Title: Security and Key Establishment in IEEE 802.15.4

Student: Eirik Klevstad

Problem description:

Internet of Things (IoT) is a network where devices, sensors, vehicles, buildings, and humans communicate and collaborate, along with collecting and exchanging information. IEEE 802.15.4 specifies the lower layers for low-rate wireless networks, which are widely seen as the foundation for current IoT communications. One of the potential weaknesses of the IEEE 802.15.4 standard is the lack of specification for key establishment and management.

This thesis will focus on key management for device-to-device security in IoT. It will review and compare the proposed protocols, and include both formal and informal security analysis, as well as analysis of both key management requirements and key agreement protocol design for IoT security. Another goal of the thesis will be to suggest improvements and alternatives to the proposed protocols.

Responsible professor: Colin Boyd, ITEM

Supervisor: Britta Hale, ITEM

Abstract

This is my abstract.

Preface

Hello, here is some text without a meaning. This text should show what a printed text will look like at this place. If you read this text, you will get no information. Really? Is there no information? Is there a difference between this text and some nonsense like “Huardest gefburn”? Kjift – not at all! A blind text like this gives you information about the selected font, how the letters are written and an impression of the look. This text should contain all letters of the alphabet and it should be written in of the original language. There is no need for special content, but the length of words should match the language.

Contents

List of Figures vii

List of Tables ix

List of Acronyms xiii

1 Introduction 1

2 Background 3
2.1 Internet of Things . . . 3
2.2 The IEEE 802.15.4 Standard . . . 5
2.3 6LoWPAN: Putting IP on Top of 802.15.4 . . . 8
2.4 Key Establishment and Key Management . . . 9
2.5 Formal Security Analysis . . . 11

3 Symbolic Security Analysis Using Scyther 13
3.1 The Scyther Tool: Verification, Falsification, and Analysis of Security Protocols . . . 13
3.2 Scyther Syntax . . . 15
3.2.1 Security Claims . . . 16
3.3 Scyther's Graphical Interface . . . 20

4 6LoWPAN Security - Adding Compromise Resilience to the 802.15.4 Security Sublayer 23
4.1 Ideas of the Paper (Change this) . . . 23
4.2 Informal Analysis . . . 23
4.3 Formal Analysis . . . 24
4.3.1 Scyther . . . 24

5 Handling Reboots and Mobility in 802.15.4 Security 27

References 29

List of Figures

2.1 The Open Systems Interconnection (OSI) stack with layers, the data they carry, and an example of technology running on the different layers. . . 6
2.2 Figure of IEEE 802.15.4's operational space compared to other wireless standards [17]. 7
2.3 Figure of the IPv6 over Low power Wireless Personal Area Networks (6LoWPAN) stack, which uses the 802.15.4 physical and link layer, but adds an adoption layer in the network layer. 9

3.1 Results of a verification process using Scyther. 20
3.2 Results of a verification process using Scyther when a claim fails. 21
3.3 When Scyther finds an attack on a protocol, if will also provide a graph of the attack. 21

4.1 Adaptable Pairwise Key Establishment Scheme (APKES) is positioned in the data link layer in the 6LoWPAN stack expanding the 802.15.4 security sublayer [22]. 24
4.2 APKES consists of a three-way handshake, and utilizes a pluggable scheme for deriving the shared secret between two nodes [22]. 25

List of Tables

List of Acronyms

6LoWPAN IPv6 over Low power Wireless Personal Area Networks.

AES Advanced Encryption Standard.

AES-CCM Advanced Encryption Standard Counter with CBC-MAC.

APKES Adaptable Pairwise Key Establishment Scheme.

BAN Burrows-Abadi-Needham.

CBC Cipher Block Chaining.

CCM Counter with CBC-MAC.

CIA Confidentiality-Integrity-Availability.

CTR Counter.

GPS Global Positioning System.

GUI Graphical User Interface.

IETF Internet Engineering Task Force.

IoT Internet of Things.

IP Internet Protocol.

MAC Media Access Control.

MAC2 Message Authentication Code.

MSC Message Sequence Chart.

OSI Open Systems Interconnection.

RAM Random Access Memory.

RFID Radio Frequency Identification.

SNMP Simple Network Management Protocol.

SPDL Security Protocol Description Language.

TCP Transmission Control Protocol.

UDP User Datagram Protocol.

WPAN Wireless Personal Area Network.

WSN Wireless Sensor Network.

Chapter 1
Introduction

This should be the introduction to the thesis.

Chapter 2
Background

2.1 Internet of Things

Over the last decade, a concept called the *Internet of Things* has gained increased attention, both from the research community and commercial actors, as well as from consumers. The term IoT was, accordingly to most sources, coined in 1999 by the British visionary Kevin Ashton in a presentation about Radio Frequency Identification (RFID) [2] [31]. Ashton's definition of the concept was a world where computers do not depend on human beings to provide them with information. Out of all the petabytes of information available on the Internet, the majority has been created and captured by humans performing some sort of action. In his opinion, IoT is about providing computers with the ability to gather information on their own.

A computational device containing some sort of sensor is attached to your everyday physical device, creating a bridge between our physical world and the cyber world [21]. The connection to the Internet allows us to monitor and control these devices and sensors from a remote distance. Another vital part of IoT is device-to-device communications, essentially enabling devices to communicate with each other without human aid, and exchange and retrieve information. Such devices could be sensors monitoring some operation, a physical area, or even attached to a physical body. The possibilities are more or less unlimited. Imagine a home automation and surveillance system for your cabin, where lights, heaters, smoke detectors, underfloor heating, motion detectors, security cameras, garage and so on, are all interconnected with each other through small wireless devices. As it is called the *internet* of things for a reason, your system and devices would be accessible over the Internet, allowing you to monitor the current status of your cabin remotely from your couch at home, as well as looking at historical data of the different sensors and devices. When the weekend arrives and you head for the mountains, the IoT provides you with an opportunity to preheat different (or all) sections of the cabin, deactivate the alarm, and perhaps instruct the sauna to start getting cosy.

Another approach is to avoid using a monitor to remotely control the system, and instead allowing the system to observe and act on your behaviour. We want the devices to know us and figure out the correct thing to do without us telling them. For example, when pulling your car into the driveway, you want the garage door that is connected with your car to open up. The garage notifies your front door that you are home, which conveniently unlocks and notifies your house to turn on the lights in your hallway and perhaps the heater in your living room.

