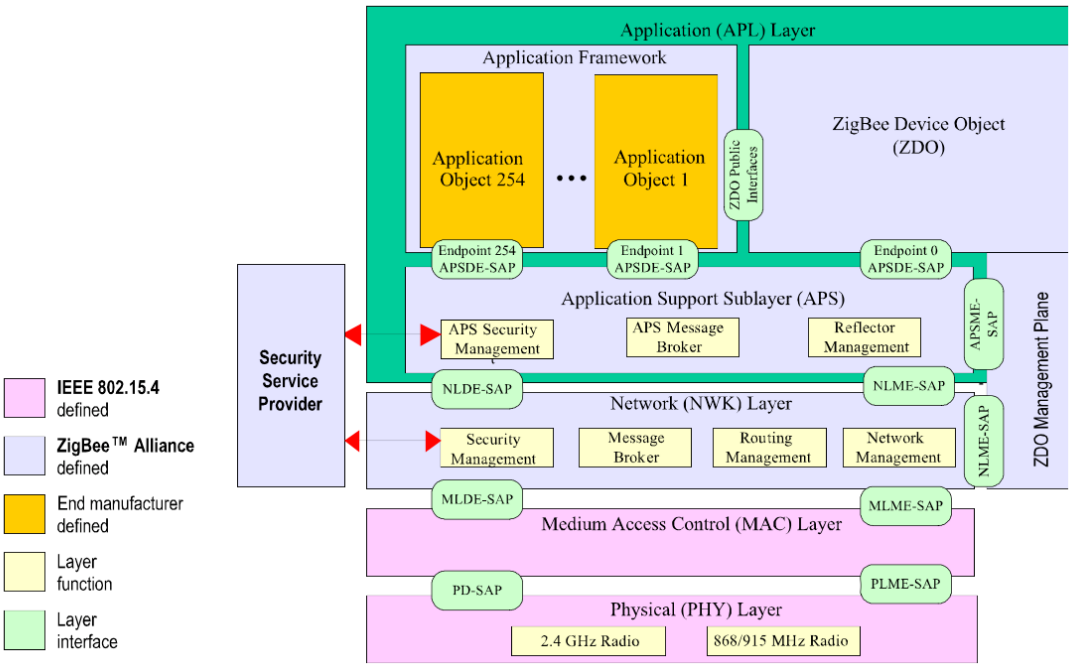

Figure 1: ZigBee stack architecture

Comparison with Bluetooth and Wi-Fi

ZigBee, Bluetooth, and Wi-Fi are three wireless communication protocols designed for different purposes, and their effectiveness heavily depends on the use case. ZigBee stands out primarily for its extremely low power consumption, making it ideal for devices that must operate for long periods without battery replacement, such as environmental sensors, home automation systems, and IoT devices in general.

Unlike Bluetooth, which consumes more power (about 60 mA in transmission), ZigBee uses only 25–35 mA during transmission and as little as 3 µA in standby mode. This is possible thanks to highly efficient power management and a lower transmission speed (250 Kbps compared to Bluetooth's 1 Mbps and Wi-Fi's 11 Mbps), which is sufficient for sending small data packets at regular intervals.

Wi-Fi, on the other hand, is designed for high-speed, high-capacity transmissions (e.g., video streaming, web browsing), but this comes with high power consumption (up to 400 mA in transmission) and greater protocol stack complexity (up to 1 MB compared to ZigBee's 32–250 KB). As a result, Wi-Fi is poorly suited for battery-powered devices or dense, distributed networks of small sensors.

From a scalability perspective, ZigBee offers superior performance: it can support up to 64,000 nodes per network, thanks to its mesh topology support. In comparison, classic Bluetooth is limited to 7 active devices per piconet, and Wi-Fi can handle about 32 devices per access point.

Finally, ZigBee is designed for very fast association times (typically 30 ms), which is crucial for devices that activate only in response to events. Bluetooth and Wi-Fi, however, require longer connection times (Bluetooth >3 sec; Wi-Fi 1 sec). See Table 2.

2.1 Zigbee Device Types:

In Zigbee networks, there are three main types of devices, each with specific roles and responsibilities:

1. **Zigbee Coordinator (ZC):** It is the main device in the Zigbee network, responsible for starting and managing the network. The coordinator assigns addresses to devices joining the network and maintains information about the topology and status of devices. In a (non-distributed) Zigbee network, there can only be one coordinator.

Wireless Parameter	Bluetooth	Wi-Fi	ZigBee
Frequency band	2.4 GHz	2.4 GHz	2.4 GHz
Physical/MAC layers	IEEE 802.15.1	IEEE 802.11b	IEEE 802.15.4
Range	9 m	75 to 90 m	Indoors: up to 30 m Outdoors (line of sight): up to 100 m
Current consumption	60 mA (Tx mode)	400 mA (Tx mode) 20 mA (Standby mode)	25–35 mA (Tx mode) 3 μA (Standby mode)
Raw data rate	1 Mbps	11 Mbps	250 Kbps
Protocol stack size	250 KB	1 MB	32 KB 4 KB (for limited function end devices)
Typical network join time	>3 sec	Variable, typically 1 sec	Typically 30 ms
Maximum number of nodes per network	7	32 per access point	64 K

Table 1: Comparison of Wireless Standards: Bluetooth, Wi-Fi, and ZigBee

2. **Zigbee Router (ZR):** These devices extend network coverage by acting as intermediaries in packet routing. Routers can communicate with the coordinator, with other routers and with end devices, facilitating communication in large networks or in environments with physical obstacles.

3. **Zigbee End Device (ZED):** These are limited-functionality devices that communicate exclusively with a coordinator or router. They do not participate in packet routing and are often designed to operate with minimal power consumption, making them ideal for battery-powered applications.

2.2 Supported Network Topologies:

Zigbee supports various network topologies, adapting to different application needs:

- **Star Topology:** All end devices communicate directly with the coordinator. This structure is simple but limits network coverage.
- **Tree Topology:** The coordinator acts as a root node, with routers and end devices organized in a hierarchical structure. This topology facilitates organized network expansion.
- **Mesh Topology:** Each device can communicate with other devices within range, allowing multiple routing paths and increasing network resilience. The mesh topology is particularly advantageous in environments where communication reliability is crucial.

2.3 Example of a Real Zigbee Network:

Considering a smart home, it's likely to contain a Zigbee network organized as follows:

- Zigbee Coordinator (ZC): A central hub, such as Amazon Echo Plus or SmartThings Hub, that initiates and manages the Zigbee network.

- Zigbee Router (ZR): Smart bulbs distributed in various rooms that, in addition to providing lighting, extend network coverage by facilitating communication between distant devices.
- Zigbee End Device (ZED): Battery-powered temperature and humidity sensors that monitor environmental conditions and periodically send data to the coordinator via routers.

