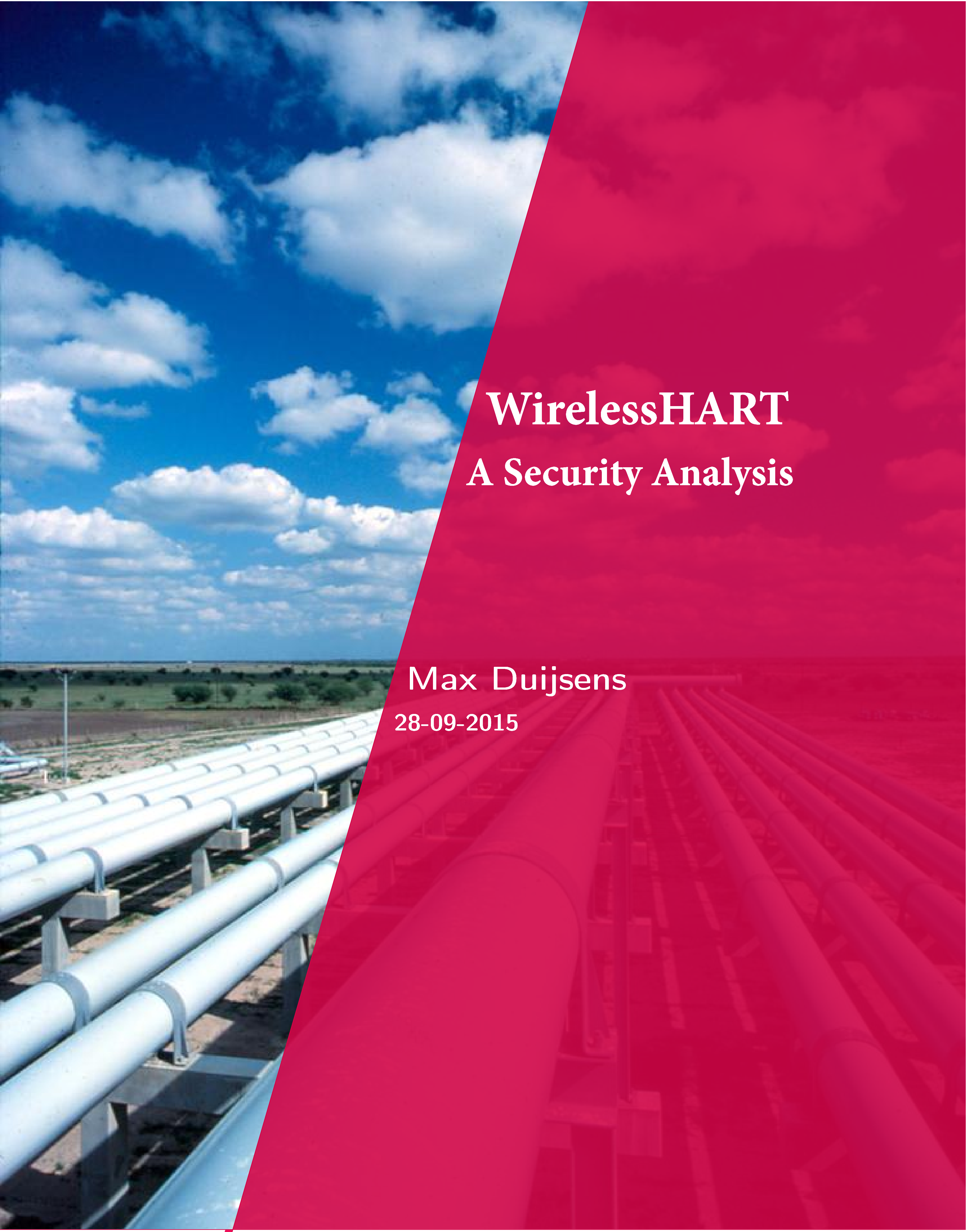


WirelessHART

A Security Analysis

Max Duijsens

28-09-2015

Technische Universiteit
Eindhoven
University of Technology

/ Department of Mathematics and Computer Science

WirelessHART
A security analysis

Master of Science Thesis

For obtaining the degree of Master of Science in Information Security Technology
at the department of Mathematics and Computer Science
of Eindhoven University of Technology.

TU/e Technische Universiteit **Eindhoven** University of Technology

Max Duijsens – 0813861
`m.f.h.duijsens@student.tue.nl`
IST

Monday 28th September, 2015

Supervisors:

B. Škorić	Eindhoven University of Technology
D. Maasland	Fox-IT
D. Niggebrugge	Fox-IT

Abstract

Network protocols play an important role in industrial automation and control. Due to the difference in security requirements between industrial control systems and well-known office environments, different protocols are used for such networks. Wireless connectivity is becoming more prevalent in these industrial environments because it reduces the total cost of ownership (wiring is expensive) and improves ease of scalability (new devices can be added easily). It is very important that industrial wireless communications are secured against packet injection and denial-of-service attacks such that the industrial process is not disturbed.

There are currently no general approaches to testing the security of these uncommon and often proprietary wireless networking protocols. In this thesis a security analysis of the WirelessHART protocol is presented. The approach presented in this thesis can also be used to analyse other wireless protocols.

The main goal of this thesis is to find vulnerabilities in the WirelessHART protocol. Two issues are addressed. First, a Software-defined radio (SDR) is used to build a generic implementation of the physical component of the protocol. This SDR is then used to capture network packets. Second, the network packets are interpreted such that a penetration tester can read and build network packets in order to perform a penetration test.

The SDR implements the physical layer of the WirelessHART protocol. This layer is based on IEEE 802.15.4 which is a protocol used in low-power wireless networks. The IEEE 802.15.4 specification defines what the components of the physical layer should be and how they work together. This layer was built using the SDR and GNURadio.

Furthermore, an implementation of the WirelessHART packet structure is built in Scapy. Scapy is a tool which transforms a byte stream (the received packet) into a human-readable format and vice-versa. Using this implementation, it is possible for a security researcher to examine and inject network traffic. Injecting network traffic using the SDR failed due to an unknown bug in the transmission component of the SDR implementation. It was possible to transmit packets correctly, however the packets do not get accepted by the gateway. Hence, a Raspberry Pi in combination with an IEEE 802.15.4 compliant radio was used. It turned out that the Raspberry Pi was too slow since the (vanilla) linux kernel with support for the IEEE 802.15.4 radio chip did not support the Raspberry Pi causing it to slow down to the point where it could no longer comply with the WirelessHART timing requirements. Therefore it was not possible to transmit packets such that they are accepted by the WirelessHART network.

Finally, using the Scapy implementation to interpret packets and the WirelessHART protocol specification, a vulnerability analysis was conducted.

The WirelessHART specification has been studied extensively in order to find vulnerabilities in the protocol. A total of 11 vulnerabilities are identified. Four of these vulnerabilities do not require the attacker to know any information about the network except information he can learn by listening to the network traffic (with the SDR). These attacks include jamming and temporary disruption attacks.

Seven attacks are identified which require the attacker to know the join key (password) of the network. If the attacker has the join key, he still can not communicate with individual devices on the network, since per-device session keys are used to encrypt network packets. One of the seven attacks is a mass de-authentication attack, triggering a join handshake for all devices in the network, which the attacker can then decrypt using the join key. The result of this attack is that he obtains the session key for all devices and can now communicate with individual devices opening the door to injecting spoofed packets and bringing down specific devices in the network.

None of these attacks could be tested, since there was no working solution to transmit network traffic such that it was accepted by the network. However, if the product to be tested is compliant with the WirelessHART standard, the attacks will work.

The mass de-authentication attack allows an attacker to shut down an entire plant network or disrupt processes if he learns the join key of the network. Therefore, care must be taken that this key is well protected and doesn't fall into the wrong hands.

Contents

Preface

After six months of hard work, the research was completed and achieved almost all goals set at the beginning of the project. It was a lot of fun to work with real industrial hardware and a software-defined radio. Figuring out how an industrial wireless networking protocol works exactly opened my eyes as to the differences between a regular IT environment and an industrial one.

I would like to take this opportunity to thank my mentors Donny and Daniel for their guidance throughout this project, you have been very helpful pointing the research in the right direction and helping out with technical details. Furthermore I would like to thank Boris for his insightful comments and useful feedback during the project and the process of writing my master thesis.

Eindhoven, Netherlands
Friday 18th September, 2015

Max Duijsens

List of Figures

List of Tables

Nomenclature

Acronym	**Definition**
TU/e	Eindhoven University of Technology
IST	Information Security Technology
ICS	Industrial Control System
SDR	Software Defined Radio
ADC	Analog to Digital Converter
DAC	Digital to Analog Converter
DSSS	Direct-sequence Spread Spectrum
FHSS	Frequency Hopping Spread Spectrum
CDMA	Code Division Multiple Access
MAC	Media Access Control
WPAN	Wireless Personal-Area Network
OSI	Open Systems Interconnection model
MIC	Message Integrity Code
O-QPSK	Offset-Quadrature Phase Shift Keying
LLC	Logical Link Control
TDMA	Time Division Multiple Access
TSMP	Time Synchronised Mesh Protocol
ASN	Absolute Slot Number

Chapter 1 Introduction

Critical infrastructure has become a core part of our modern society. We depend on the availability and reliability of water treatment plants, gas and electricity distribution networks, power plants and many more types of critical infrastructure. These infrastructures are oftentimes controlled by computer systems (called industrial control systems or ICS) to manage these complex and potentially dangerous processes. These industrial control systems make sure that motors are spinning at the correct speed, valves are opened at the correct time etc.