The possibilities that are revealed as the IoT grows larger and the services expand are infinite. The concept is highly applicable for different scenarios involving home automation, standalone consumer products, industrial and environmental facilities, as well as medical surveillance. While larger automation systems for homes and facilities have been the target for the research community and early adopters, the consumer market has been focused on so-called *wearables*. Wearables are fundamentally devices that you wear, such as smart watches, fitness trackers, virtual reality glasses, headphones, and smart clothing. Such human-centric devices are less about automation, and more focused on personal improvement. Nevertheless, the increase in IoT devices possibly provides us with a more cost efficient future, both in our use of time, as well as energy and consumption of other resources.

As the IoT is built upon the Internet, it faces the same types of security issues as the Internet itself. The amount of "things" connected to the Internet is calculated to be 6.4 billions by the end of 2016, which is almost a 30% increase from 2015. By 2020, the expected number of these "things" is more than 20 billion [16], providing attackers with equally many possible devices to attack. Given the knowledge that some of these devices may be medical (or have other sensitive applications), we quickly recognize potential catastrophic scenarios.

The IoT architecture can resemble the neural system of the human body. The perception layer controls our sensors which we use to obtain information about our environment by observing, feeling, smelling, tasting or hearing. As previously described, IoT devices are often deployed with one or more sensors to perform these "human operations" for information collection. The perception layer is mainly focusing on sensing and allowing IoT devices to observe their environment and collect information. Examples of such technologies are RFID, Wireless Sensor Network (WSN), and the Global Positioning System (GPS) [19]. Information from our human sensors are carried to the brain through a neural network. Much alike in IoT, the collected information is transmitted using the transportation layer. The transportation layer is running over some wireless or wired medium such as 802.15.4, 6LoWPAN, 3G, Bluetooth or Infrared. Finally the information is processed by an intelligent entity. In our human body example, this would be the brain. In the IoT, the brain would be an intelligent processing unit in the application layer which is

able to compute and evaluate actions based on the received information [20] [33]. The application layer is also responsible for controlling the sensors, and performing global system management, and present data for the end user of the system.

As these layers covers different characteristics of IoT, they consists of different types of hardware and provide different types of services, hence they are subject to different types of security threats and solutions. The most adjacent problems to the scope of this thesis are the problems related to key establishment and key management, which define how two devices safely can establish secure communication between each other. Or in other words, how collected information is safely transmitted between the sensors and the "brain".

In an IoT world, the protection of data and privacy is an essential part. As previously mentioned, IoT technology may be a solution for problems involving sensitive information. In a medical facility, a possible scenario could be a WSN, which is a dynamic and bi-directional network of nodes where each node has one or more sensors connected to it. A patient may have sensors implanted in their body, as well as different instruments attached for measuring different properties. All these devices communicate with each other wirelessly, and the network is therefore a possible target for an attacker. To prevent the attacker from eavesdropping, and possible forging content in the network, encryption and authentication at the different nodes is crucial.

2.2 The IEEE 802.15.4 Standard

Following the evolution of IoT, the need for cheap devices to communicate efficiently between each other has arisen. Existing architectures such as 802.11 (WiFi) and Bluetooth are too expensive in terms of processing and energy consumption, as the idea of IoT is to connect even the smallest devices to a network or Internet. As these devices are small, they have a limited battery life, and hence need to use energy in a highly efficient matter.

Protocols using the IEEE 802.15.4 standard are envisioned for applications supporting smart homes, medical surveillance, monitoring systems for environmental and industrial systems, as well as sensor systems for heating and ventilation. As we know from the IoT, it is really the imagination that puts an end to the possibilities for interconnected devices. The OSI stack defines the internal structure of communications systems, and is shown in Figure 2.1. As the 802.15.4 standard only defines the physical and data link layer of the OSI stack, which can be seen in Figure 2.1, specifications need to be developed to utilize the possibilities provided by 802.15.4 in the upper layers. ZigBee [1], maintained by the ZigBee Alliance, is the most notable example of specifications that uses 802.15.4 as its base. Others include

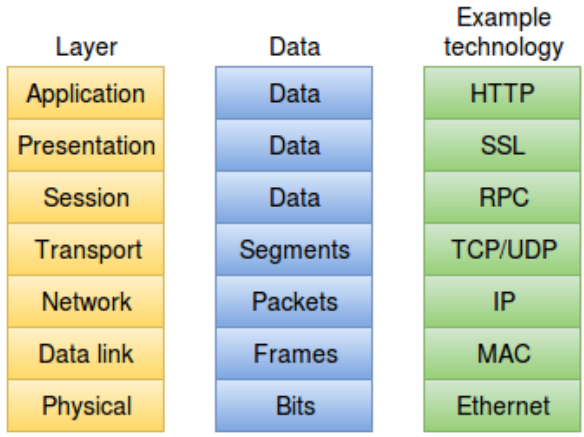

Figure 2.1: The OSI stack with layers, the data they carry, and an example of technology running on the different layers.

WirelessHART [15], MiWi [32], and ISA100.11a [28].

The fundamental intention of the 802.15.4 standard is to provide low-rate, low-power communication between devices within a sensor network or Wireless Personal Area Network (WPAN). Its main use case is to let multiple devices within a short range communicate with each other over a low-rate radio, while maintaining a modest energy consumption. Figure 2.2 paints a picture of what 802.15.4 is, compared to more well-established concepts such as WiFi (802.11) and Bluetooth, focusing on energy consumption, complexity and date rate. While being smaller and more cost efficient than those found in more complex networks, devices running on 802.15.4 networks have a much more limited range (about 10 meters), and in most cases a throughput below 250 Kbps [17]. Not only is the 802.15.4 standard significantly lighter in terms of data rate and power consumption, it is also aimed at a different market than regular WPANs. WPANs are oriented around a person, creating a personal network for the user, which has higher demands to data rate, and can allow a higher energy consumption. 802.15.4, however, focuses on interconnecting devices that do not necessary have this constraint, such as industrial and medical applications.