In this configuration, the central hub (ZC) manages the network and coordinates communications. The smart bulbs (ZR) not only receive commands from the hub but also act as intermediaries for the sensors (ZED), ensuring sensor data reaches the hub even when they're out of the coordinator's direct range. This example illustrates how different device types collaborate to create an efficient and reliable Zigbee network.

2.4 Cryptographic Keys and Session Management Overview

The ZigBee protocol adopts a security model based on 128-bit symmetric keys, designed to ensure data confidentiality, integrity, and authentication. This model, introduced with ZigBee 1.0 and maintained in ZigBee 3.0, operates at two main levels: network security (NWK) and application security (APS). Three main key types are used: network key, Trust Center link key, and application link key, each with specific functions and management methods [2][1][3].

2.4.1 Network Key

The network key is used to encrypt and authenticate messages at the network level. All devices authorized to join the network share this key, which is periodically updated by the Trust Center to mitigate compromise risks. Each update is identified by a sequence number (from 0 to 255), allowing devices to recognize the correct key version to use.

2.4.2 Trust Center Link Key

This key is used for secure end-to-end communications between a device and the Trust Center. Its main functions include:

1. Encrypting the initial transfer of the network key to a joining device.
2. Managing network key updates for reconnecting devices.
3. Protecting APS security messages between routers and the Trust Center.

The Trust Center can manage these keys centrally, using unique keys per device, keys derived from shared data (such as IEEE addresses [11]), or a global key common to all devices.

2.4.3 Application Link Keys

In addition to the Trust Center link key, devices can establish application link keys for secure communications between themselves, without involving the Trust Center. These keys can be manually configured by the application or generated by the Trust Center upon request. In the latter case, the Trust Center generates a random key and sends it to both devices involved in the communication. This mechanism adds an additional layer of security for direct communications between devices.

3 Threats and Security Properties of Zigbee

In this section we analyze the main threats that could compromise Zigbee communication security and the security properties that must be verified to ensure safe protocol operation.

3.1 Common Threats in Zigbee Protocols

The following threats represent some of the main security risks for Zigbee:

- **Eavesdropping**: Since Zigbee uses wireless communication, an attacker could intercept transmitted messages to obtain sensitive information, such as encryption keys or control data.
- **Replay Attack**: An attacker could record a valid message and retransmit it later to deceive network devices, performing unauthorized operations.
- **Key Compromise**: If an attacker manages to compromise the cryptographic keys used in the protocol, they could decrypt communications, impersonate legitimate devices, or modify transmitted messages.

3.2 Expected Security Properties

To mitigate the previously described threats, a secure Zigbee protocol must guarantee the following fundamental security properties:

3.2.1 Authentication

Authentication ensures that communicating devices are indeed those they claim to be, preventing impersonation attacks. We use Lowe's model [7] to formalize authentication properties:

1. **Aliveness**: An entity must be certain that the other party has participated in the communication session.
2. **Weak Agreement**: The two involved entities agree they have communicated with each other, though not necessarily on the exact communication details.
3. **Non-Injective Agreement**: The two entities agree on the communication and session identifiers, though duplicate sessions may occur.
4. **Injective Agreement**: Similar to non-injective agreement, but with the guarantee that each session is unique and not reused.

3.2.2 Confidentiality

Confidentiality guarantees that transmitted information remains inaccessible to unauthorized entities. To achieve this property, Zigbee adopts the **AES-CCM*** cryptographic suite, a modified variant of the *CCM* (Counter with CBC-MAC) mode applied to AES-128. The CCM* mode integrates into a single cryptographic procedure data encryption, authentication, and integrity verification, while also supporting "encryption-only" or "authentication-only" operations (Figure 2).

During transmission, the plaintext message is divided into 128-bit blocks and processed by AES-CCM*, which produces the ciphertext accompanied by a **Message Integrity Code (MIC)**. The recipient, in turn, uses the same procedure to decrypt the data and generates an independent MIC to compare with the received one. If the two values match, the message is considered authentic and intact.

The process uses a **nonce** of 13 bytes constructed from the security control field, the frame counter, and the source address in the auxiliary header. The MIC can be 32, 64, or 128 bits long, and provides stronger authenticity guarantees than simpler mechanisms like CRC.

Since AES-CCM* is formally recognized as a secure scheme for authenticated symmetric encryption, this discussion assumes its robustness is not a critical weakness. Our analysis therefore focuses on proper key management and preventing nonce reuse attacks.

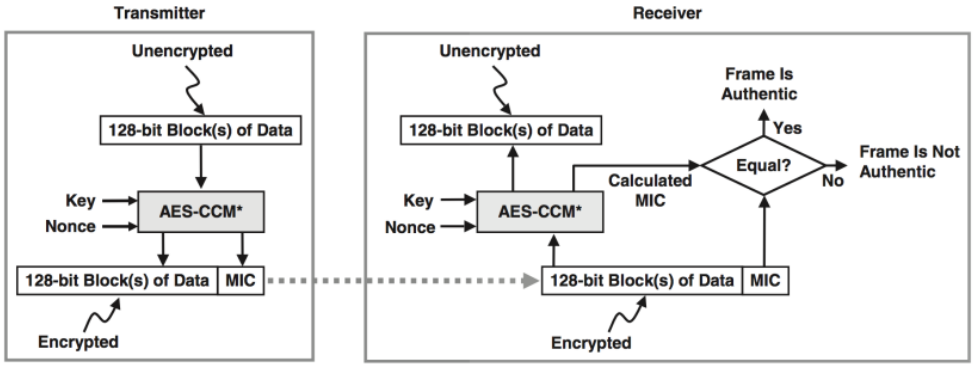

Figure 2: AES-CCM*

3.2.3 Integrity

Integrity guarantees that received messages have not been altered during transmission. Zigbee ensures this property through two complementary mechanisms:

- **MIC (Message Integrity Code)**: A cryptographic code calculated by the sender and verified by the recipient, designed to detect intentional or accidental message modifications. In Zigbee, the MIC is produced as an integral part of the AES-CCM* operation, ensuring consistency between encryption and integrity verification.
- **Frame Counter**: An incremental counter included in encrypted messages, used to prevent replay attacks and guarantee transmission order and uniqueness. The frame counter also contributes to nonce generation, and its reuse can compromise the security of the encrypted channel.

The MIC generated via CCM* provides more advanced protection than traditional checksums, as it resists both unintentional errors and malicious data tampering.

3.3 Specifications to Verify

Among the various protocols that make up Zigbee, the paper by Li, Podder and Hoque [6] focuses on the formal verification of four in particular.