The CIA Triad In a regular office automation environment, the confidentiality-integrity-availability (CIA) triad [1] is often used to describe security requirements. Confidentiality of information is the most important requirement and is therefore at the top of the triad. Often, the office can continue with minor loss of availability or integrity of the computer systems, so these requirements are placed at the bottom of the triad: important, but less than the confidentiality requirement. Refer to figure 1.1 for a graphical representation. In industrial environments, the CIA triad is turned upside down. These environments impose very strict requirements on the availability and integrity of the ICS. Having to stop a plant from producing power or having to stop distribution of gas or water because an ICS needs to be rebooted is not tolerated. Chemical processes can also not be interrupted at any time for example because a chemical reaction is in progress. Minor loss of confidentiality can be accepted if the system can keep running as usual. Think of for example the temperature in a mixing barrel or the gas pressure in a gas pipeline is not critical to keep confidential. What is important is that the operator always gets those values from the sensors (availability), and that these values are accurate and displayed correctly on the operator's control station (integrity). Therefore the integrity and availability requirements are on top of the (upside down) "pyramid", while confidentiality is at the bottom.

Network Protocols Because of this difference in requirements, the network communication protocols in industrial environments are also different from regular office and consumer computer networks. Where the well known WiFi (802.11) communication protocol is built around high throughput and doesn't provide many mechanisms to prevent interference and replay attacks [2], one can imagine that in an industrial environment, throughput is less important while resistance to interference (availability) and replay attacks (integrity) is a hard requirement.

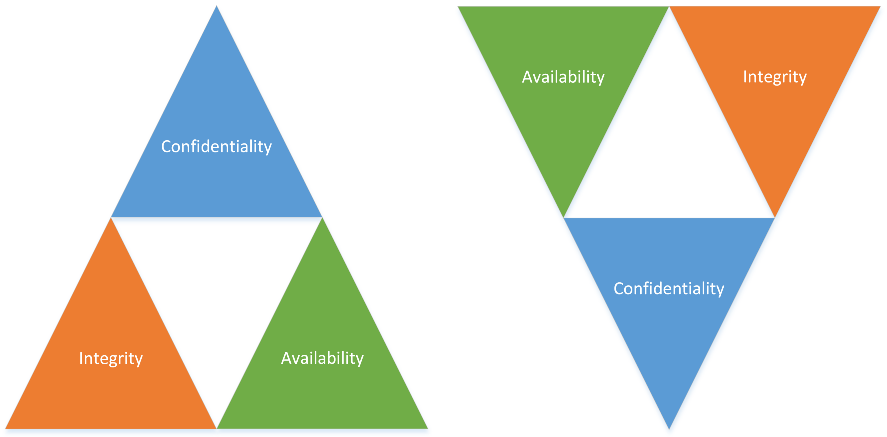


Figure 1.1: CIA Triad of an office environment on the left, industrial environment on the right.

WirelessHART An example of a wireless networking protocol used in these industrial environments is WirelessHART [3]. WirelessHART is a protocol which is developed as a multi-vendor interoperable wireless networking standard. This means that products from different vendors will be able to communicate with each other on the same WirelessHART network. It is designed with industrial automation requirements in mind, and therefore has high resistance to jamming and interference. It also provides (mandatory) mechanisms which increase the reliability of the network.

The development of the WirelessHART protocol was started in 2004 by the HART Communication Foundation. The HART Communication Foundation consists of 37 companies that developed the (wired) HART and FieldBus protocols. The companies that worked on WirelessHART are (amongst others): ABB, Emerson, Endress+Hauser, Pepperl+Fuchs and Siemens. These vendors all sell devices which communicate using WirelessHART.

The protocol was introduced to the market in September 2007. In April 2010 it received the approval from the International Electrotechnical Commission (IEC). This approval means that WirelessHART was an official wireless standard which can be used in industrial control networks as of April 2010.

A good use-case that uses WirelessHART's capabilities is the monitoring and controlling of a gas pipeline. Since in a WirelessHART network different devices can pass packets between each other it is possible to extend the wireless network over a very long range without having additional gateways or router devices in the middle. All sensors which monitor the pipeline (gas pressure, etc.) can communicate with each other via WirelessHART. Between each pair of devices, authenticity and integrity is guaranteed which means that on the entire path from source to destination these requirements can be guaranteed. Using this multiple hop network architecture, measurements from wireless connected sensors kilometres away can reach the operator, and the operator can control devices far away without having a large (usually expensive) cable bundle.

Penetration Testing Testing the security of network protocols is an important aspect of testing the security level of the computer network. By using techniques such as fuzzing, potential bugs in parsers of network packets can be discovered. When testing common network protocols like ethernet, there are many tools available to be used by a penetration tester. However, for not so common networking protocols like WirelessHART these tools are not (yet) available. Furthermore, interfacing with wireless networks is notoriously difficult for a person without extensive knowledge about the design of radio communication chips. Oftentimes with these non-common network protocols, there is only a handful of vendors that sell hardware capable of communicating with the network and most of the time this hardware is closed source. Sometimes the specification of the protocol is open or can be obtained by paying a fee. This is also the case with WirelessHART, the protocol specification can be purchased by anyone willing to pay the fee.

Software Defined Radio A good solution to this problem is to use a universal software-defined radio (SDR). An SDR consists of two components: a software component and a hardware component. The hardware component is connected to a computer, where all the signal processing is performed in a software program. This means that communication with any wireless protocol is possible without buying (closed-source) hardware for each new protocol, since the software component of the SDR can be modified to be compliant with any wireless protocol.

In this thesis, a method is presented for using a software-defined radio to communicate with the WirelessHART network. The work in this thesis, can be used to communicate with any wireless networking technology given that the user knows the exact specification of the protocol involved. Though this thesis focuses on WirelessHART, the work can be modified to be compatible with other wireless protocols as well.

An SDR is used to communicate with the WirelessHART network, after which Scapy (a python tool used to interpret and build network packets) [4] is used to interpret and build WirelessHART network packets. A wrapper is also programmed in Python to use Scapy and the SDR to mimic a legitimate WirelessHART client. A penetration tester can use this client to test the security of the WirelessHART network.

Finally, several attacks are considered. These attacks are focused around the availability and integrity requirements of the WirelessHART network, since these requirements are the most important in an industrial environment.

1.1 Research Description

1.1.1 Setting the Scene

In modern society, testing products for their security properties is critical to preventing unwanted access, modification and disruption of these products. For many of the common technologies used (like WiFi or Ethernet) there are well-defined procedures and tools available on how to test the security of these technologies. However these approaches may not work for industrial environments, since the security requirements are different.

Wireless technologies are even more difficult to test, since the interface to communicate with these networks is usually proprietary. Furthermore, wireless networks are prone to additional security risks on top of the risks already involved in common networking protocols. This lies in the fact that the attacker does not have to be visible or inside the building that hosts the wireless network: it's possible to use a powerful antenna to communicate with wireless networks from kilometres away [5]. Therefore it is extra important that these networks are well secured.

Fox-IT is a company based in the Netherlands. One of the services Fox-IT provides is Penetration Testing. During these penetration tests, a customer's network is tested to verify the penetration testers cannot gain access to confidential data. Closed-source protocol testing occurs regularly. Since there are no tools available to communicate with closed-source wireless networks like WirelessHART, Fox-IT likes to have a method to communicate with these types of networks. Furthermore Fox-IT is interested to confirm whether there are any vulnerabilities in the WirelessHART protocol.

1.1.2 Research Questions

The security testing of non-common wireless technologies is a problem for security researchers. It is not straightforward to connect to any given wireless network with any computer and start sending network packets. The reason for this is that each of the different networking technologies use their own radios and their own packet structures, etc. Therefore, three research questions are formulated that need to be answered in order to find attacks in the WirelessHART protocol:

1. *Which (generic) hardware can be used to communicate with a WirelessHART network?*
2. *How can network packets of a WirelessHART network be interpreted and generated?*
3. *Which attacks are possible in a WirelessHART network?*

The answers to these questions can also be used to find attacks in other uncommon wireless network protocols.

1.2 Scope & Demarcation

In addition to the research questions defined in paragraph 1.1.2, this paragraph will define further restrictions on the scope of this research:

This research will only investigate the WirelessHART network. While the developed software can certainly be used for testing the security of other network protocols, this is not the goal of this thesis. Other networking protocols will not be mentioned throughout this thesis except in the chapter with background information.

WirelessHART uses channel hopping to provide some resistance against jamming and interference. Since channel hopping only adds overhead in the implementation and is not crucial to the security of the protocol, channel hopping will not be implemented in this thesis.

Only attacks specific to WirelessHART will be considered. There are many well-known attacks on wireless mesh networks. For example the attacks taught in the Security and Privacy in Mobile Systems class part of the Kerckhoffs Programme [6].

Attacks requiring physical access to the WirelessHART devices are excluded. This excludes amongst others side-channel attacks and analysis of hardware components.

1.3 Project Stages

The method of research consisted of five phases. The research was conducted in 29 weeks.

Phase 1: Literature Review First, a literature review was done on wireless communication technologies, WirelessHART and how security tests are performed on networking technologies in general. Also the different software-defined radio's (SDR) available on the market at the time were evaluated. This phase took around three weeks. At the end of this stage, a specific SDR was chosen and purchased.

Phase 2: Building a WirelessHART Sniffer Second, a WirelessHART sniffer was built using an SDR. This phase took about three weeks. Reading and fully understanding the WirelessHART physical layer took the most time. However, it was well-documented how network packets are to be received in the WirelessHART protocol specification.

Phase 3: Interpreting and Crafting Network Packets Once it was possible to sniff network "packets" (which were still byte-streams at that point), these byte-streams had to be interpreted. For this process the Scapy tool was used, and the WirelessHART network packet encodings where programmed into Scapy. This process took a long time since almost the entire WirelessHART standard had to be implemented in Scapy. This phase took the most time of the entire research, about ten weeks. The reason for this long duration is that the structure of the standard is not convenient when implementing the protocol. Some chapters describe concepts required to understand other chapters, but without cross-references so it is very difficult to find those definitions and explanations when reading the chapters on the actual implementation and packet structures. Furthermore, there is room for interpretation when reading the standard and therefore simply implementing it the way it is written, leads to an implementation which might be incompatible with the WirelessHART hardware available.