Four basic security services are provided in the 802.15.4 link-layer security package, namely access control, message integrity, message confidentiality, and replay protection (sequential freshness) [30]. The IEEE 802.15.4 standard is delivered with a total of eight different security suites, providing none, some, or all of the described security services, and it is up to the application designer to specify and enable the desired security properties. In the most secure end of the scale we find Advanced

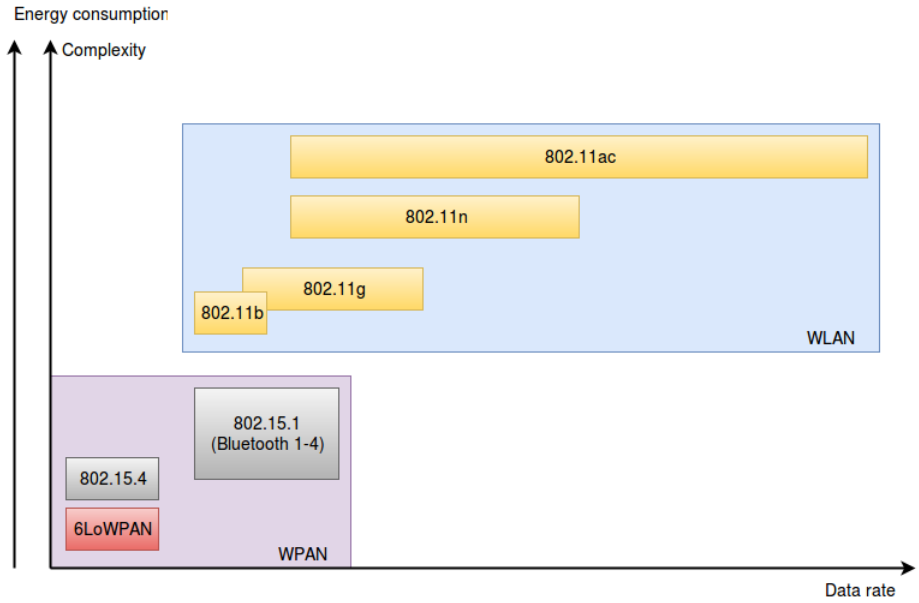

Figure 2.2: Figure of IEEE 802.15.4's operational space compared to other wireless standards [17].

Encryption Standard (AES)-Counter with CBC-MAC (CCM), which is encryption using the block cipher AES with either 32, 64 or 128-bit Message Authentication Code (MAC2). Such a suite provides both strong encryption and possibly unforgeable messages (a 64-bit MAC2 gives an adversary a 2^{-64} chance of successfully forging a message, and is used to enable legitimate nodes in the network to detect if a received message have been tampered with). On the other end of the scale we find a suite providing only confidentiality using AES in Counter (CTR) mode. This suite does not, however, provide any form of authentication – giving adversaries the possibility to forge messages, which can not be said to be especially secure.

One of the things the 802.15.4 standard does not specify is how to deal with key establishment and key management, which therefore has to be dealt with in the higher layers.

There exists multiple keying models, which essentially are descriptions on what type of key a node uses to communicate with other nodes. Examples of such models include network shared keying, where the entire network share the same symmetric key, and pairwise keying, where each node pair agree upon a key for them to use when communicating with each other. Over the years, many different proposals have surfaced regarding the most secure and efficient way of establishing keys between

nodes, and how to perform key management. Some of them have been good, some of them not so good. The Diffie-Hellman Key Exchange is perhaps one of the more successful ... However, recent result

2.3 6LoWPAN: Putting IP on Top of 802.15.4

Initially, the Internet Protocol (IP) was considered to be too "heavy" for low-power wireless networks such as the ones described by the 802.15.4 standard. The idea of implementing IP on top of 802.15.4 networks was born as early as 2001 under the question "Why invent a new protocol when we already have IP?"[26]. With IP, the community already had a bundle of existing protocols for management, transport, configuring and debugging, such as Simple Network Management Protocol (SNMP), Transmission Control Protocol (TCP) and User Datagram Protocol (UDP), as well as standardized services for higher layers such as caching, firewalls, load balancing, and mobility. Nevertheless, the initial idea of using IP in combination with sensor networks or WPANs was not accepted by various groups such as ZigBee [26]. The rejection did not, however, stop the initiative, and a group of engineers within Internet Engineering Task Force (IETF) started designing and developing what would later be known as 6LoWPAN.

A significant advantage with combining IP and 802.15.4 is the simplification of the connectivity model between various devices in the networks. As most 802.15.4-based specifications usually need custom hardware that tends to be complex, the possibilities to interconnect different networks with each other is somewhat limited. By turning to IP, the need for complex connectivity models is obsolete as it is possible to use well-understood technologies such as bridges and routers. Another advantage with using IP is that the routers between the 6LoWPAN devices and the outside networks (so-called edge routers) do not need to maintain any state as they are only forwarding datagrams.

6LoWPAN enables wireless 802.15.4 sensor devices to connect directly to the Internet via IPv6 by providing an adoption layer in the network layer between it and the data link layer as shown in Figure 2.3. The adoption layer provides a unique functionality which both fragments and compresses incoming packets to enable the embedded devices in 802.15.4 networks to receive the packets while using the least amount of memory and energy [22]. Its fundamental idea is that you only "pay" for what you use. The dispatch header field identifies the type of header to follow, and consists of 1 byte [26]. Such a header starts with either 00 or 01, respectively indicating whether the frame is a non-6LoWPAN frame or a regular 6LoWPAN-frame. Currently, only five different dispatch headers have been defined [18]. Therefore, there is a fair space for new headers as the standard and technology evolves. However, the special case of a header consisting solely of ones, adds an additional byte to the

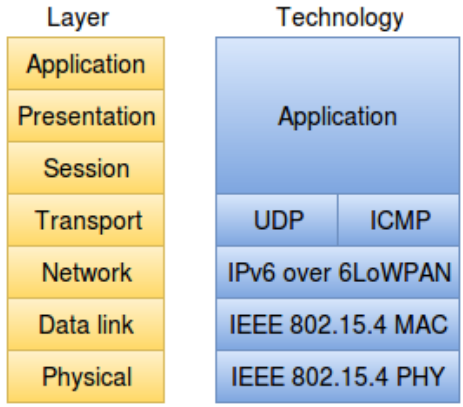

Figure 2.3: Figure of the 6LoWPAN stack, which uses the 802.15.4 physical and link layer, but adds an adoption layer in the network layer.

header, enabling a total of 320 different header types [26]. This greatly differs from IPv4 and Zigbee which define only one monotonic header, and can be used to greatly minimize the header size of a packet as some types of frames may consist of smaller payloads than others.