3.3.1 Network Key Sharing

When a device requests network access and is authorized, it receives a network key from the coordinator. The transmission of the network key from trust center to end-device occurs by encrypting the network key with the trust center link key. Two methods are defined to generate this key:

- **pre-configured global key**: This was the solution adopted by Zigbee 1.0. It is based on using a global key shared between device and coordinator (5A 69 67 42 65 65 41 6C 6C 69 61 6E 63 65 30 39). Obviously if the attacker discovers the global key they can obtain the network key.
- **install code**: In ZigBee 3.0, the use of keys derived from an installation code was introduced. The installation code, consisting of 6, 8, 12 or 16 bytes plus a 16-bit CRC, is shared between trust center and device and is processed by both parties through an AES-MMO hash function to generate a preconfigured link key. The install code, together with the device's EUI64 address, must be communicated to the Trust Center out-of-band (for example, via phone or Internet) to allow secure authentication during network joining.

  Figure 3 shows a diagram of how the pre-configured key is derived from the install-code by both involved parties.

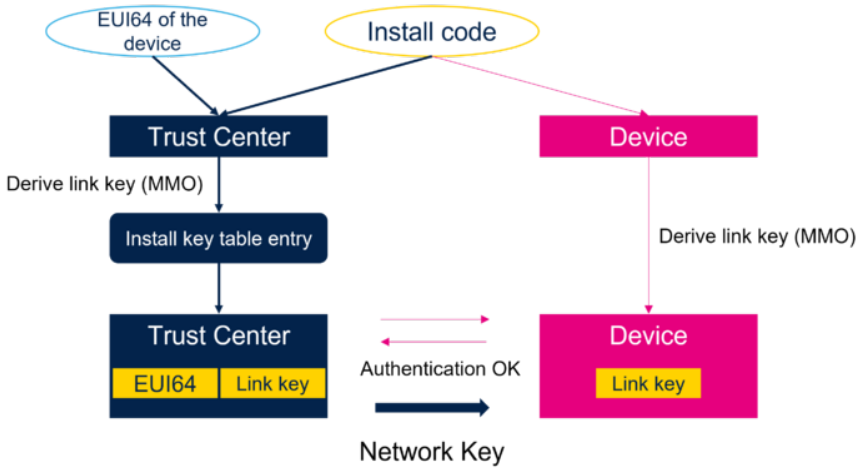

Figure 3: Installation code generation

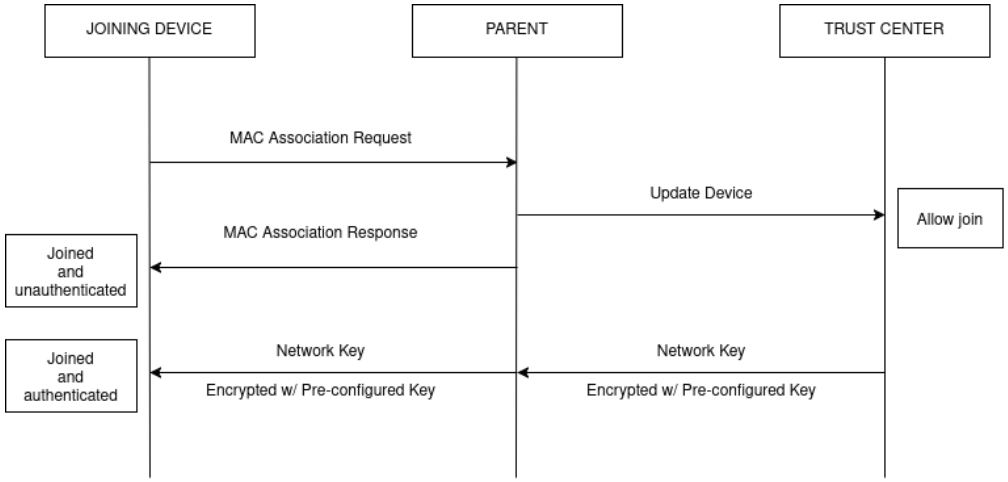

Figure 4: Joining a secure network

3.3.2 Joining a secure network

To join a network, a device must receive approval from the parent through a beacon request. If accepted, the device transitions to the joined state. Upon receiving the MAC association request, it notifies the trust center of the device's request. At this point, the trust center may request additional information from the device before completing authentication. If the trust center recognizes the device, it authenticates the device and sends the Network Key encrypted with the Trust Center Link Key. Figure 4 shows the joining scheme.

After authentication, Zigbee 3.0 specifications require the device to request an update of the Trust Center Link Key. Figure 5 shows the behavior adopted by both parties during this phase.

3.3.3 Application Key Establishment

To establish secure end-to-end communication between two devices, Zigbee provides for the use of Application Keys to encrypt communications. Application Keys are unique for each pair of devices. The model used to establish the Application Key is as follows:

1. An initiator sends a Request Key to the coordinator with Key Type "Application Link Key"
2. The coordinator responds to the initiator with the Transport Key command containing the Application