Phase 4: Transmitting WirelessHART Traffic Once network traffic can be interpreted and packets can be created, it is trivial to also build network packets using the same protocol specification and Scapy implementation. However, care must be taken that all default values in Scapy are set correctly. Not all default values are documented in the WirelessHART specification (like source addresses, initial nonce value, default time-to-live, etc.). Furthermore, due to the focus on availability and non-repudiation security requirements in the design of the protocol, it proved to be very difficult to transmit packets using an SDR. The reason is that an SDR is not capable of very strict timing requirements (accurate to <2 ms). This process took another six weeks.

During this phase, another piece of hardware was tested as well: an AT86RF233 chip which can be connected to a Raspberry Pi or an Arduino. This chip is compatible with the physical layer of the WirelessHART network, but needs the Scapy implementation to build the network packets. It turned out that this chip is not fast enough to cope with the high throughput of the WirelessHART network. However, it was possible to sniff traffic using this chip and build a prototype of a tiny network sniffer, approximately 3x5cm in size. Programming the network (linux kernel) driver for the Raspberry Pi to be able to communicate with this chip took a lot of time and debugging as well as I had no previous experience programming a network driver. Building this kernel driver to communicate with this chip, took around four weeks.

Phase 5: Identify Potential Attacks The last phase was a theoretical phase. Several attacks to the WirelessHART protocol were found. However, it is noteworthy that none of these attacks are possible if the attacker does not have the join key (128bit AES key, will be explained in 4.1.3) to the network. This key is required in order to communicate with the network and is different for each network. It is the main security feature to prevent unauthorised access in the WirelessHART protocol.

Also a brute-forcer for the last 64bits of the 128bit join key was programmed during this phase. The brute-forcer uses multi-core processing in order to speed up the brute-force of the last 64bits of the key. This all rides on the assumption that the attacker can get the first part of the key using other methods such as social engineering.

This phase took about three weeks.

1.4 Thesis Outline

This thesis starts with background information required to understand the concepts discussed in this thesis in chapter 2. This chapter covers the characteristics of wireless sensor networks in section 2.1, relevant standards in section 2.2 and gives an overview of the different tools used in this project in section 2.3.

The next chapter covers the architecture of a WirelessHART network in section 3.1, followed by section 3.2 which explains how packets traverse the network. Next, the different layers of the protocol are explained in sections 3.3 and 3.4.

Chapter 4 explains how network communication was established. First, section 4.1 explains how network traffic can be captured including an overview of the security features and how these can be used to capture and interpret network traffic. Section 4.2 then builds upon this knowledge to transmit traffic.

Finally in chapter 5 attacks on the WirelessHART protocol are described in detail. These attacks are divided into attacks for which the join key is not required (section 5.2) and attacks for which the attacker needs knowledge of the join key (section 5.3).

Finally the thesis is concluded in chapter 6 and some thoughts on future work are provided.

Chapter 2
Background Information

This chapter elaborates on the concepts needed to understand the rest of this thesis. First an introduction is given to wireless communication, digital to analog conversion and different techniques employed by wireless communication protocols. Furthermore a short overview of the relevant standards is given. Finally an overview of tools used throughout this thesis is described.

Readers already familiar with wireless communication and common security tools may skip this section, while for readers not so familiar with wireless communication techniques and common security tools may find this section very useful.

2.1 Wireless Communication

2.1.1 Digital to Analog Conversion

A digital to analog converter (DAC) is an electronic component which converts a binary signal into an analog one (current, voltage or electric charge). An analog to digital converter (ADC) does the exact opposite. For converting an analog signal into a digital signal, samples are taken at a pre-defined sample-rate. This means that if the sample rate is x, the converter takes x samples per second of the source signal and converts those to the output signal. An important characteristic of a DAC or ADC is the sampling rate [7].

2.1.2 Software Defined Radio

When receiving radio signals (for example wireless network signals), these signals need to be converted to a digital signal readable by the computer that wishes to process them. This is done using analog to digital signal converters (ADC's) and vice versa. These converters then feed a component which converts the digital signal into bytes that can be read by a computer or vice versa. That process is called (de-)modulation Since each wireless protocol uses their own modulation scheme, separate hardware needs to be used for each wireless protocol. Such hardware is referred to as a network adapter. The problem for this research is that such adapters include an implementation of the protocol as well. The adapters will compute checksums, set headers with source/destination addresses, etc. These are exactly the kind of things we want to test in this thesis in order to evaluate the security of the products. A solution is to use a Software Defined Radio instead of a protocol specific network adapter.

A software defined radio (usually referred to as SDR) is a signal receiver with analog to digital converters on board. It consists of two components: a hardware and a software component. The hardware component is usually connected to a host computer via USB but could also be connected using other interfaces like ethernet or PCI-e. An SDR does not perform any de-modulation and hence cannot perform any alternations on the digital signal (at least not intentional protocol-compliant modifications). This also explains the "software defined" part of the name, since all (de-)modulation is performed in the software component on the host computer.

An example of a software package which can communicate with the hardware component of an SDR is GNURadio [8]. In GNURadio, it is possible to do all the signal processing on the host computer. This means an SDR can be used to intercept and transmit almost any kind of wireless communication.

Two popular SDR's are the bladeRF developed by Nuand [9], and the USRP B200 series developed by Ettus Research [10]. Both have similar functionality but have different specifications. For example the bladeRF is a full duplex SDR (transmit using one antenna while receiving using the other antenna). The USRP B200 supports full duplex as well as half-duplex (transmit and receive using a single antenna) modes, meaning that both these SDR's are capable of transmitting and receiving radio signals simultaneously.

2.1.3 CDMA, FHSS and DSSS

In wireless communications, both endpoints agree on using a specific frequency. There are different regulations worldwide on which frequency bands are open to the general public. However, almost everywhere frequencies in the 2.4GHz range are free to use by anyone, without requiring a license. To enable multiple networks to operate in near proximity, the 2.4GHz frequency band is divided into channels. Effectively a channel is simply a range in the frequency band (for example WiFi channel 11 spans 2.401-2.423MHz). The width of this channel is called the channel bandwidth.

Direct-sequence Spread Spectrum (DSSS) is a method to share one particular frequency channel among different devices. This is done by encoding the radio signal in such a way that it does not use the entire available bandwidth of the channel. One of the implementations (used in 802.15.4, the protocol analysed in this thesis) is CDMA (Code Division Multiple Access). In CDMA the original binary signal is XOR'ed with a device-specific pseudorandom code (see figure 2.1). This code has a higher frequency than the bit signal, which means that every bit is XOR'ed with multiple bits of the code. The result is a signal that is similar to the pseudorandom code. Given that the receiver knows the code of the sending device, it can extract the original signal by correlating the received signal with the sender's pseudorandom code. The result is that many different devices (each using a dissimilar pseudorandom code) can transmit data at the same time on the same frequency. [11]

Frequency Hopping Spread Spectrum (FHSS) is an implementation/extension of CDMA. On top of XOR'ing the bit signal with a random sequence, FHSS provides a mechanism to rapidly change channels using another pseudo-random sequence known by both the receiver and the transmitter. This allows for even more devices to share the same frequency band.

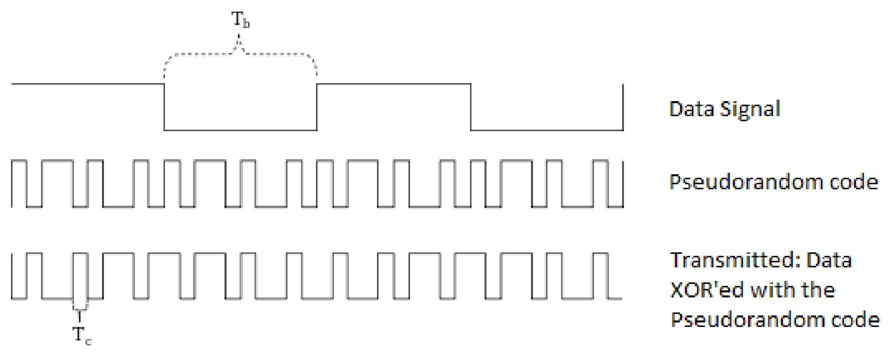


Figure 2.1: CDMA signal encoding [12]

2.1.4 Offset Quadrature Phase-Shift Keying

Offset Quadrature Phase-Shift Keying is a digital modulation scheme that modulates the phase of a reference signal (the carrier wave). It varies the phase of a sine and cosine wave in order to transmit data [13].

To transmit a bit sequence, the sequence is first grouped in pairs of two bits. Such a group is called a symbol. These symbols are then converted to a polar signal (meaning all bits equalling 0 are converted to -1, where the bits equalling 1 remain 1). The symbols are then sent to different channels, the first bit of a symbol is sent to the so called I channel, the second bit of a symbol is sent to the so called Q channel. The I channel is then multiplied by a cosine wave of a given frequency. The Q channel is multiplied with a sine wave of the same frequency (usually implemented as the same cosine wave but shifted 90 degree so it becomes a sine wave). In *Offset* QPSK, the Q channel is delayed by one bit (half symbol). As a result, the I and Q channels do not transition at the same time. This means that the transitions in the resulting signal is maximum 90 degrees.

Now we have two signals, each containing a sequence of bits. These two signals can be added together to form the resulting composite signal which can be transmitted.

2.1.5 OSI Model

The Open Systems Interconnection (OSI) Model is a conceptual model that standardises the different functions of a communication protocol or computer system [14]. This model is used on many occasions when describing networking protocols and communication systems. The model is built from different layers, where each layer fulfils a specific function.