Compared to other alternatives such as ZigBee or Z-wave, 6LoWPAN's implementation did not prove to be any more expensive in terms of code size and Random Access Memory (RAM) requirements. 6LoWPAN seems to be a natural choice for the future IoT as a networking protocol. It is scalable thanks to IPv6, and can be compressed to a only a few bytes using its fragmentation and compression mechanism. Following the expected bloom in IoT devices over the next few years (20 billion by 2020), and the fact that the IPv6 address space is not going to be exhausted any time soon (roughly 2^{95} addresses for each and every one of us), 6LoWPAN may be a reasonable approach.

2.4 Key Establishment and Key Management

As described, devices in IoT are supposed to communicate between each other, usually over some network. There is, however, not always a guarantee for that the network used for communication is secure, but rather the opposite. An attacker may be eavesdropping on the network, and may even be capable of intercepting and spoofing traffic sent between different nodes. From a security perspective, an attacker as described is violating both the confidentiality and integrity of the exchanged information. To cope with this, devices should be encrypting and authenticating the data that they are exchanging.

In modern cryptography, encryption and decryption is in most cased done either by using symmetric key encryption or asymmetric key encryption. Symmetric key encryption is the case where communicating parties possesses the same key which is used for encryption as well as decryption. While being a straightforward and fast way of encrypting information, it has a major drawback in the case of one of the parties is compromised, then the channel that initially was secure would now be insecure as the adversary could easily encrypt any message that it intercepts. Asymmetric encryption (or public-key encryption) is the case where each communicating party possesses a public key and a private key. The public key is published and is used by anyone who wants to send an encrypted message to the owner. When the party receives a message that is encrypted with its public key, it uses the private key which is generated from the public key to decrypt the message. That way, in case of being compromised, the party could simply generate a new key pair consisting of a public and a private key, and publish the new public key for others to send encrypted messages under.

Key establishment, is a fundamental idea where two communicating parties exchange cryptographic keys which enable them to perform some sort of cryptography on the messages that are sent between them. The problem is, however, how do we safely agree upon the keys to use in the encryption-decryption process if the network itself can not be trusted?

In the 1970s, Whitfield Diffie and Martin Hellman introduced the Diffie-Hellman key exchange, which was one of the first practical examples of public key exchange within cryptography [**?**].

Key management. How to store keys.

Authentication is an important aspect of key establishment. More specific, confirming the identity of the party you are establishing keys with. If authentication is skipped, then the protocol will be weak for so-called man-in-the-middle attacks where an adversary intercepts and relays messages between two communicating parties to learn or modify its content.

There exists multiple key models, or schemes for how to perform key management

Authentication

Network shared key Fully Pairwise keys Public Key Cryptography

One of the traditional key exchange infrastructures for the Internet is to use a trusted third party. Usually this third party is a secure key server that is responsible for serving cryptographic keys to users and programs. The keys that are provided

by the key server usually included in a certificate containing additional information about the identity of the owner of that particular key. Example of such systems in the well-known public key system OpenPGP.

However, for WSN and other types of sensor networks in IoT, such an approach is infeasible due to the unknown topology of the network, communication range constraints, complexity, and network dynamics [14]. A possible approach is so-called key pre-distribution, where the keys used for securing communication is implemented in the devices before deploying them in their intended environment. This is, however, an unsafe option. Because of devices often being deployed in hostile environment where they are left unattended, they are also susceptible to be compromised by an attacker [22]. The attacker may physically tamper with the device to obtain cryptographic keys, which then can be used to decrypt the communication in case of the network using a network shared key. Another scenario is in the case of the system utilizing a pairwise key scheme, where each node pair share the same symmetric key. This leads to high memory requirements and complexity for the devices, and will also make if significantly harder for new nodes to join the network.

2.5 Formal Security Analysis

As security protocols grows larger and more complex, they become more and more difficult for humans to analyse. One of the examples of the need for formal security analysis is the Needham-Schroeder protocol [27] from 1978. The Needham-Schroeder Public-Key Protocol is based on public-key cryptography and was intended to allow two communicating parties to mutually authenticate each other. Throughout this section, the protocol will be used as an illustrative example to underline the importance of formal security analysis.

One of the pioneering works on security analysis was conducted by Burrows, Abadi and Needham with their Burrows-Abadi-Needham (BAN) logic. BAN logic is a set of rules which can be used to determine whether received information is legit or not, by formally describing the interaction between communicating parties [5]. It showed promising results in finding security flaws and drawbacks for several authentication protocols, but was later abandoned due to the fact that it verified insecure protocols as secure, and in some cases perfectly sound protocols to be insecure [25]. One of the protocols that was formally verified using BAN logic was the Needham-Schroeder protocol.

In fact, 17 years later after being deployed and widely used, Lowe discovered, using the automatic tool Casper, that the Needham-Schroeder protocol was in fact insecure, and vulnerable to a man-in-the-middle attack [3] [23]. The discovery of that such a flaw had gone unnoticed for so many years puzzled the research community,

leading to an increased interest in formal security analysis [11]. Researchers started developing tools for exhaustive search of the problem space of a protocol in order to detect possible abnormalities in protocol behaviour.

The Dolev-Yao model is a formal model used to prove the security properties of cryptographic protocols. While initially being a verification model built for public key protocols, the Dolev-Yao model is also the basis for most of the security analysis done by verification tools [12]. The model is built upon three primary assumptions [13]: Perfect cryptography, complete control of network, and

Firstly, the Dolev-Yao model assumes that the cryptography is perfect, essentially meaning that the cryptographic system can not be tampered with, and an encrypted message can only be decrypted by the party possessing the corresponding decryption key. The second assumption is that the adversary has complete control over the communication network, hence he is able to observe all messages that are sent between communicating parties, and can inject messages given that he is able to forge its content in a valid matter. Lastly, messages that are sent in the network are to be observed as abstract terms, where the attacker has two possible outcomes; either he learns the complete content of the message, or he learns nothing at all.

More to come.

Chapter 3
Symbolic Security Analysis Using Scyther

There exists multiple state-of-the-art tools for performing formal analysis of security protocols, for example Avispa [29], ProVerif [4] and Scyther [6]. This thesis uses Scyther as its tool for conducting formal security analysis, and the following chapter will give an introduction to Scyther, how it works, and examples of usages.

3.1 The Scyther Tool: Verification, Falsification, and Analysis of Security Protocols

Scyther is a tool for both verification, falsification, and analysis of security protocols developed by Cas Cremers. The tool is based on a patter refinement algorithm that enables unbounded verification, falsification and characterization [8]. Scyther allows its users to verify security protocols in two different ways. The first option is to execute Scyther scripts through the command-line interface, which provides an output file containing the results of the protocol verification. Option two, is to use Scyther's own Graphical User Interface (GUI), which provides panels for both verification results, and in case of attacks being found; a visual graph of Scyther's proposed attack on the protocol. The most recent release of Scyther was published on April 4, 2014, and is currently available for Windows, OS X and Linux.