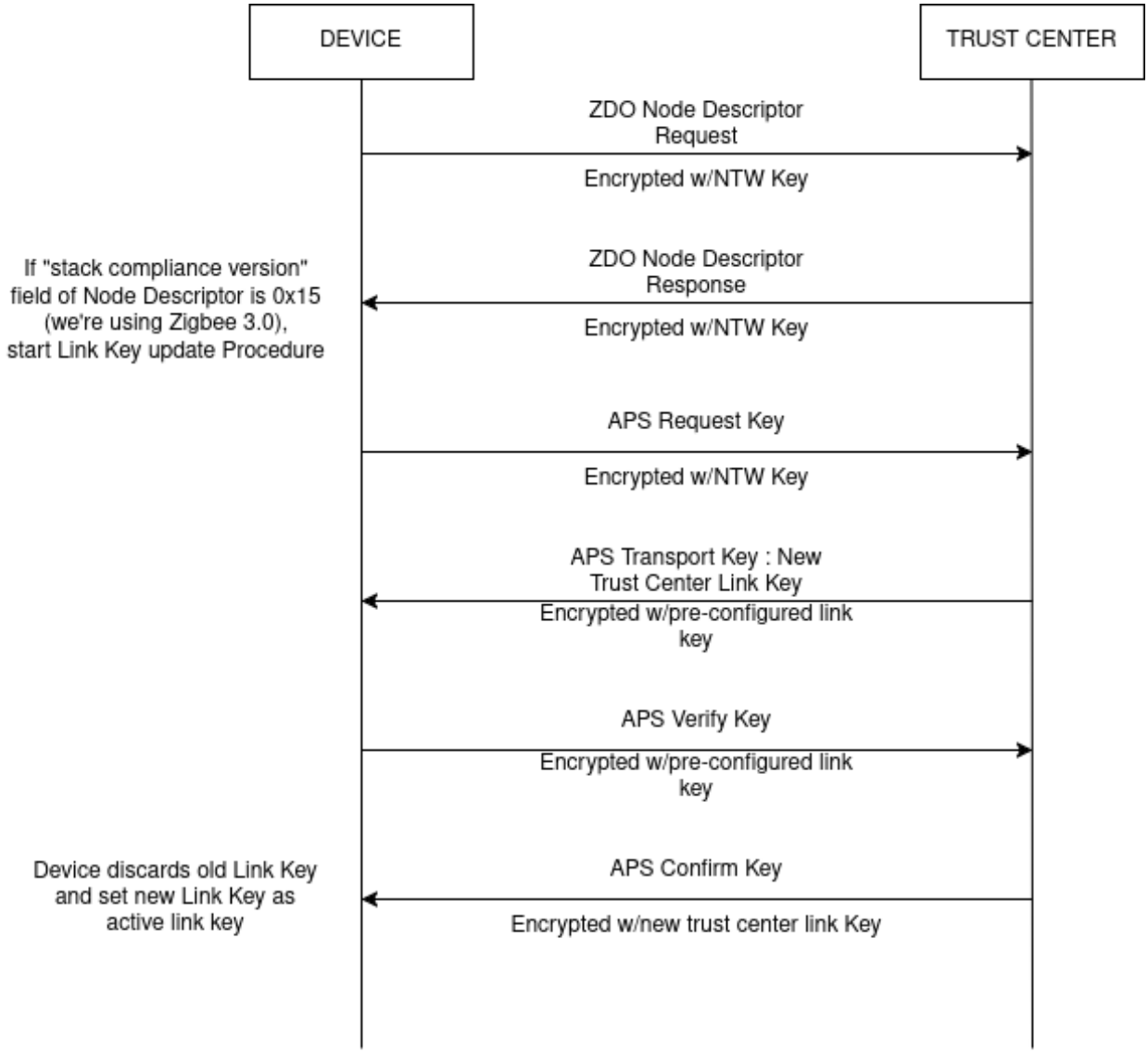

Figure 5: Updating Trust Center Link Key

Key

3. The coordinator shares the Application Key with the second device

3.3.4 Network Key Update

To prevent replay attacks, each Network Layer packet contains a Frame Counter in the Auxiliary Header. This frame counter has a maximum value of 0xFFFFFFFF, after which it must be reset before new messages can be sent. To be reset, a Network Key update must occur and the NwK frame counter value must be greater than 0x80000000.

The Trust Center can distribute the new network key in two ways:

- Broadcast: the new key is sent to all devices, encrypted with the previous network key.
- Unicast: the new key is sent individually to each device, encrypted with their respective Trust Center link key.

After distribution, the Trust Center uses the Key Switch command to instruct devices to switch to the new key, incrementing the sequence number. This mechanism ensures the network remains secure even during periodic key updates.

If a device misses a network key update, it will be kicked out of the network and will perform a trust center rejoin.

4 Introduction to Tamarin Prover

To verify the security of the specifications introduced in section 3.3, the work by Li et al. uses the Tamarin Prover tool [6][4]. From this point onward, I will refer to Tamarin Prover using the abbreviated form *Tamarin*.

Tamarin is a formal verification tool based on symbolic modeling, designed to analyze the security of cryptographic protocols. Given a protocol model and a set of security properties to verify as input, Tamarin can determine whether there are executions that violate these properties, potentially providing counterexamples.

In the following paragraphs, I will illustrate the process of defining the protocol and security properties in the context of Tamarin.

4.1 Protocol Specification in Tamarin

In Tamarin, protocol modeling and adversary capabilities are defined through rewriting rules. These rules describe how the system evolves over time, modifying the state represented by a multi-set of facts. The basic components of rewriting rules are:

- Terms: Represent messages, bitstrings or variables used in the protocol.
- Facts: Used to define information about the current system state, perform checks on this information, and record actions taken.
- System states: Represented by multi-sets of facts, describing the system configuration at a given time.

The set of rewriting rules defines a labeled transition system on which Tamarin performs verification of user-specified properties, such as secrecy, authentication, or integrity.

4.1.1 Terms

Terms in Tamarin represent data manipulated by the system, such as messages, keys, nonces or variables. They can be:

- Constants: Fixed values, like alice, bob, k1.

- Variables: Symbols that can assume different values, like x, y, k.
- Functions: Applications of functions to terms, like enc(m, k) (encryption of message m with key k) or hash(m) (hashing of message m).

Term construction is governed by the *term algebra*. Terms are used to build facts and rules. In the first case we can only use ground terms, while in the second case both ground terms and terms with variables can appear. Terms can be defined as secret (preceded by $\sim$) or public (preceded by $).

4.1.2 Facts

Facts are the building blocks of the system state. They represent information such as:

- Current state: For example, Key(alice, k1) indicates that Alice possesses key k1.
- Events: For example, ReceivedMessage(m1) records that message m1 was received.
- Checks: For example, In(m1) indicates that message m1 is available as input for the adversary.

Facts can be:

- Temporary: Consumed when a rule is applied.
- Persistent: Not consumed and remain in the multi-set even after rule application. They are preceded by (!)

The main built-in facts are: In(x), Out(x), K(x), Fr(x). The meanings of these built-in facts are respectively: presence of term x on the channel, output of term x to the channel, adversary knowledge of term x, fresh generation of a new term x.

4.1.3 Rewriting rules

Rewriting rules appear in the following form:

$$L - [A] -> R$$

where:

- L is a multi-set of facts defining the preconditions
- A is a multi-set of facts defining the actions (labels for the rule, used to track transitions)
- R is a multi-set of facts defining the conclusions

The set of multi-set rewriting rules and states ($\Gamma_t = \{F_0, F_1, ..., F_n\}$ where F_i is a multi-set of facts) defines a labeled transition system.

Given a multi-set rewriting rule R, a multi-set of facts $S_t = \{F_0, \ldots, F_n\}$ and a sequence of multi-set traces $T_t =< a_0, \ldots, a_{t-1} >$ at time t:

1. we can apply R to S_t if and only if there exists a ground instance $l - [a_t] \rightarrow r$ such that $l \subseteq^{\#} S_t$
2. after applying rule R we have that

$$S_{t+1} = S_t \backslash^{\#} l \cup^{\#} r \tag{1}$$
$$T_{t+1} =< a_0, ..., a_{t-1}, a_t > \tag{2}$$

The application of rules to system states may sometimes require substitution and unification operations [8].

Tamarin includes some built-in rules:

- Out(x) -[]→ K(x) (the adversary derives information)
- K(x) -[K(x)]→ In(x) (the adversary injects a message)
- ∅-[]→ Fr(x) (generate fresh value)

4.1.4 Equational Theories

Tamarin's symbolic proof search mechanism supports a specific class of user-defined equations, known as convergent equational theories with the finite variant property [5]. These theories are essential for modeling cryptographic properties and algebraic structures in protocols. However, it's important to note that Tamarin does not automatically verify whether the provided equations belong to this class. Consequently, defining equations outside this class may lead to non-termination or incorrect results without any explicit warning from the tool.