The OSI Model has 7 layers:

- Layer 1: Physical Layer
- Layer 2: Data Link Layer
- Layer 3: Network Layer
- Layer 4: Transport Layer

- Layer 5: Session Layer
- Layer 6: Presentation Layer
- Layer 7: Application Layer

Layers 1-3 describe the communication medium. The Physical Layer consists of the byte stream and analog signals and takes care of transmission and receipt via for example a cable or over the air. The exact encoding of those bits and how to transmit them is defined in this layer. The Data Link Layer takes care of building data frames and transmitting and receiving them (via the physical layer) between two nodes. The Network Layer takes care of structuring the network packets or datagrams. It takes care of e.g. addressing, routing and traffic control.

Layer 4-7 describe the host layers. These layers usually live in the operating system kernel and not in the network adapter. They take care of building and presenting payloads to the end-user.

Upon passing a network packet through different layers upon receipt, the header and footer of that specific layer is stripped from the byte stream. E.g. the Data Link header is removed when the packet is passed to the Network layer, the Network layer processes the packet and removes the Network layer header before passing it to the Transport layer, etc. When transmitting a network packet, this process occurs in reverse order. The user or application builds a payload in the Application layer. Each subsequent layer then adds a header (and in some cases footer) to the byte stream in order to build a network packet that can be transmitted via the Physical layer.

2.2 Relevant Standards

Two standards are relevant to this research. First of all the WirelessHART standard [3] which defines the WirelessHART protocol. This standard references the IEEE 802.15.4 (version 2006) standard for the implementation of the physical layer [11]. Both these standards have been analysed and relevant parts of the standards will be summarised in this section.

2.2.1 IEEE 802.15.4 (2006)

The IEEE 802.15.4 (2006) standard defines two layers of the OSI-model: a Physical layer and a Data-Link Layer. The Data-Link layer is also referred to as the MAC (Media Access Control) layer. This MAC layer is designed specifically for operating Low-Rate Wireless Personal Area Networks (LR-WPANs). It is maintained by the IEEE working group, which released it in 2003, updated it in 2006 and 2011. It forms the basis for many different wireless mesh protocols like ZigBee, MiWi, ISA100.1A, 6LoWPAN and WirelessHART. These protocols each extend the physical and MAC layers with upper protocol layers. These upper protocol layers are not defined in the IEEE 802.15.4 standard. The IEEE 802.15.4 standard supports three frequency bands. The 2.4GHz band is used in this thesis.

Design Goals The standard aims to offer standardised networking tools to enable very low-power network communications (transceivers powered by a button or coin cell for example). The focus of this standard is on low power consumption and simple network architectures, which comes at the cost of low(er) speed. In contrast with for example WiFi, which offers much higher bandwidths compared to 802.15.4 at the cost of more power consumption and more complex hardware. The manufacturing costs of 802.15.4 hardware is very minimal (a simple transceiver costs around 10 euro to buy) due to the simplistic requirements for the physical layer [11].

Protocol Architecture The protocol defines two layers: the Physical layer and the MAC layer.

The Physical layer defines three frequency bands (868 and 915 MHz as well as 2.4 GHz). In this thesis we focus on the 2.4 GHz band since this is used by the WirelessHART protocol. The physical layer also defines how modulation should be performed. For the 2.4 GHz frequency, Offset-Quadrature Phase Shift Keying (O-QPSK) is used (see section 2.1.4). The Physical layer defines 16 channels in the 2.4 GHz band. The centre frequency of a channel can be calculated as follows [11]:

$$\forall k \in \{11 \ldots 26\} : F(k) = 2405 + 5(k - 11)\text{MHz} \tag{2.1}$$

The MAC (or Data-Link) layer of the 802.15.4 protocol defines the structure of network packets. It is important to note that this structure can be used to transmit raw data payloads over an 802.15.4 network, but in itself does not provide any security or availability guarantees to those payloads. It merely defines the routing protocol and packet structure that should be used [11].

2.2.2 WirelessHART

WirelessHART is a networking protocol which extends the 802.15.4 (version 2006) protocol. It replaces the 802.15.4 MAC layer with its own layer and furthermore defines an application layer [3]. The protocol is developed for use in industrial wireless networks, where the network payloads contain HART (a wired networking protocol) commands. The protocol uses a time synchronised, self-organising (devices can maintain their own routing table) and self-healing (when a device fails, the other devices make sure the network does not fail) network architecture. It uses only the 2.4GHz frequency band. WirelessHART packets can be transmitted using any 802.15.4-compliant radio.

More details about the technical operation of WirelessHART will be presented in Chapter 3.

2.3 Tools

This section describes the tools used in this project. Only functionality which is relevant for this thesis is presented.

2.3.1 GNURadio

GNURadio is a program which can interface with the hardware component of a software-defined radio (SDR) [8]. GNURadio is open-source and versions for Linux, Windows and OSX exist for a wide range of platforms. As explained in chapter 2.1.2, the hardware component of an SDR converts an analog signal (radio waves) into a digital signal and then conveys it to the host computer it is connected to. Upon transmission this process happens reversed. GNURadio can be used on the host computer to receive this signal. The digitised signal can then be processed using well-known digital signal processing techniques. These techniques are included in a GNURadio installation and are called *blocks*. Using these blocks, signal processing chains can be built. These chains are called *flowgraphs*. A flowgraph takes as input the bits received by the SDR, and give any type of output. As an example: it is possible to tune the SDR to a specific FM radio frequency. The input will be a signal which contains audio, modulated as an FM signal. Using blocks, a flowgraph can be built that takes as input this signal, performs the FM de-modulation (using multiple blocks chained together) and gives as output a (raw) audio stream.

2.3.2 Scapy

Scapy is a tool written in Python used to process network packets [15]. In essence its functionality is converting a byte-stream received from a network interface, into a Python object using a protocol specification. This Python object can then be modified by other programs and be converted back into a byte-stream by Scapy according to the (same) specification. These specifications define which byte in the byte stream represent which fields of the network packet. The specification is called a *layer*.

Many network protocols have been implemented in Scapy such as WiFi, TCP/IP, Bluetooth and many more. WirelessHART (or 802.15.4) is not one of them, so Scapy cannot handle this protocol out of the box [15]. There is however a very basic implementation of 802.15.4 which will be presented in section 4.1.4.

There exists an add-on for Scapy which makes it communicate (receive and transmit byte-streams) over UDP sockets instead of a physical interface. This add-on is called Scapy-Radio developed by the Defence and Space group part of Airbus [4]. By sending packets (in byte format) over network sockets, they can be received by other (local) applications such as GNURadio which can then process it further.

2.3.3 Linux-WPAN-Next

Linux-WPAN-Next is a custom Linux kernel, maintained by Alexander Aring which supports WPAN networking [16]. It contains drivers for several microchips that can receive and transmit network packets compliant with the 802.15.4 standard. Furthermore it defines how network packets should be processed by the kernel such that they can be received and transmitted to and from user-space. Promiscuous (aka. monitor or sniffer) mode is currently not supported by the kernel driver for the AT86RF233 chip (see next section).

2.3.4 AT86RF233

The AT86RF233 is a microchip developed by Atmel. [17] This chip can handle the Physical and MAC layers of the 802.15.4 standard, and can be used to receive and transmit network packets in a 802.15.4 network. It is possible to communicate with this chip over an SPI (Serial Peripheral Interface) link. The full communication protocol with the chip will not be detailed here, as it is irrelevant to this thesis. It can be found in the data sheet of the chip [17]. In this project a printed circuit board developed by OpenLabs is used [18] which can be connected to a Raspberry Pi.

A kernel driver in combination with a custom kernel that supports WPAN networking (Linux-WPAN-Next) will then be used to define how the kernel should communicate with the board and how network packets can be received and transmitted to and from user-space. The costs of the AT64RF233 PCB board (including headers to connect it to a Raspberry Pi) is 10 euro excluding shipping. For comparison, a cheap (new) WiFi adapter costs around 15 euro.

Chapter 3
WirelessHART

This chapter will elaborate on the WirelessHART and 802.15.4 protocols. The aim of this chapter is to give the reader sufficient background information about the WirelessHART protocol.

3.1 Network Architecture

In [19], an overview is given of the different components of a WirelessHART network. The network is composed of at least the following network devices:

- Network Manager
- Security Manager
- Gateway
- Nodes

The nodes can either communicate using a gateway (also known as a router in other protocols, there can be multiple gateways in a single network), or using multiple hops. A network topology using a gateway is called a star-topology. In a multiple-hop topology a node sends a packet through another node to the destination, it is called a mesh-topology. The network architecture is clearly defined in the standard to be a star topology combined with a mesh topology [3]. Devices cannot communicate directly to other devices, but have to send their packets through the gateway. However they do not need to be in direct reach of the gateway and can use mesh networking to reach the gateway. This is the default operating mode of a WirelessHART network.

Refer to figure 3.1 for an overview of how these components are connected.

Besides connecting network devices, the gateway in a WirelessHART network also acts as a bridge that connects the WirelessHART network to the plant network. Devices on the WirelessHART network can send and receive HART commands to and from the plant network. A single WirelessHART network can have many gateways. Gateways can communicate directly with each other via the WirelessHART network.

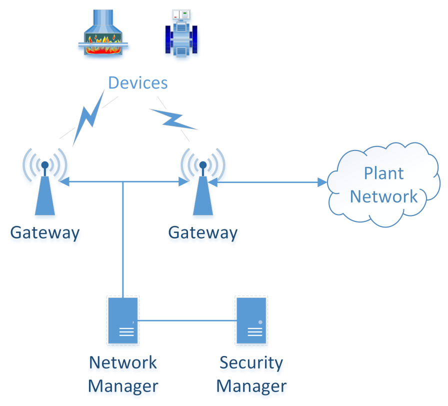


Figure 3.1: Different types of devices in a WirelessHART network

The gateway communicates directly with a network manager. The network manager is responsible for the configuration (like key distribution, routing, etc.) of the WirelessHART network. Each network can only have one network manager. Communication from a node to the network manager is always done via the gateway.