A security protocol's specification describes messages that are sent between parties and computation done at either party's side. This can be seen as the blueprint of what the different entities are actually allowed to do, and how they communicate with each other [7]. Such a blueprint can be modelled using Scyther to analyse attacks on the protocol, as well as discovering possible protocol behaviours and execution patterns.

Most algorithms used in verification tools perform *bounded* verification, which means that they are capable of handling a finite space of protocol behaviours. This is a restriction, implying that the algorithm is not able to verify all possible behaviours (or states) of the protocol, but rather a finite subset of the possible state-space.

One of Scyther's unique features is its ability to handle an *unbounded* (or infinite) state-space, and that it is guaranteed to *terminate* [9]. By constructing an algorithm that terminates every time, it allows for providing useful results, even when no attack is found, or if the algorithm is not able to verify the protocol in the unbounded state-space [8].

Scyther performs its analyses under the *perfect cryptography assumption* [9]. This assumption states that all cryptographic functions are perfectly designed, hence it is impossible for the adversary to learn anything about the encrypted message without possessing the corresponding decryption key.

As mentioned, a protocol specification contains a set of roles which serves as blueprints that describes what the protocol is able to do. When executing the protocol, each of the different roles can be executed multiple times, and in parallel with each other by one or more agents [7]. An execution of a role is referred to as a *run*, and defines an unique instance of the protocol with respect to local constraints and the binding between the role and the actual agent acting out the roles behaviour.

Scyther allows its user to state security claims which are evaluated at the end of each run of the protocol, either ending in a successful verification of the security property, or in a failure. In the presence of a failure for some security property, Scyther will also provide a concrete attack on the protocol, and will also present an attack graph to easier illustrate the possible threat. If the protocol developer is unsure of what types of claims that should be stated for each role, Scyther has support for so-called verification of automatic claims, where Scyther will provide the appropriate claims for each variable or key based on its type.

Another novel feature that Scyther provides is complete characterization of the protocol, which creates a finite set of possible traces. A trace is a possible way for the protocol to execute, and it is rare for protocols to have more than 1-5 different traces [8]. By using Scyther to characterize the different roles, it is possible to discover possible malicious ways for an attacker to execute the protocol, and implement precautions or change the protocol to avoid such execution patterns.

One of the major novelties in Scyther is the possibility for performing so-called multi-protocol analysis, which in short terms is to analyse multiple protocols that are co-existing in the environment. Such an analysis has previously been infeasible because of an incredible wide state-space, but thanks to Scyther's unique algorithm that operates on an unbounded state-space, it allows for conducting multi-protocol analysis.

For analysis of protocol security, Scyther provides different classes of attacks, which deflects from other available analysis tools.

3.2 Scyther Syntax

The syntax used in `.spdl`-files, which are protocol files that can be run and verified by Scyther, can resemble popular object-oriented languages such as C, C++ or Java. The structure of a minimum working example for the protocol known as APKES is shown below, consisting of an outer class defining the protocol and multiple agents (or roles) inside the protocol. In this example, we define that our protocol consists of two communicating parties; U and V, without giving them any specific behaviour.

```
protocol APKES(U, V){
  role U { };
  role V { };
};
```

For each of the different roles in the protocol, behaviour can be added as a sequence of send and receive events, as well as variable declarations, constants and claims. For the U role, we can define a simple behaviour as shown below, where U generates a random nonce `Ru` and sends it to V, before receiving a message from V containing the random nonces `Ru` and `Rv`. All events are labelled with either `send` or `recv` followed by a subscript and a number. The number indicates the message's position in a Message Sequence Chart (MSC), and must be incremented for each message sent.

```
role U{
  fresh Ru: Nonce; # Freshly generated nonce
  var Rv: Nonce; # Variable for receiving a nonce

  send_1(U, V, Ru); # Send message to V containing Ru
  recv_2(V, U, Ru, Rv); # Receive message from V containing
   Ru and Rv. Store Rv in variable
};
```

Typically, a `send`-event has a corresponding `recv`-event at the receiving role with the same number.

```
role V{
  [...]
```

```
  recv_1(U, V, Ru); # Receive message sent from U containing
    Ru. Store Ru in variable
  send_2(V, U, Ru, Rv); # Send message to U containing Ru
    and Rv
};
```

Along with support for creating fresh nonce, variables and terms, Scyther also provides a wide set of cryptographic elements such as hash functions, symmetric-key cryptography, public-key cryptography, as well as declaring user specific types and macros, which are abbreviations of complex expressions into simpler once. In the following example, a hash function is used to define a function that generates a Message Integrity Code (MIC) (which is essentially the same as a MAC2). One the next line, we have created a macro representing the generation of a pairwise key between U and V. The key is represented as an encryption of the two values `Ru` and `Rv` using a symmetric key that is shared between U and V. Constants and functions defined outside of a role are considered to be global, and available to all of the defined roles in the protocol.

```
hashfunction MIC; # An hashfunction to represent a Message
    Integrity Code (MIC) generation.

macro PairwiseKey = {Ru, Rv}k(U, V);
```

3.2.1 Security Claims

A sequence of events within a role are usually followed by a set of claim events. Claim events are used for describing security properties of a role, for example that some value should be considered secret, or that certain properties holds for authentication. Such claims can be verified using Scyther. If the protocol is not instructed with any security claims, Scyther is able to generate the appropriate claims for the protocol with respect to secrecy and authentication.

Secret

The first, most trivial security claim is secrecy. Secrecy expresses that the stated property is to be kept hidden from an adversary, even in the case of where the adversary controls the network used for communication. However, if one of the agents gets compromised by the adversary and the protocol is executed between an honest agent and the adversary, it would in the end learn what was meant to be kept hidden from it [12]. The secrecy claim does not hold for such cases (nor is it intended to),

but for each case where the protocol is executed between two honest agent where the secret property is successfully kept hidden from the adversary.

For our example protocol, we can claim that the two values `Ru` and `Rv` are supposed to be secret and thereby hidden from the adversary as shown below.

```
role U{
  [...]

  # Claims:
  claim(U, Secret, Ru);
  claim(U, Secret, Rv);
};
```

Such claims will obviously fail as there is no encryption used on the messages that are passed between the two roles.