Key Properties of Convergent Equational Theories are:

Confluence: A convergent equational theory is *confluent*, meaning that applying the equations in any order will always lead to the same normal form. This property ensures consistency in the symbolic representation of terms during proof search. Within a confluent equational theory, if starting from a term t I can reduce to two terms t_1 and t_2, and from t_1 I can reduce to t_3, then from t_2 I can also reduce to t_3.

Termination: The theory must be *terminating*, ensuring that the term rewriting process using the equations always reaches a normal form. This guarantees that the proof search process doesn't enter infinite loops. For example, the rule $x * y = y * x$ doesn't guarantee termination.

Finite Variant Property: The theory must possess the *finite variant property*, which ensures that any term can be rewritten into a finite set of possible variants. This property is crucial for the efficiency of Tamarin's symbolic reasoning, as it allows the tool to handle equational theories in a computationally tractable way.

4.1.5 Trace

A trace represents a sequence of events (rule applications) describing system execution. Each trace is a possible protocol execution, and Tamarin verifies security properties on all possible traces. Traces include:

- Events: Actions performed, such as sending or receiving a message.
- Labels: Additional information associated with events, such as Send(m1) or Receive(m1).

Security properties are expressed as logical formulas over traces, for example: "There exists no trace where a secret message is revealed to the adversary."

Given a transition system P, the set of possible executions on P is defined as:

$$trace(P) = \{< a_1, \ldots, a_n \mid \exists S_1 \ldots \exists S_n (\emptyset \rightarrow_{a_0} S_1 \rightarrow_{a_1} S_2 \rightarrow \ldots \rightarrow_{a_{n-1}} S_n\}$$

4.1.6 Lemma

As mentioned earlier, in Tamarin, security properties are expressed as guarded fragments of first-order logic. The syntactic construct used by Tamarin to indicate a property to be proven is the **lemma**. Lemmas can be of two types:

- **Exists-trace**: These lemmas are proven by finding a single valid protocol trace that satisfies the formula.
- **All-traces**: These lemmas are proven by negating the property and attempting to find a counterexample. If no counterexample is found, the property holds for all traces.

To specify a lemma, we can combine the following atoms:

- The constant **false** ($\bot$);
- Logical operators: $\neg$ (negation), $\wedge$ (conjunction), $\vee$ (disjunction), $\implies$ (implication);
- Quantifiers and variables: $\forall$ (universal quantifier), $\exists$ (existential quantifier) and variables like a, b, c;
- Term equality: $t_1 \approx t_2$
- Temporal ordering and equality: $i \leq j$ and $i = j$;

- Action facts at temporal points: F@i (for an action fact F at temporal point i).

The semantics of trace formulas, on a trace T and a valuation Θ is:

$$
\begin{aligned}
T, \Theta \vDash F@i &\iff 1 \leq \Theta(i) \leq n \wedge \Theta(F) \in T[\Theta(i)] && (3)\\
T, \Theta \vDash i \dot{<} j &\iff \Theta(i) < \Theta(j) && (4)\\
T, \Theta \vDash i \doteq j &\iff \Theta(i) = \Theta(j) && (5)\\
T, \Theta \vDash t_1 \approx t_2 &\iff \Theta(i) \approx \Theta(j) && (6)\\
T, \Theta \vDash \neg\varphi &\iff T, \Theta \nvDash \varphi && (7)\\
T, \Theta \vDash \varphi \wedge \psi &\iff T, \Theta \vDash \varphi \wedge T, \Theta \vDash \psi && (8)\\
T, \Theta \vDash \exists x : s.\varphi &\iff \exists v \in D_s \mid T, \Theta[x \to v] \vDash \varphi && (9)\\
& && (10)
\end{aligned}
$$

A protocol P satisfies a lemma ϕ if the traces generated from ϕ are contained in the set of traces generated from P:

$$P \vDash \phi \iff trace(\phi) \subseteq trace(P)$$

5 Modeling the Zigbee Protocol in Tamarin

In this paragraph I present the Tamarin modeling of the Zigbee protocol by Li (et al.)[6]. To be consistent with the rule nomenclature in the Tamarin code, I will use the state diagrams from the original paper.

5.1 Initial Key Generation

The modeling first specifies the rules for generating fundamental keys: the *network key* by the coordinator, the *pre-configured key* shared between coordinator and device, and finally the link key shared between two devices.

Since the rules are repetitive in structure, I will only describe the rule for generating the *pre-configured key*, while for subsequent rules I will only highlight the most interesting steps.

The rule for generating the *pre-configured key* is as follows:

```
1 rule D_pck_generation:
2   [ Fr(~pck) ]
3   --[ SecretPCK(~pck) ]->
4   [ !PCK($D,$C,~pck) ]
```

The precondition `[ Fr(~pck) ]` indicates that `~pck` is a fresh value (unique and generated at runtime), where Fr is a special fact for generating fresh terms and $\sim$ denotes a secret term. The label `--[ SecretPCK(~pck) ]-->` marks `~pck` as secret, useful for verifying security properties. The postcondition `[ !PCK($D, $C, ~pck) ]` stores the key as a persistent fact (`!`), associating it with devices `$D` and `$C`, where `$` indicates message variables. In summary, the rule generates a secret PCK key, stores it persistently and associates it with two specific devices, ensuring the security of their communications.

The model proposed in the original paper also defines the rules `Reveal_nk`, `Reveal_pck`,`Reveal_ntlk`, necessary for defining lemmas where secret keys become public. In the case of the network key, the rule is:

```
1 rule Reveal_nk:
2   [ !NwkKey(C,~nk) ] --[ RevNK(C)]-> [ Out(~nk) ]
```

Applying this rule means that if C possesses a secure network key (`[!NwkKey(C,~nk)]`), it is transmitted on the channel (`[ Out(~nk) ]`).

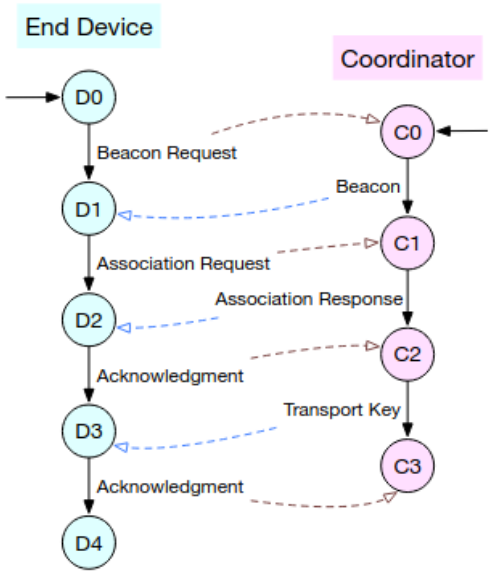

Figure 6: State diagram of network joining

5.2 Joining a New Node to the Network

To model the communication for joining a new node to the network, the authors followed the scheme in Figure 6 and described in paragraph 3.3.2.