A security manager assists the network manager in the security side of the network communications. It generates session keys, join keys and network keys (this will be further detailed in section 4.1.3). Multiple networks can be managed by a single security manager, but each network has at most one security manager. The devices on the network never communicate directly to the security manager, this is done via the network manager.

Upon initialisation of the network, the network manager requests (and receives) a unique Network ID from the security manager, as well as the required keys (specifically two broadcast keys and a join key). The network manager then sets up a connection with the gateways. When a new device joins the network, it must supply the network manager with the correct network ID as well as proof that it knows the join key for that network.

A device (also referred to as node) can join the network if it is configured with a join-key and a network id. This is either a manual process, or devices are shipped with pre-configured join-keys and network id's. In the simple (default) configuration of WirelessHART, it is important that these two parameters are the same for all network devices or they will not be able to join the network.[1]

3.2 Network Operation

Before going into details how network packets are actually transmitted, this section describes the operation of a WirelessHART network at a conceptual level [3]. Please keep in mind the difference between a network manager (a device taking care of the network configuration like routing, key distribution, etc.) and a network administrator (a human which is setting up the network).

[1]Note: there are actually other modes of operating the security as well, for example using a device-specific joinkey. However since this is not default behaviour, this configuration will not be considered in this thesis.

We assume a WirelessHART network has been set up as described above. This means that there is a gateway, network manager and security manager in place and they all share two broadcast keys, a global join key and a network id. Only the network id and join key are known by the network administrator (a human usually), the other keys are stored in the network manager and cannot be extracted during normal (intended) operations.

The WirelessHART standard specifies that a gateway should transmit network advertisements (as often as possible). These advertisements contain information for the joining device on how to join the network (network id, timestamps and time slots being the most important, this will be explained in section 4.1.3). The advertisements are usually the same, but it is possible that they change over time as the network becomes more populated. This means that there are fewer time slots available for joining the network (because they are occupied by the various devices), so the advertisements are adjusted accordingly.

In order to join a new node to the network, the network administrator configures the node with the network id and the join key. The device also has a EUI-64 address, which can be compared to a MAC address in the Ethernet protocol. This address is set in the hardware of the device by the manufacturer and is not intended to be changed. The node begins by listening for advertisements on a random channel. If it receives nothing within a certain period of time (usually approximately 10 seconds) it switches to another random channel. As soon as the node receives an advertisement for the network it has been configured for, it parses the advertisement. By parsing the advertisement it learns which channels are in use by the network, and it can synchronise its clock with the network by using the timestamp contained in the advertisement and the local time (will be further detailed in section 3.3.1). After receipt of the first advertisement, the node will be able to calculate which channel will be used by the gateway to transmit the next advertisement. It calculates the next channel, switches its radio to that specific channel and listens for another advertisement. It will capture three more advertisements like this in order to synchronise its local clock with the network. Note that there is no actual clock, rather a timer which increments by 1 every 10 milliseconds. The value of this timer is called the Absolute Slot Number (ASN) and must be exactly the same across all devices on the network.

Once the ASN is synchronised, the node transmits a join request in one of the time slots specified in the advertisements. The join request is targeted towards the network manager, but the packet will have to traverse the gateway to get to the network manager. Therefore, a part of the packet required for routing (the Data-Link layer, will be further detailed in section 4.1.1) is encrypted using a so called well-known key. This key is public and is documented in the WirelessHART standard. The payload (the Network layer, will be further detailed in section 4.1.1) is encrypted using the join key as configured by the network administrator. The join request contains a nonce and an encrypted payload with information about the node (e.g. battery power level).

Upon receipt of the join request, the network manager can validate that this is a valid join request if it is possible to decrypt the packet with this network's join key. Furthermore some parts of the packet are "authenticated" using a message integrity code, which is calculated using the join key. This proves to the network manager that this device has knowledge of the join key and is therefore allowed to join the WirelessHART network. In the default configuration of a WirelessHART network, checking for knowledge of the join key is the main mechanism for checking whether a device is allowed to join the network. There are other configuration possibilities like physical address filtering/whitelisting, but these features are not documented in the WirelessHART standard. Therefore these alternate features for further restricting access to the WirelessHART network will not be considered in this thesis.

After the join request, a handshake will be performed between the network manager and the node. They will first exchange a session key and all further communications between the network manager and this specific node are encrypted using this session key. In this first message exchange, the network manager assigns a nickname to the joining device. This nickname can be compared to an IP address in an Ethernet network and is 2 bytes long. Subsequent communication addressed to and from the device will use this nickname instead of the EUI-64 address. Furthermore a session key between the node and the gateway as well as the broadcast key will be sent to the node (which stores them). Finally, routing information is exchanged defining in which time slots this node is allowed to transmit and at which time slots it is supposed to listen for incoming traffic.

After the handshake, the node is successfully joined to the network and it can receive and transmit packets.

For a graphical overview of the handshake see figure 3.2. The following notations are used:

- $\{m\}_{key}$ is used to denote a symmetric encryption of m using key;
- $\#(m)_{key}$ is used to denote a message integrity code over m using key;
- "hdrs" denotes the headers of the network packet;
- $_{jk}$ is used to denote the join key;
- $_{wk}$ is used to denote the public well-known key.

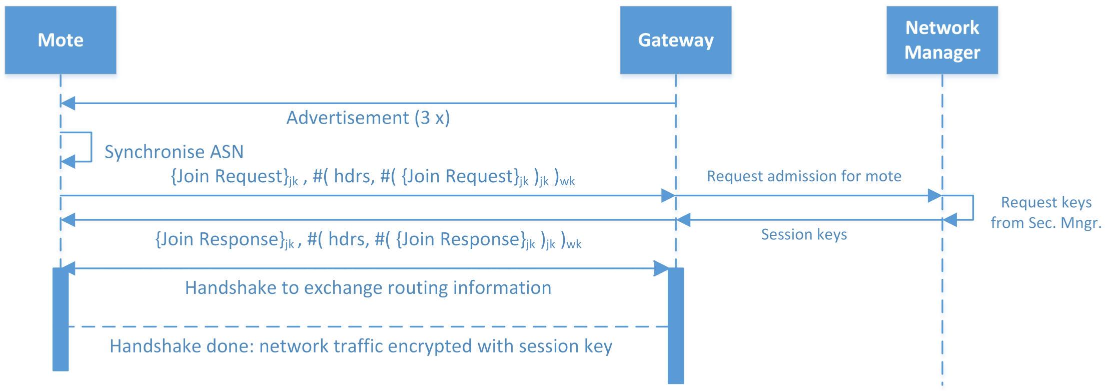

Mote
Gateway
Network Manager
Advertisement (3 x)
Synchronise ASN
{Join Request}jk , #(hdrs, #({Join Request}jk)jk)wk
Request admission for mote
Request keys from Sec. Mngr.
Session keys
{Join Response}jk , #(hdrs, #({Join Response}jk)jk)wk
Handshake to exchange routing information
Handshake done: network traffic encrypted with session key

Figure 3.2: The message sequence in a WirelessHART join handshake

3.3 Physical and Data-Link Layer

This section further details on how WirelessHART devices transmit and receive packets.

The WirelessHART protocol's physical layer is based on IEEE 802.15.4 (2006) [3]. It uses O-QPSK (Offset Quadrature Phase Shift Keying, see section 2.1.4) and operates in the 2.4GHz band with a data rate up to 250kbps. To resist interference and jamming, DSSS in combination with FHSS is used. This is also known as channel hopping. The radio hops over multiple frequency bands (channels) using a pseudorandom sequence [19][3]. There are 15 channels (out of 16) used in the 2.4GHz band for channel hopping. The remaining channel (26) is not used, since it requires a license to use in some countries. The other channels are part of the ISM band, meaning that these channels are free to use all over the world.

The physical layer encapsulates the Data-Link layer. The Data-Link layer takes care of routing between two devices that can directly communicate with each other. These devices do not have to be the actual sender and final destination of the packet, since a packet can travel multiple hops from the sender before reaching the final destination. The data-link layer consists of two sub-layers: the MAC (Medium Access Control) layer and the LLC (Logical Link Control) layer. The MAC layer takes care of the actual routing of packets while the LLC layer takes care of priority, flow control and error detection. This means that the MAC layer is below the LLC layer, but both are inside the Data-Link layer of the OSI model.

The MAC layer uses TDMA (Time Division Multiple Access) for link scheduling. It uses time slots, where each time slot has a duration of 10 ms. This time slot duration has been reverse-engineered from sniffing packets on a running network, it does not appear to be documented. This time slot provides enough time for a single data unit including acknowledgement to be sent and received.

TDMA uses superframes to perform the actual link scheduling [3]. A superframe is a series of time slots. The size of this superframe may vary between networks, but must be the same across all devices in a single network. A superframe can be visualised as a series of 'slots' in which data can be transmitted or received. Each of the slots in the superframe has a number, which is based on the Absolute Slot Number (ASN) which can be seen as a global timestamp in the network. The slot number of a slot in a specific superframe can be calculated as follows:

$$\text{slotnr} = \text{ASN mod superframe\_size} \tag{3.1}$$

Where the superframe_size is the total number of slots in this superframe. Refer to figure 3.3 for an overview of what a superframe looks like. So if a device wants to transmit a packet at a given time (ASN), it can calculate whether it is allowed to do that by calculating the slot number in the superframe. As an example take a superframe of size 1024 (it has 1024 time slots), the slot number can be calculated as in equation 3.1 and checked against the slots designated for this node during the handshake. The timestamp when the next packet can be transmitted can also be calculated using the knowledge that the ASN increments by 1 every 10 ms. A device can calculate the time until the next allocated transmission time slot before sending the packet using the following formula:

$$\text{wait} = (\text{current\_ASN - next\_transmission\_slot mod sf\_size}) * 10\text{ms} \tag{3.2}$$

After calculating how long the device has to wait before the next transmission slot arrives, it simply sleeps for this many milliseconds before transmitting the packet. The receiver can follow this same process and sleep until the next time slot designated for receiving.