SKR (Session-Key Reveal)

SKR is mostly the same as the regular secrecy claim, but is used to highlight that the term that is claimed to be secret is also a session key. This instructs Scyther to reveal the session key to the adversary after it has executed its run, hence proving that the session keys are working correctly and that current session keys are kept secret from the adversary [10]. For the SKR claim to function correctly, Scyther's session key reveal checkbox needs to be checked in the settings. If not, this claim is identical to the ordinary secrecy claim presented above.

Non-injective Synchronization

Synchronization requires that all protocol messages occur in the expected order with their expected values, and that the behaviour is equivalent to as if the protocol was executed without the presence of any adversary. An injective synchronization property states that the protocol executes as expected over *multiple* runs, claiming that it is not possible for an attacker to use information from previous runs into disrupting the current protocol execution [12]. Such an attack is known as a replay attack, and is used by an adversary to inject traffic into the protocol execution to induce undesirable or unexpected behaviour.

Scyther, however, does not support this enhanced form of synchronization, hence it is vulnerable to replay attacks [10]. Using its language, we can claim that a property holds non-injective synchronization by using the `ni-synch` statement.

```
claim(U, Nisynch, Ru);
```

Aliveness

Aliveness is considered to be the weakest form of authentication, guaranteeing to the initiator of a protocol (U) that if it completes a run successfully, then the intended responder (V) of the run has previously executed the protocol [24]. This does not necessarily mean that V knew that he was interacting with U, nor does it mean that it has executed the protocol any time recently.

Weak Agreement

Weak agreement strengthens the authentication form introduced as aliveness. Such an authentication states that the responder in fact was executing the protocol with the initiator (U), and not just having run the protocol at some point in time [24]. By claiming that the protocol holds under the weak agreement, we state that if U successfully completes a run with the intended responder (V), then V confirms that it has previously run the protocol with U. Such a claim would prevent an adversary from acting as a responder by running another run of the protocol in parallel with a run with U, and conducting a man-in-the-middle-attack. The Needham-Schroeder case presented in Chapter 2 failed on this claim, allowing Lowe to construct his attack.

Non-injective Agreement

Where the authentication provided by weak agreement does not specify which of the two communicating parties acted as initiator and responder, non-injective agreement does. It guarantees that if the initiator (U) successfully completes a run of the protocol, apparently with the responder (V), then V has completed a run with U, where he acted as a responder [24]. This does, however, not indicate that they both have executed exactly one run. There is still a possibility that U has executed multiple runs with a responder which he believed to be V, but may in fact have been communicating with the adversary. Another guarantee provided by non-injective agreement is that if U also sends a set of variables to V in the completed run, then they both agree on that the data values corresponds to all in the sent set of variables.

Below is an example of the example protocol claiming that V is in fact alive, has run the protocol at some time with U, and that during this particular run, it was U and V that was communicating.

```
role U{
```

```
  [...]

  # Claims:
  claim(U, Alive);
  claim(U, Weakagree);
  claim(U, Niagree);
};
```

Running, Commit

The set of variables from non-injective agreement can be defined using the running and commit signals. By using such a claim, we can verify that a variable sent from U to V, and then returned to U has not been changed from its initial value.

Scyther models such a claim by explicitly defining for each receive event containing an item from the set of variables. By marking the variables using a claim event of type running, the variable is marked as currently running in memory. The commit claim can then be used to state that the variables exchanged between U and V (in this case `Ru`) have not been tampered with, as seen below.

```
role U{
  [...]

  send_1(U, V, Ru);

  recv_2(V, U, Ru, Rv);
  claim(U, Running, V, Ru); # Ru running in memory

  claim(U, Commit, V, Ru); # Claim that Ru has not been
    tampered with
};


role V{
  [...]

  recv_1(U, V, Ru);
  claim(V, Running, U, Ru); # Ru running in memory

  send_2(V, U, Ru, Rv);
}
```

3.3 Scyther's Graphical Interface

As mentioned, Scyther provides a GUI for easily understand and assess the security of a protocol. If we continue on our example from the section on Scyther's syntax, we now want to verify all the stated security claims. By using the GUI, we can configure the verification process by providing a maximum number of runs, the adversary compromise model, as well as more advanced options for how to prune the search space. Scyther provides three options in its GUI; verification of claims, verification of automatic claims, and characterization of the protocol. Figure 3.1 contains the results of running a verification of the claims previously described for a secure protocol.

Claim				Status		Comments
APKES	U	APKES,U1	Secret {Ru,Rv}k(U,V)	Ok	Verified	No attacks.
		APKES,U2	Nisynch	Ok	Verified	No attacks.
		APKES,U3	Niagree	Ok	Verified	No attacks.
		APKES,U4	Alive	Ok	Verified	No attacks.
		APKES,U5	Commit V,Ru	Ok	Verified	No attacks.
	V	APKES,V1	Secret {Ru,Rv}k(U,V)	Ok	Verified	No attacks.
		APKES,V2	Nisynch	Ok	Verified	No attacks.
		APKES,V3	Niagree	Ok	Verified	No attacks.
		APKES,V4	Alive	Ok	Verified	No attacks.

Figure 3.1: Results of a verification process using Scyther.

Scyther returns an `OK` status code for each claim that is successfully verified. As we see in Figure 3.1, Scyther is not able to find any attacks on the protocol. To illustrate the case of Scyther actually finding an attack, we introduce two new claims

where `Ru` and `Rv` are claimed to be secret. In our example protocol, both nonces are sent in plaintext between U and V, hence this claim will naturally fail as seen below in Figure 3.2.

Claim				Status		Comments	Patterns
APKES	U	APKES,F1	Secret Ru	Fail	Falsified	At least 1 attack.	1 attack
		APKES,F2	Secret Rv	Fail	Falsified	At least 1 attack.	1 attack

Figure 3.2: Results of a verification process using Scyther when a claim fails.

Whenever Scyther finds an attack on a protocol, it will also provide a concrete description of the attack as graph. An example of such a graph is shown in Figure 3.3. It contains description of the different runs that Scyther executes, and shows how an adversary can pass messages a cross different runs to learn some secret information, or construct an attack.