To simulate out-of-band message transmission (particularly for the pre-configured link key), the authors defined the two rules `ChanOut_S` and `ChanOut_S`.

```
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) ]
```

The rules `ChanOut_S` and `ChanIn_S` respectively define:

- Receiving a new message `x` from the network. If in a state a message `x` exchanged by two entities `A` and `B` is received (`[ Out_S($A,$B,x) ]`), then `x` was definitely exchanged by A and B (`[ !Sec($A,$B,x) ]`)
- Sending a message `x` on the secure network. If it's always true that message `x` is secure (`[ !Sec($A,$B,x) ]`), then the message can be forwarded on the secure channel (`[ In_S($A,$B,x) ]`)

Since the protocol is modeled for both ZigBee version 1.0 and version 3.0, the authors defined the two facts `!ZigBeeV3()` and `!ZigBeeV1()`[6].

Below is the rule modeling the first Beacon request expected when a device joins the network:

```
rule D1_Beacon_req:
  [ !PCK($D,$C,~pck) ]
  --[ Beacon_req() ]->
  [ Out(<$D,'0x07'>), BeaconReq($D), !ZigbeeV3(), !ZigbeeV1() ]
```

As can be seen in the conclusion facts, both `!ZigbeeV3(), !ZigbeeV1()` are present, so after the beacon request it's possible to proceed with further rules belonging to both versions.

If a certain rule is only applicable in one of the two versions, then only one of the two facts will be present in the premise multi-set fact, as in the case of:

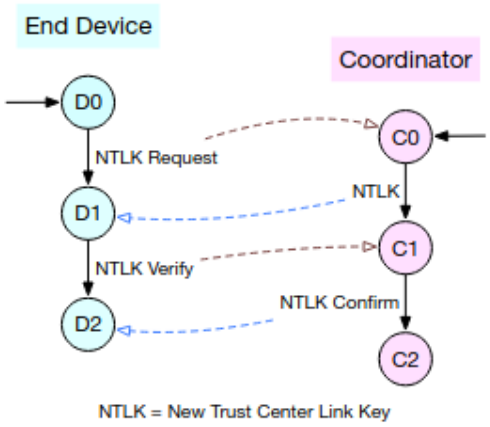

Figure 7: State diagram of trust center link key update

```
rule D2_3_association_req:
  [ In(<C,D,'panID'>),  BeaconReq(D), !PCK(D,C,~pck), !ZigbeeV3() ]
  --[ Send_Association_Req(D,C) ]->
  [  Association_req(D,C), Out_S(D,C,~pck) ]

rule D2_1_association_req:
  [ In(<C,D,'panID'>),  BeaconReq(D), !PCK(D,C,~pck), !ZigbeeV1() ]
  --[ Send_Association_Req(D,C) ]->
  [  Association_req(D,C), Out(<D,'pck'>) ]
```

While similar, in `D2_3_association_req` the conclusions involve forwarding the pre-configured key on the secure channel `Out_S(D,C,~pck)` , while version 1.0 involves in the conclusions forwarding on the channel the pair `<D,'pck'>` which is therefore transmitted insecurely.

5.3 Trust Center Link Key Update

Zigbee 3.0 introduced the need to update the Trust Center Link Key after joining the network. This step is modeled as in Figure 7.

Note that within the `D4_NTLK_verify` rule:

```
rule D4_NTLK_verify:
  [ In(senc(ntlk, ~pck)), NTLK_req(D,C), !PCK(D,C,~pck) ]
  --[NTLK_Verifiy_Req(D,C,ntlk)]->
  [ Out(senc(sdec(ntlk,~pck),(sdec(ntlk,~pck)))) ]

rule C4_NTLK_verified:
  [ In(senc(sdec(ntlk,pck),(sdec(ntlk,pck)))), NTLKSent(C,D,ntlk) ]
  --[NTLK_verified(ntlk)]->
  [ Counter(D,'0'), !SendMsg(D,C) ]
```

the notation `senc(ntlk, ~pck)` indicates that the value `ntlk` has already been encrypted with the key `~pck`. It's not an operation that happens when it's written, but represents data that was encrypted earlier. Therefore, we can interpret it as "ntlk is a key encrypted with pck." This interpretation is crucial because it explains why, when the device receives `senc(ntlk, ~pck)`, to obtain ntlk it must apply `sdec(senc(ntlk, pck), pck)`, i.e., decryption with the key `pck`.

5.4 Network Key Update

The network key update is an automatic process that the protocol executes when the frame counter associated with the network key reaches its maximum value. This mechanism is modeled by the rule:

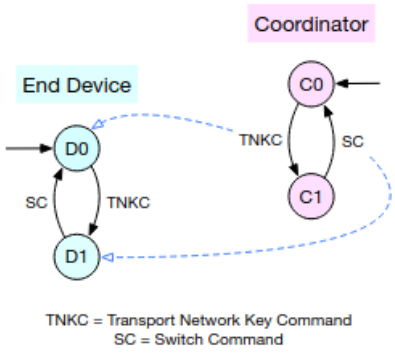

Figure 8: State diagram of network key update

```
rule Frame_counter_increase:
[ !SendMsg(D,C), Counter(D, val) ]
--[Msg_Send()]->
[ Counter(D, inc(val)), Out(inc(val)) ]
```

This rule describes how, for each message exchanged between device D and coordinator C, the counter value is incremented `(inc(val))`. However, the rule specifies that the new counter value is exposed externally via `Out(inc(val))`, making it potentially accessible to an attacker. This assumption is motivated by the fact that, in the protocol details, the counter value is not protected by any security mechanism. Consequently, an attacker could exploit this information to derive details useful for an attack, and Tamarin could identify scenarios where this exposure leads to a security violation.

Other rules do not specify the maximum limit for the counter, as this is not relevant information. Indeed, the protocol would not undergo any variations.

6 Formalization of Security Properties in Tamarin

In the formal security model developed with Tamarin, the security properties of the protocol are expressed as *lemmas*, which correspond to logical statements that must be verified within the system. Each lemma uses a combination of existential and universal quantifiers, events, and predicates to formalize crucial aspects of protocol security.

An initial group of lemmas relates to the correctness of protocol execution. For example, the lemma `execute` states that if an event `Beacon_req()` occurs at a certain instant, then there will be a subsequent moment when another event `NTLK_verified(y)` will take place. Formally, this is written as:

$$\exists\text{-trace} \quad (\exists \#i.\text{Beacon\_req}()@\#i) \Rightarrow (\exists y \#j.\text{NTLK\_verified}(\text{y})@\#j)$$

This means that, in at least one possible execution trace of the protocol, the connection request necessarily leads to verification of the session key `NTLK`.

A similar idea applies to the lemma `execute_network_update`, which ensures that a connection request implies a network update (`Update_network()`).