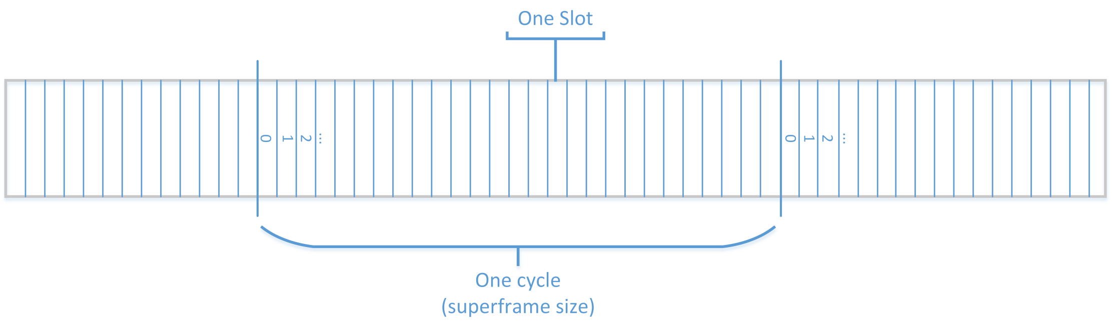


Figure 3.3: Structure of a superframe

It is clear that when handling these strict timing requirements, it is very important that the ASN counter is the same on all devices. Otherwise one device may be transmitting while another is not listening yet. More details on how the ASN is synchronised will be explained in section 3.3.1.

The connection between two devices is called a link. The link is set up using the superframe, neighbour id and time slot (ASN) [3]. All the devices in the network share a list of channels which can be used (identical across all devices), which is set by the network manager. Furthermore each device maintains a list of links it has set up with other devices.

3.3.1 Time Synchronisation

In order for two nodes to communicate successfully over a link, their transmission and reception should be timed correctly in the time slots defined in the superframe. This requires synchronisation of clocks between every two nodes in the network. While the standard states that time synchronisation is important, it does not provide any mechanism to achieve it. Since the Linear WirelessHART Development Kit [20] which was available for this project is closed source, it was not possible to find the time synchronisation mechanism immediately. However, the micro controller which provides the WirelessHART stack in these devices is developed by Dust Networks. The founder of Dust Networks, published a paper in 2008 which describes a time synchronisation protocol called TSMP (Time Synchronised Mesh Protocol) [21]. This was two years before the WirelessHART standard was published but the similarities between the two protocols are very big. Furthermore, on the Wikipedia page of WirelessHART there is a statement that the WirelessHART protocol is based on Dust Networks' TSMP protocol. Although it cannot be said with absolute certainty that this is the time synchronisation protocol implemented in the Dust Networks chip, it is very likely that it is. Using techniques described in chapter 4.1, it was confirmed that the TSMP protocol is actually used to synchronise network time in Dust Networks' WirelessHART chips.

As an illustration of how time synchronisation works in the TSMP protocol we give the following example: Assume there are two nodes A and B. A and B have a link already set up between them (which means their clocks are already somewhat in sync), and A transmits a packet to B over this link. A transmits the first bit of the packet as close to the starting time as possible in (one of) the link's time slot(s). Upon receipt, B will take note of the local time of the receipt of the first bit. Let's call this local timestamp b_1. Since every data packet which is a unicast (1 sender to 1 receiver) must be acknowledged by the receiver in TSMP, B will send an ACK (acknowledgement of reception) packet back to A. Before transmitting the ACK packet, B calculates the difference between the start of the time slot and b_1. This difference is then included in the ACK packet sent to A. This way A knows how much the difference between the transmission and receipt of the first bit of the original packet was, and can adjust the local clock accordingly. This creates the concept of time parent (B is the parent in this case)[21].

This describes how synchronisation works if the devices are already somewhat in sync. However, initial synchronisation is needed. This is achieved using network advertisements which contain the actual absolute slot number (ASN). The joining node will listen for network advertisements. Once it receives an advertisement, it records the local time of the receipt of the first bit of the packet. It will record the ASN from the advertisement together with this local timestamp. This process is performed three times (three advertisements are parsed), after which the node is able to estimate the network's ASN by calculating the difference between the timestamps at which the advertisements are received and the difference in the ASN's that where listed in those advertisements.

The joining node will transmit a keep-alive packet (used to announce its presence) in a time slot dedicated to joining devices (which it learns from the received advertisements as well). The gateway will respond with an ACK containing the time offset allowing for the joining node to synchronise its clock with the gateway.

3.3.2 Link Channel Calculation

When two devices in the network want to communicate with each other, they have to negotiate (or be assigned) several parameters (like time slots, each other's addresses, etc.) required to communicate with each other and route packets on the WirelessHART network. These parameters define how two devices communicate and is referred to as a link.

Each link is assigned a channel offset by the network manager. This offset is an integer, which is used to calculate the actual channel (frequency) a transmission will take place on [3].

When a node wants to send a packet onto the WirelessHART network, it has to calculate the channel on which it can transmit in the next time slot available for transmission. This is done as follows:
Each device keeps an array of enabled channels. This list is defined by the network manager upon initialisation of the network (or manually configured), and is conveyed via the network advertisements. This list is an ordered array containing the channel numbers active for the network. (For example, if channels 11 and 12 are enabled the array would contain A[0] = '11', A[1] = '12').
Upon transmission, the sending device chooses a link from the (local) link list that can reach the destination and calculates the channel to transmit as follows [22]:

$$\text{index} = (\text{Channel Offset} + \text{Absolute Slot Number}) \mod \text{Nr of Active Channels} \tag{3.3}$$

The channel to transmit on is then $A[index]$. Note that Nr of Active Channels equals the size of A, and the channel offset is negotiated during the establishment of the link. This ensures that each slot number has a different channel. Since modulus is used over a predictable number, the sequence of channels can be pre-computed given that the slot numbers are known by the sender and receiver.

Transmission is then initiated at the designated time slot. This process repeats at the next designated time slot, which may be in the same superframe or in a consecutive one.

In case of a collision at the receiving device (two senders send a packet in the same time slot), the receiver will not send out an acknowledgement of receipt. The sender uses a random back-off mechanism and waits for the next opportunity for transmission. How this mechanism works is not relevant for this thesis.

3.4 Network and Higher Layers

The payload of the Data-Link layer is called the Network layer. The Network layer takes care of routing from source to destination, so it includes the source and destination address amongst other fields that are relevant for routing a packet from the source to its destination. More details on the actual fields will be given in section 4.1.1.

The payload of the Network layer is called the Transport layer. This layer has only a single header field used to indicate the payload type. The payload of the Transport layer is the Application layer which are the actual HART commands.

3.5 Capturing WirelessHART Packets

The official method to perform an assessment of a WirelessHART network is to use the WiAnalys toolkit distributed by the HART foundation. The problem is that this toolkit is only available to members of the HART foundation (the vendors that sell WirelessHART equipment). It is a sniffer capable of capturing packets on all 15 channels simultaneously and storing the packets for later analysis using a software package. This toolkit is not available to end-users and security testers, so a different tool is needed.

In [23] such a tool is developed. The authors use 15 IEEE802.16.4 receivers (Freescale MC13192) interconnected using an FPGA development board (Altera Stratix-II). Packets captured by the 15 radio receivers are forwarded to the FPGA, which wraps the WirelessHART packets in the ethernet protocol. It then transmits these ethernet frames to a computer via a regular ethernet connection. The computer can then analyse the WirelessHART packets using a packet analyser.

However, the code used on the FPGA is not published. Furthermore, their set-up requires the engineering of a circuit-board which is a process that takes too long for a penetration tester without knowledge of electrical engineering. Therefore, a less complex solution is needed.

Chapter 4 Communicating with the Network

This chapter describes the setup (and problems encountered) in order to communicate with the network using a "home-made" WirelessHART-capable device. First, a sniffer was built. Part of building this sniffer is analysing how a legitimate device receives traffic, which checks are performed and when a packet is considered valid. Once this is known, a sniffer can be built. The next section explains how this knowledge can be expanded in order to transmit network packets.

4.1 Sniffing Network Traffic

This section describes how network traffic of the WirelessHART network was captured. All of the work presented in this chapter has been developed or reverse-engineered by the student except when noted otherwise. First the structure of WirelessHART network packets will be detailed, after which the actual implementation of the network sniffer will be explained.

4.1.1 WirelessHART Packet Structure

Figure 4.1 gives an overview of the structure of WirelessHART network packets [3]. The top layer is the Data-Link layer. The second layer is the Network layer and the bottom layer reflects the Transport layer which consists of a single header field (TL Control Byte) and the payload. The payload of the Transport layer is the actual HART command, which is also called the Application layer.

The Physical Layer The physical layer of the packet consists of a physical header. This header starts with 4 null-bytes signalling the receiving device that a packet is coming (the receiver can detect energy on the specific frequency). The null-padding is followed by a single byte 0x*A7* (10100111 in binary), which allows the receiver to synchronise his clock rate to the sender's [3]. The physical layer defines 15 channels to be used by WirelessHART.