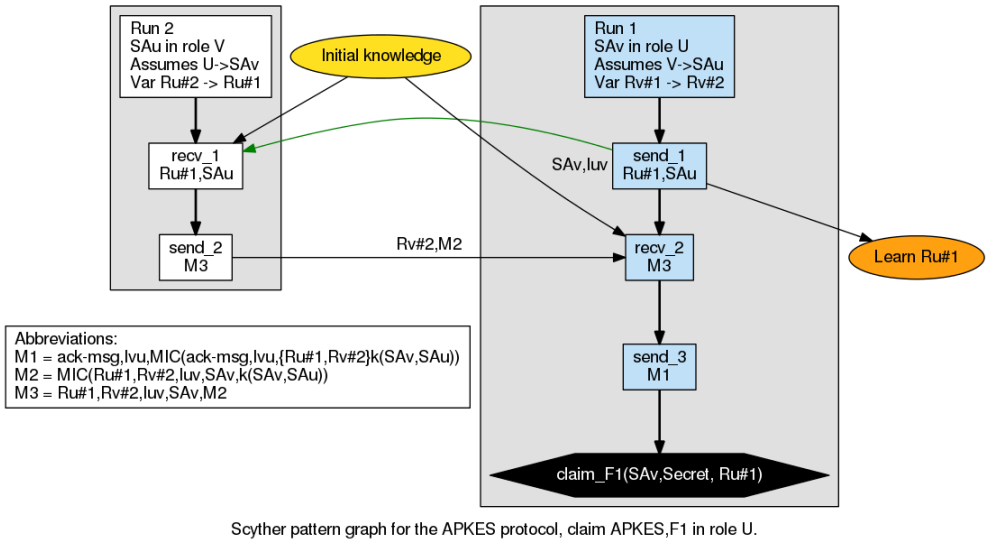

Figure 3.3: When Scyther finds an attack on a protocol, if will also provide a graph of the attack.

Chapter 4

6LoWPAN Security - Adding Compromise Resilience to the 802.15.4 Security Sublayer

APKES is a proposed protocol for handling key establishment and key management in 6LoWPAN. While currently being implemented in the operating system Contiki, which is a operating system targeted at the sensor community, it has not undergone any formal security analysis. This chapter will cover its ideas and conduct a formal security analysis using Scyther.

4.1 Ideas of the Paper (Change this)

As previously described, 6LoWPAN is a protocol stack for integrating WSNs running on 802.15.4 with IPv6 networks and allows for the nodes in the network to communicate with each other or remote hosts using only IP. APKES provides a framework for establishing pairwise keys for 802.15.4 networks using 6LoWPAN.

There exists, however, no description of how to perform key establishment.

The main idea with APKES is to provide a key establishment scheme for 6LoWPAN where the appropriate pairwise key establishment scheme is up to the developer to decide. Such an approach would enhance the overall usability for the protocol, as there is no superior scheme for handling key establishment in 802.15.4, it solely depends on the network. The plugged in scheme has only one function, which is to feed APKES with the shared secret for the communicating nodes, and APKES handles both key generation and communication with neighbours.

APKES usually consists of three phases.

When using pluggable schemes, the protocol needs the IDs

4.2 Informal Analysis

...

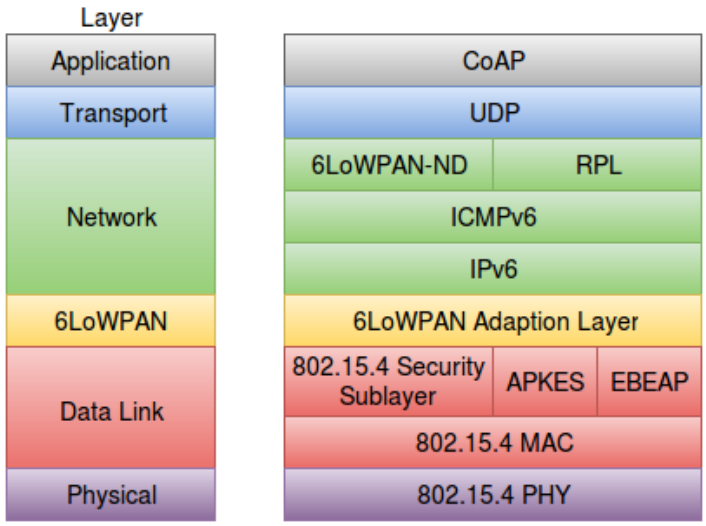

Figure 4.1: APKES is positioned in the data link layer in the 6LoWPAN stack expanding the 802.15.4 security sublayer [22].

4.3 Formal Analysis

4.3.1 Scyther

Script. Explanations. Compare.

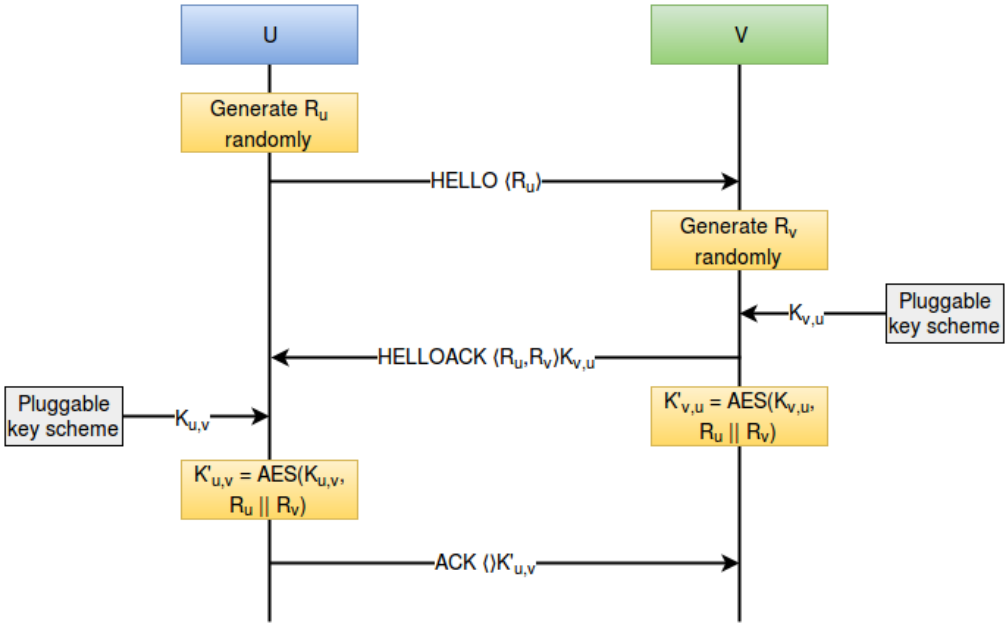

Figure 4.2: APKES consists of a three-way handshake, and utilizes a pluggable scheme for deriving the shared secret between two nodes [22].