6.1 Formalization of Secrecy Properties

In addition to execution lemmas, the model includes secrecy properties, ensuring that certain keys do not become public unless specific conditions occur. For example, the lemma `secrecy_NK` has the form:

$$\forall x \#i. \quad \text{SecretNK}(x)@\#i \Rightarrow \neg(\exists \#j.\text{K}(x)@\#j) \quad | \quad \ldots$$

This formula establishes that if a value `x` is declared secret (`SecretNK(x)`) at a certain instant, then there will be no moment `#j` when the key `x` will be publicly known (`K(x) @ #j`), unless one of the conditions indicated to the right of the formula occurs. The exception conditions are expressions like `Ex C #r. RevNK(C) @ r`, which indicate events where key secrecy might be revoked (`RevNK`).

This pattern repeats for other critical protocol keys, such as `NTLK`, `PCK` and `LK`, with lemmas like `secrecy_NTLK`, `secrecy_PCK` and `secrecy_LK`. All follow a similar structure: secrecy is preserved unless specific revocation events intervene (`RevNK`, `RevPCK`, `RevNTLK`).

This formalization allows proving that the protocol's security properties are respected or, otherwise, identifying scenarios where they might be violated.

6.2 Formalization of Authentication Properties

In the analysis of cryptographic protocol security, *authentication* is a fundamental aspect ensuring that entities involved in communication have actually participated correctly in message exchange. The concept of *authentication* has been formalized in various ways, and Lowe's model provides a structured classification of guarantees offered by a protocol.

6.2.1 Key Agreement

A secure protocol must ensure that when two entities establish a session key, both have certainty of using the same key. This principle is expressed by the following lemma:

```
lemma key_agreement:
 "All nk1 nk2 coordinator device #i #j.
        Send_Network_Key(nk1,coordinator,device) @ i
        & Send_Network_Key(nk2,coordinator,device) @ j
          ==>
           nk1 = nk2"
```

Formally, this means that if a *coordinator* sends two keys `nk1` and `nk2` to the same device `device` at different times `#i` and `#j`, then the keys must be equal. This constraint prevents an attacker from inducing the device to believe it has negotiated a different key than the one actually established with the coordinator.

6.2.2 Key Uniqueness

In addition to ensuring there are no discrepancies in the key exchanged between two entities, it is also necessary to ensure that the same key is not sent twice at distinct time instants. This concept is expressed by the lemma:

```
lemma uniqueness_of_key:
 "All key coordinator device #i #j.
         Send_Network_Key(key,coordinator,device) @ i
         & Send_Network_Key(key,coordinator,device) @ j
           ==>
            #i = #j"
```

This property guarantees that a key is sent only once per session. If the coordinator sends a network key `key` to the device at two instants `#i` and `#j`, then these two instants must coincide. In other words, the same key cannot be transmitted multiple times at different moments, thereby reducing the risk of replay attacks.

6.2.3 Weak Agreement

Weak agreement represents the weakest level of authentication. It guarantees that if a device `d` has received a key `x1` from coordinator `c`, then at least one of the following conditions must be true:

1. The coordinator has actually sent a network key (`Send_Network_Key(c,d,x2)`)
2. The key has been compromised and made public (`RevNK(c)`)
3. The preconfigured key `PCK` between coordinator and device has been revealed (`RevPCK(d,c)`)

```
lemma weak_agreement:
  " All c d x1 #i.
        NwkKeyRecv(d,c,x1) @ #i
        ==>
         ((Ex x2 #j. Send_Network_Key(c,d,x2) @ #j)
        |(Ex #r. RevNK(c) @ r)
        | (Ex #r. RevPCK(d,c)  @ r))"
```

This property ensures that a device cannot claim to have received a key without there being a justification for this claim, i.e., without an actual transmission or key compromise.

6.2.4 Non-Injective Agreement

A stronger level of authentication is *non-injective agreement*. Unlike *weak agreement*, this lemma requires that the received key `x` must exactly match the one sent by coordinator `c`. However, there is no constraint on the possibility of the same key being used in multiple sessions.

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

Here the difference from `weak_agreement` is that different keys `x2` are not considered: the received key `x` must be exactly the one sent. However, the lemma does not require each execution to be distinct, so the same `x` could be reused in multiple sessions without invalidating the property.

6.2.5 Injective Agreement

Injective agreement is the strongest level of authentication and requires not only that the received key be exactly the one sent, but also that there is a well-defined temporal order.

```
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)
        | (Ex #r. RevPCK(d,c)  @ r)"
```

This lemma establishes that if `d` receives key `x` from coordinator `c` at instant `#i`, then there must have been a previous moment `#j < #i` when `c` sent that key.

Unlike non-injective agreement, here it is required that each execution be distinct and temporally ordered, ensuring the protocol does not allow replay attacks or key reuse in different sessions.

7 Analysis of Results: Detected Vulnerabilities

The verification conducted using the Tamarin tool demonstrates that Zigbee 3.0 satisfies all expected security properties. In contrast, the Zigbee 1.0 version presents vulnerabilities regarding key confidentiality. By accessing Tamarin's web interface and loading the model that formalizes the protocol and the security properties described in sections 5 6 [10], the lemma `secrecy_NK` is highlighted in red on the left side (Figure 9). Clicking on the `SOLVED` entry displays the cases of lemma violation. Figure 10 shows the constraint system that justifies the key confidentiality violation. The use of a pre-configured key whose secrecy is not required by the protocol specifications allows decrypting the last message exchanged by rule `C2_1_Send_Nwk_Key`, thus enabling the attacker to derive the network key.

```
lemma secrecy_NK:
  all-traces
  "∀ x #i.
        (SecretNK( x ) @ #i) ⇒
        ((((¬(∃ #j. K( x ) @ #j)) ∨ (∃ C #r. RevNK( C ) @ #r)) ∨
          (∃ C D #r. RevPCK( D, C ) @ #r)) ∨
         (∃ C D #r. RevNTLK( C, D ) @ #r))"
simplify
solve( !KU( ~nk ) @ #vk )
  case C2_1_Send_Nwk_key
  SOLVED // trace found
next
  case C2_3_Send_Nwk_key
  by sorry
next
  case Reveal_nk
  by contradiction /* from formulas */
qed
```

Figure 9: Tamarin found a trace that violates the lemma Secrecy_NK

8 Conclusions

The analysis conducted on the Zigbee protocol yielded two significant results. First, it allowed identifying a serious security vulnerability in version 1.0, while simultaneously demonstrating that version 3.0 corrects these issues, ensuring the confidentiality and authenticity of exchanged messages. Second, this study represents another example of Tamarin-prover's effectiveness as a tool for formal verification of cryptographic protocols, highlighting its ability to detect security flaws and validate corrections.

References

[1] ZigBee Specifications 2012. URL: http://www.zigbee.org/wp-content/uploads/2014/%2011/docs-05-3474-20-0csg-zigbee-specification.pdf.