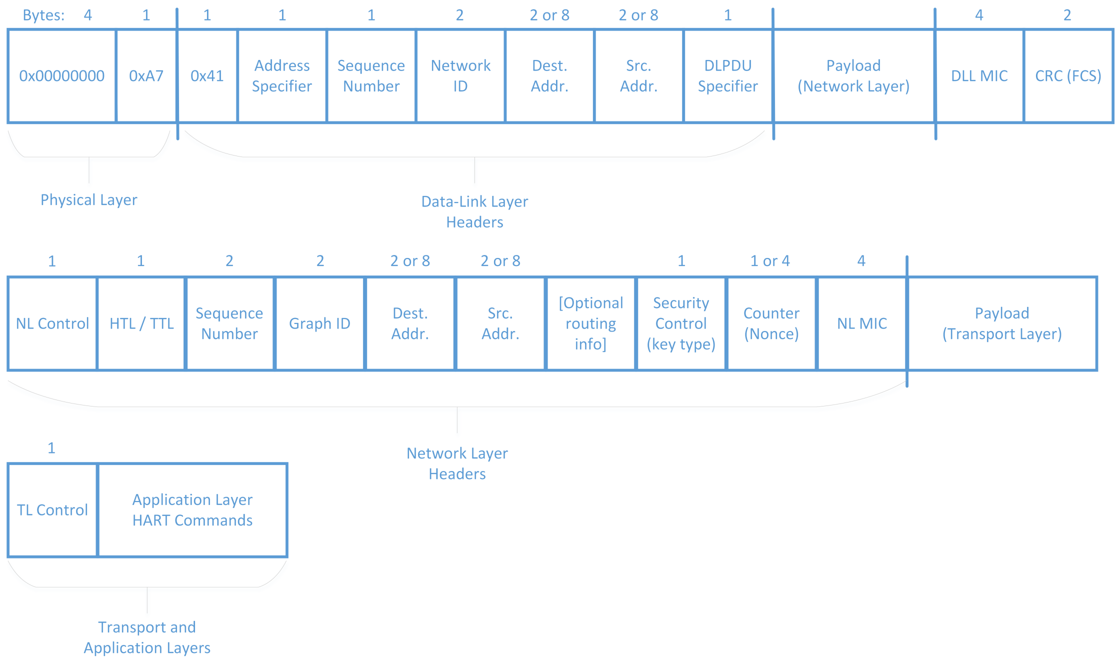


Figure 4.1: WirelessHART network packet structure

The Data-Link Layer The Data-Link layer defines a header and a footer. The header starts with a static byte 0x41 which indicates the data mode and security mode of the IEEE 802.15.4-2006 standard are enabled. This byte is always static in a WirelessHART network because these fields are not used, and by setting the rest of the bits to 0 allows for any 802.15.4 compatible radio to receive WirelessHART packets [3]. In the header there are also fields which indicate routing information. The network id, destination and source addresses indicate whether the receiver has to further parse this packet. The Data-Link layer parser drops the packet if either one of these fields does not match its configuration. For example the packet is not destined for the network the parser is connected to or the destination address at the Data-Link layer does not match the parser's address.

The addresses at the Data-Link layer indicate the source and destination of this "hop". For example a packet destined to the network manager (which has an address of 0xf980), will have to traverse the gateway first. The sender of such a packet would set the Data-Link layer destination address to 0x0001 (the gateway), and the Network layer destination address to 0xf980 (see next paragraph). This means that in a multi-hop configuration these Data-Link layer fields are changed upon each hop.

Furthermore a sequence number is included. This sequence number is the least-significant octet of the absolute slot number (ASN) in which this packet was transmitted. The parser can check whether this packet was actually transmitted in the specific ASN or whether the packet is "old". Receipt of an old packet can happen for example when a packet echoes in a room or in the case of a replay attack.

In the footer of the packet, the Data-Link layer contains a MIC and a CRC16 checksum (also referred to as a Frame Checksum or FCS). This CRC checksum is calculated via a "regular" ITU-T CRC16 polynomial [11]. This CRC checksum is calculated over the entire packet starting from the first byte of the Data-Link layer header up to and including the 16 bit CRC checksum field. A property of CRC16 calculations is that if the CRC checksum is in place, and the CRC is calculated over the same data again, the CRC field will end up being 0. This is the way a receiver could check whether the CRC is valid: calculate the CRC over the entire packet and check whether the outcome of this calculation equals 0.

More details on the MIC calculation can be found in paragraph 4.1.3.

The payload of the Data-Link layer is the Network layer.

The Network Layer The Network layer of the packet defines a header. This header contains fields which mainly relate to the security features of the protocol. A time-to-live or hop-to-live (TTL/HTL) field is included, which makes sure that packets can only be forwarded at most 128 times. This counter is decremented upon receipt and checked whether it is non-zero. If the TTL reaches zero, the packet is dropped. It also includes a sequence number which is different from the Data-Link layer sequence number. The Network layer sequence number is again the least significant octet of the absolute slot number (ASN), but it does not have to be equal to the time slot the packet was received in. This field is filled as soon as the network layer receives a transmit request, so it is a coarse way to measure how long ago this packet was created. Furthermore the source and destination address are included. The source and destination addresses of the network layer are the actual first source address and final destination address of the packet. So in a multi-hop route, these addresses remain the same when traversing other devices.

There are also a few fields containing meta-data for the AES encryption. The Security control field determines the key type used to encrypt the payload (0x00, 0x01 and 0x02 mean session key, join key and handheld key respectively). The handheld key was not used by the WirelessHART product tested. There is also a counter which is either the least significant 8 bits of the nonce used to encrypt and sign the packet when the session key is used, or it is the full 32 bits of the nonce when the join or handheld keys are used.

As last header field, a Message integrity code (MIC) is included. This MIC is calculated over the full Network layer of the packet using the source address and nonce.

The payload of the Network layer is encrypted using AES-128 in CCM mode using the meta-data from the headers (nonce and key type). More details on how the payload is encrypted and signed can be found in paragraph 4.1.3.

The Transport Layer The Transport layer of the packet contains a single header byte. The first three bits of this control byte indicate whether the payload is supposed to be acknowledged by the destination, whether it is a response to a previous request and whether it is a broadcast payload. Furthermore the last 5 bits of the control byte indicate the Transport layer sequence number. This sequence number is only used to detect duplicate and missing commands and to correlate a response with a request.

The payload of the Transport layer contains the actual HART commands (wrapped in another layer, which is not relevant for this thesis).

4.1.2 Reverse-engineering how a Legitimate Device Receives Packets

In the Linear WirelessHART Development Kit, a chip by Dust Networks is used [20] which contains the WirelessHART implementation. This Dust Networks chip handles the Physical and Data-Link layers of the protocol. It forwards only the Network (and higher) layer of the protocol to the kernel of the device. The Network layer is then parsed by the kernel. This behaviour is not documented in public documentation and has been reverse engineered by using a packet sniffer which will be detailed in section 4.1.4.

Since the Dust chip does not provide any debugging methods (at least not publicly available), it was impossible to debug the Data Link layer of the protocol. Therefore a sniffer had to be built in order to find out how the Data Link layer of the protocol operates.

4.1.3 WirelessHART Security Features

WirelessHART has multiple security features which allow it to guarantee confidentiality, integrity and authenticity. In this section, the security features of the WirelessHART protocol will be explained per network layer.

Key types

There are seven different key types used in the WirelessHART network [3]. All keys are 128 bit AES keys.

At the Data-Link layer, a well-known key as well as a network key is used. The well-known key is equal to: 7777 772E 6861 7274 636F 6D6D 2E6F 7267 in hexadecimal notation. This key is specified in the standard. It is specified to be "randomly chosen", but when decoded into ASCII it reads: "www.hartcomm.org" so it is not so random after all. Furthermore, specifying the key in the standard makes it public. Therefore, using the well-known key provides no security guarantees at all since everyone has this key.

The network key is a key which is generated by the security manager upon network initialisation. It is distributed to the network devices during the join handshake. This key is random and unique for each initialisation of a network and is therefore not known by an attacker. It is used to verify whether a packet transmitted on a specific network is valid for that specific network.

Both the well-known key as the network key must be the same across all devices successfully joined in the same network. The well-known key is hard-coded because the key is specified in the standard, the network key will have to be conveyed to the network devices during the join handshake.

In the Network layer there are five different keys, see table 4.1. There is a unicast session key between the device and the network manager, a unicast key between the device and the gateway and two broadcast keys: one for communication originating from the network manager and one for communication originating from the gateway. Finally there is a join key.

The join key is a globally set key, which can be changed by the network administrator (human). This is the only key which is directly changeable via the maintenance port (also known as a configuration interface like SSH, telnet, etc.) on the network devices.

The unicast keys are used for one-on-one communication. Devices do not have peer-to-peer unicast keys by default so all communication has to traverse the gateway in order to reach the destination.

The broadcast keys are used to transmit broadcast commands from the network manager or gateway to all devices. These four keys are conveyed to the joining device during the handshake.

Table 4.1: Network Layer Key Types

Key	Key Type	Used for
Unicast Network Manager	Unique for each device	One-on-one communication between the network manager and the device
Unicast Gateway	Unique for each device	One-on-one communication between the gateway and the device
Broadcast Network Manager	Equal for all devices	Broadcast packets to all devices from the Network Manager
Broadcast Gateway	Equal for all devices	Broadcast packets to all devices at once from the Gateway
Join Key	Equal for all devices, preconfigured by network administrator	Joining the network. It must be the same across all devices and preconfigured by the network administrator.

Nonces

Both the Data-Link and Network layers of the protocol include Message Integrity Codes (MICs). The algorithm used for calculating these MICs is AES-128 in CCM mode. This AES mode expects a nonce to be used in order to compute the MIC.

In the Data-Link layer, the nonce is constructed from the ASN (which is not transmitted in the packet, but can be deduced if the receiver is in sync with the network) and the sender's address (from here on: source address). The source address is either the full EUI-64 address (for a join request packet), or the nickname of the device zero-padded. This means the length of the source address or nickname field is always 8 bytes. The size of the ASN is 5 bytes meaning the Data-Link layer nonce is 13 bytes in total.