Chapter 5

Handling Reboots and Mobility in 802.15.4 Security

AKES. Allow nodes to leave and join network. Discover new neighbours. Reboot.

References

[1] Alliance, T. Z. Zigbee. http://www.zigbee.org/. Accessed: 2016-04-15.

[2] Ashton, K. That "Internet of Things" Thing. http://www.rfidjournal.com/articles/view?4986. Accessed: 2016-03-31.

[3] Basin, D., C. J. Cremers, and C. Meadows (2012). Model Checking Security Protocols.

[4] Blanchet, B. ProVerif. http://prosecco.gforge.inria.fr/personal/bblanche/proverif/. Accessed: 2016-02-23.

[5] Burrows, M., M. Abadi, and R. M. Needham (1989). A Logic of Authentication. In *Proceedings of the Royal Society of London A: Mathematical, Physical and Engineering Sciences*, Volume 426, pp. 233–271. The Royal Society.

[6] Cremers, C. J. Scyther. https://www.cs.ox.ac.uk/people/cas.cremers/scyther/index.html. Accessed: 2016-02-23.

[7] Cremers, C. J. (2006). *Scyther: Semantics and Verification of Security Protocols*. Ph. D. thesis, Eindhoven University of Technology.

[8] Cremers, C. J. (2008a). The Scyther Tool: Verification, Falsification, and Analysis of Security Protocols. In *Computer aided verification*, pp. 414–418. Springer.

[9] Cremers, C. J. (2008b). Unbounded Verification, Falsification, and Characterization of Security Protocols by Pattern Refinement. In *Proceedings of the 15th ACM conference on Computer and communications security*, pp. 119–128. ACM.

[10] Cremers, C. J. (2014). *Scyther User Manual*.

[11] Cremers, C. J., P. Lafourcade, and P. Nadeau. Comparing State Spaces in Automatic Security Protocol Analysis. *Formal to Practical Security 5458*, 70–94.

[12] Cremers, C. J. and S. Mauw (2005). Operational Semantics of Security Protocols. In *Scenarios: Models, Transformations and Tools*, pp. 66–89. Springer.

[13] Dolev, D. and A. C. Yao (1983). On the Security of Public Key Protocols. *Information Theory, IEEE Transactions on 29*(2), 198–208.

[14] Eschenauer, L. and V. D. Gligor (2002). A key-management scheme for distributed sensor networks. In *Proceedings of the 9th ACM Conference on Computer and Communications Security*, CCS '02, pp. 41–47. ACM.

[15] Foundation, H. C. WirelessHART. http://en.hartcomm.org/main_article/wirelesshart.html. Accessed: 2016-04-15.

[16] Gartner. Gartner Says 6.4 Billion Connected "Things" Will Be in Use in 2016, Up 30 Percent From 2015. http://www.gartner.com/newsroom/id/3165317. Accessed: 2016-03-31.

[17] Gutierrez, J. A., M. Naeve, E. Callaway, M. Bourgeois, V. Mitter, and B. Heile (2001). IEEE 802.15.4: A Developing Standard for Low-power Low-cost Wireless Personal Area Networks. *network, IEEE 15*(5), 12–19.

[18] Hui, J. and P. Thubert (2011). Compression Format for IPv6 Datagrams over IEEE 802.15.4-Based Networks. *RFC6282*.

[19] Jing, Q., A. V. Vasilakos, J. Wan, J. Lu, and D. Qiu (2014). Security of the Internet of Things: Perspectives and Challenges. *Wireless Networks 20*(8), 2481–2501.

[20] Khan, R., S. U. Khan, R. Zaheer, and S. Khan (2012). Future internet: The internet of things architecture, possible applications and key challenges. In *Frontiers of Information Technology (FIT), 2012 10th International Conference on*, pp. 257–260.

[21] Kopetz, H. (2011). *Real-Time Systems: Design Principles for Distributed Embedded Applications*, Chapter Internet of Things, pp. 307–323. Springer.

[22] Krentz, K.-F., H. Rafiee, and C. Meinel (2013). 6LoWPAN Security: Adding Compromise Resilience to the 802.15.4 Security Sublayer. In *Proceedings of the International Workshop on Adaptive Security*. ACM.

[23] Lowe, G. (1996). Breaking and Fixing the Needham-Schroeder Public-key Protocol Using FDR. In *Tools and Algorithms for the Construction and Analysis of Systems*, pp. 147–166. Springer.

[24] Lowe, G. (1997). A Hierarchy of Authentication Specifications. In *Computer Security Foundations Workshop, 1997. Proceedings., 10th*, pp. 31–43. IEEE.

[25] Mao, W. and C. Boyd (1993). Towards Formal Analysis of Security Protocols. In *Computer Security Foundations Workshop VI, 1993. Proceedings*, pp. 147–158. IEEE.

[26] Mulligan, G. (2007). The 6LoWPAN Architecture. In *Proceedings of the 4th Workshop on Embedded Networked Sensors*, EmNets '07, pp. 78–82. ACM.

[27] Needham, R. M. and M. D. Schroeder (1978). Using Encryption for Authentication in Large Networks of Computers. *Commun. ACM 21*(12).

[28] of Automation, I. S. ISA100.11a. http://www.nivis.com/technology/ISA100.11a.php. Accessed: 2016-04-15.

[29] Project, A. AVISPA. http://www.avispa-project.org/. Accessed: 2016-02-23.

[30] Sastry, N. and D. Wagner (2004). Security Considerations for IEEE 802.15.4 Networks. In *Proceedings of the 3rd ACM workshop on Wireless security*, pp. 32–42. ACM.

[31] Simmonds, C. "The Internet of Things": What the Man Who Coined the Phrase Has to Say. www.theguardian.com/sustainable-business/2015/feb/27/the-internet-of-things-what-the-man-who-coined-the-phrase-has-to-say. Accessed: 2016-03-31.

[32] Technology, M. MiWi. http://www.microchip.com/wwwAppNotes/AppNotes.aspx?appnote=en536181. Accessed: 2016-04-15.

[33] Wu, M., T.-J. Lu, F.-Y. Ling, J. Sun, and H.-Y. Du (2010). Research on the architecture of internet of things. In *2010 3rd International Conference on Advanced Computer Theory and Engineering(ICACTE)*, Volume 5, pp. V5–484–V5–487.