[2] Zigbee Base Device Behavior Specification Version 1.0 2016. URL: http:%20//www.zigbee.org/wp-content/uploads/2014/10/docs-13-0402-13-00zi-Base-%20Device-Behavior-Specification-2.pdf.

[3] Zigbee Security Basics 2017. URL: https://research.kudelskisecurity.com/2017/11/08/%20zigbee-security-basics-part-2/.

[4] David Basin et al. "Symbolically analyzing security protocols using tamarin". In: *ACM SIGLOG News* 4.4 (Nov. 2017), pp. 19–30. DOI: 10.1145/3157831.3157835. URL: https://doi.org/10.1145/3157831.3157835.

[5] Hubert Comon-Lundh and Stéphanie Delaune. "The finite variant property: how to get rid of some algebraic properties". In: *Proceedings of the 16th International Conference on Term Rewriting and Applications*. RTA'05. Nara, Japan: Springer-Verlag, 2005, pp. 294–307. ISBN: 3540255966. DOI: 10.1007/978-3-540-32033-3_22. URL: https://doi.org/10.1007/978-3-540-32033-3_22.

[6] Li Li, Proyash Podder, and Endadul Hoque. "A formal security analysis of ZigBee (1.0 and 3.0)". In: *Proceedings of the 7th Symposium on Hot Topics in the Science of Security*. HotSoS '20. Lawrence, Kansas: Association for Computing Machinery, 2020. ISBN: 9781450375610. DOI: 10.1145/3384217.3385617. URL: https://doi.org/10.1145/3384217.3385617.

[7] G. Lowe. "A hierarchy of authentication specifications". In: *Proceedings 10th Computer Security Foundations Workshop*. 1997, pp. 31–43. DOI: 10.1109/CSFW.1997.596782.

[8] Alberto Martelli and Ugo Montanari. "An Efficient Unification Algorithm". In: *ACM Trans. Program. Lang. Syst.* 4.2 (Apr. 1982), pp. 258–282. ISSN: 0164-0925. DOI: 10.1145/357162.357169. URL: https://doi.org/10.1145/357162.357169.

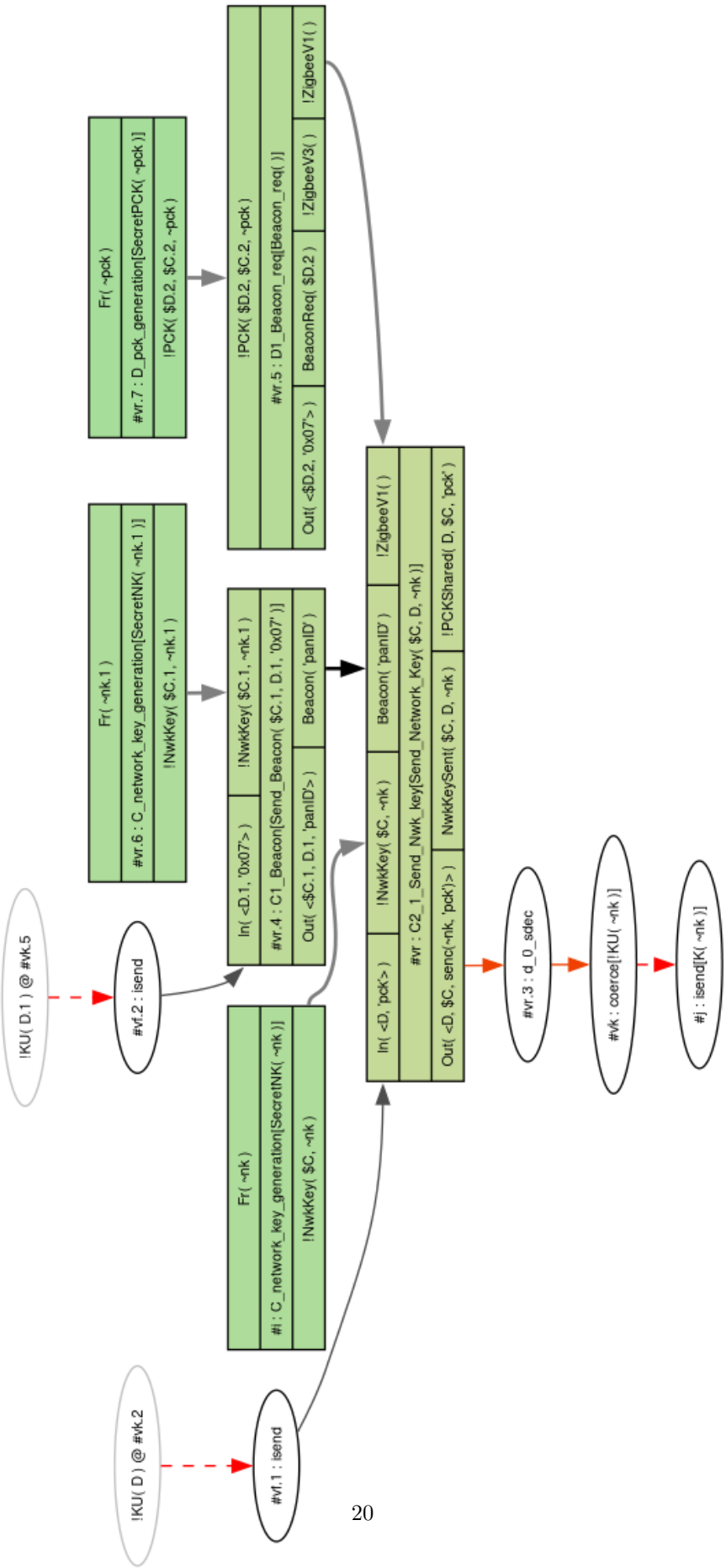

Figure 10: Network Key Disclosure

[9] Yahoo Finance - Zigbee market spread. URL: https://finance.yahoo.com/news/zigbee-market-projected-hit-usd-145700998.html?guccounter=1&guce_referrer=aHR0cHM6Ly93d3cuZ29vZ2xlLmNvbS8&guce_referrer_sig=AQAAACGyZdU9lFYpmkk8SURvRoPDE60vY3Oul-Rx2p_m2UXrsFgoIZxwH1DTM6NY_AS-dsaq0H4CwfMXLxz0vyZALrCS-Tba9EOeXNKgUkQBMIR-3fxysfVn19IEFzNDMZbMn2J_Hg0PFa91alS4SEJTA-nKzfT939oMdZthVX_ihQ_-.

[10] Li-Syr. *Li-Syr/Zigbee-Tamarin*. URL: https://github.com/Li-Syr/zigbee-tamarin.

[11] Dell Technologies. URL: https://www.dell.com/support/kbdoc/en-us/000067530/what-is-an-organization-unique-identifier-o-u-i.