The nonce used for calculating the MIC in the Network layer consists of the following: The first byte of the nonce is set to 1 for join responses, 0 for all other packets. The next three bytes shall be the three most significant bytes of the counter for that specific device. The next byte will be the counter field as it is transmitted in the Network layer of the packet. The last 8 bytes of the nonce will be the source address (either the EUI-64 address or zero-padded nickname). Therefore the nonce has a total length of 13 bytes.

Data-Link layer

The first layer of the packet (after the Physical layer), the Data-Link layer contains a checksum calculated over the entire frame using CRC-16, as well as a Message Integrity Code (MIC) [3]. The CRC checksum provides integrity of the network packet. The packet will be discarded instantly without doing any further checks if the CRC fails.

Furthermore the Data-Link layer contains a Message Integrity Code (MIC). The MIC is calculated using either the Well-Known key or the network key[1]. The well-known key is used for PDU's (packets) that have to be parsed by devices which are not yet joined in the network (for example advertisements), or which are in the process of de-authenticating from the network. For all other packets, the network key is used.

The MIC is calculated using a nonce which is constructed as described in the previous paragraph.

When a packet traverses the network, all devices in the network will be able to validate this MIC since the network key is known by all devices which are part of the network. According to the specification, this provides authenticity. But since every device in the network knows this key, it merely shows that this packet was transmitted by a device which has successfully joined the network. The only authenticity provided by this key is that this packet is valid for this network since devices not part of the network do not know this key.

The Network Layer

The second layer of the packet, the Network layer contains another MIC. This MIC is calculated using either the join key for devices that are in the process of joining the network, or using the applicable session key for all other packet types (see table 4.1). The MIC is not calculated over the entire packet. Specifically the TTL, MIC and counter fields are set to 0x00 when calculating the MIC. These fields are filled later, before handing over the packet to the Data-Link layer for transmission.

According to the specification, the Network layer MIC provides proof of authenticity to the receiver. However, since any device which knows the join-key can intercept the joining handshake, all devices on the network may know each other's session keys (even though that is not part of the specification and not part of the intended behaviour). So this MIC actually provides no more authenticity than the MIC already present in the Data-Link layer and is therefore superfluous.

The payload of the Network layer is encrypted using AES-128 CCM, using the same nonce as the one used for calculating the MIC (see section 4.1.3). The key used to encrypt the payload depends on the packet type: for join requests the join key is used, for all other packets the corresponding session key is used. This does not provide complete confidentiality, since any device that knows the join-key can intercept the session key of another device and therefore decrypt payloads sent by these other devices. Although this is not intended behaviour (each device should only listen to packets destined for itself), this is definitely a possible attack vector.

Other layers of the protocol do not have any additional security features and rely on the authentication and confidentiality provided by the Data-Link and Network layers of the protocol.

[1]Note: When calculating the MIC using the well-known key, it does not provide any authenticity guarantees since this key is publicly known

Authentication

Authentication of the packet is provided by checking whether the nonce can be successfully reconstructed and the received packet is not a duplicate of a previously received packet (the nonce is fresh). If the nonce is not fresh, the packet will be dropped. Otherwise the MIC will be checked. If the MIC check fails, the packet will be discarded.

This is implemented by each device keeping a table of devices it has received packets from. The last nonce used in those packets is stored in a table. For the next packet, the nonce must be higher than the one stored in the table. If this is the case we speak of a fresh nonce.

Note that this process provides authentication (via the MIC) but also replay protection (via the nonces).

4.1.4 Building the Sniffer

This section describes the process of building a WirelessHART sniffer and how packets can be interpreted.

Choosing a Software Defined Radio

In order to capture network packets, a Software-Defined Radio (SDR) was used in combination with GNURadio. For this project a bladeRF x115 (see section: 2.1.2) was used. The bladeRF was chosen since it can transmit and receive signals in full-duplex mode (at the same time) and has a larger FPGA than other devices on the market. Another SDR which was investigated is the USRP B210 [10]. This SDR is comparable to the bladeRF but has slightly different specifications. The USRP supports a wider band of frequencies (50 MHz - 6 GHz) than the bladeRF (300 MHz - 3,8 GHz). The USRP also supports a higher sampling rate (61 Msps vs. 40 Msps in the bladeRF). These differences also translate to the pricing of the devices. The bladeRF is priced at $650 where the USRP costs $1100 when buying from the manufacturer. The bladeRF has sufficient capabilities to receive and transmit WirelessHART packets, and it can also be repurposed to analyse other wireless protocols in the 2.4GHz band as well as in the 950 MHz band after this project is finished. Furthermore, with a sampling rate of 40 Msps it should be possible to capture several channels at the same time (to defeat channel hopping). Therefore it was decided to order a bladeRF x115 for this project.

Fox-IT ordered a USRP B210 at a later stage (for a different project), which was later compared to the bladeRF. Both devices are suited to communicate with the WirelessHART network.

Building the Physical Layer

The handling of the Physical layer was done in GNURadio (see Figure 4.2). The Physical layer consists of two parts: hardware and software. The hardware used is the hardware component of the Software-Defined Radio (the bladeRF x115) which communicates with GNURadio, the software component which is running on a laptop or virtual machine. The bladeRF sends samples of a radio signal over a USB3.0 connection to GNURadio. GNURadio should then de-modulate this signal and extract bytes from that de-modulated signal. In order to accomplish this, an implementation of the (de-)modulation scheme (called a flowgraph) needs to be built in GNURadio. For this the

IEEE 802.15.4 standard was used which gives a detailed description on the design of radio chips able to communicate with the network [11].

In 2008, Choong et al. developed a GNURadio module which is able to de-modulate an 802.15.4 signal and send it to a packet sink[24][25]. A packet sink is a block in GNURadio that takes a byte stream and chops it into network packets that can be forwarded to other blocks so it can be saved in a file or printed in the terminal. Since WirelessHART uses 802.15.4 as its physical layer, it should be possible to use this module for capturing “raw” 802.15.4 packets. When developing the flowgraph it turned out the demodulating flowgraph developed by Choong et al. [24] was not usable for this project since the packet sink was only compatible with Wireshark. Since Wireshark cannot be used to transmit packets (will be explained in section 4.1.4), a different tool for analysing the WirelessHART network packets is used in this project. Furthermore the flowgraph was much too elaborate which was not needed for this project (since its executed in a lab environment and doesn’t need any signal filtering, etc.). Therefore a new flowgraph was created which would provide only the features absolutely necessary.

Hence, a flowgraph was developed in GNURadio which demodulates the O-QPSK (Offset-Quadrature Phase Shift Keying) modulation which is used by the 802.15.4-2006 standard. This was accomplished using a Quadrature Demod. block with a gain setting of 1. A gain of 1 is chosen because we are in a lab setting without much interference and the devices are very near to each other. After the demodulation, a single-pole IIR filter was used to clean up the demodulated signal. Then, a Clock Recovery block was used to recover the actual bits from the demodulated signal. These bits were fed into a Packet Sink, which chops the byte stream into packets and strips off the physical header (synchronisation pre-amble and a "start of packet" indicator and the total packet length). This results in a stream of network packets properly formatted to be interpreted by any network traffic analyser software.

WirelessHART uses channel hopping (see section 3.3). Since channel hopping makes it more difficult to process the signals, it was decided to disable the channel hopping feature so the rest of the transceiver could be built first. Channel hopping could be added later, for example using the process described in another one of Choong’s papers [26].

Choosing a Tool to Interpret Network Packets

The output of the GNURadio flowgraph is a stream of network packets. These packets are still just bytes which are not easy to read and interpret. To solve this issue a parser is needed to read the network packets and display them in a human-friendly manner.

There are many different tools for analysing network traffic, Wireshark being the most well-known and common tool. However, Wireshark does not support WirelessHART packets out of the box so it requires WirelessHART to be implemented as a Wireshark dissector (a protocol specification instructing Wireshark how to display the packet). Furthermore Wireshark cannot be used to “build” packets, only to interpret them.

Another tool to analyse network traffic is Scapy [15]. Scapy is a Python module which can transform a byte stream (network packet) into human-readable form and vice versa. It also does not support WirelessHART out of the box so the protocol needs to be implemented as a Scapy layer. Refer to chapter 2.3.2 for more details on Scapy.

The benefit of using Scapy over Wireshark is that Scapy can use the same protocol implementation not only to interpret packets it receives, but also to build packets in a user-friendly way. It's possible to create a packet by initialising a Python object which inherits a specific Scapy class and then set values in the packet by simply calling the `setfieldval()` function on that object. Furthermore, Scapy is written in Python which allows for easy wrapping with other Python scripts. Having a wrapper around the protocol interpreter is very convenient in order to keep track of network information.

Therefore it was decided to use Scapy as packet analyser and builder.

Sending Packets from GNURadio to Scapy

In order to send the byte stream coming from GNURadio into Scapy, GNURadio needs to communicate with Scapy. The only way for GNURadio to send packets to Scapy without saving them to a file is to use network sockets. The problem here is that Scapy can only listen on physical network interfaces, not on network sockets. Therefore Scapy had to be modified in order to listen to network sockets instead of a physical network interface.

There exists an add-on for GNURadio called gr-scapy_radio, part of the scapy-radio project [4]. This module contains various components, but the most important two components needed for this project are:

- An "Add GR Header" block for GNURadio. This block prepends a specific 8-byte header to each network packet coming out of the packet sink. This header can then be used by Scapy to determine which protocol specification should be used to translate the bytes into human-readable format.
- A Scapy module which can receive packets in binary format from a network socket. This module allows Scapy to receive packets from a network socket instead of an actual network interface.

In the `add_gr_header` block, a protocol id has to be set. For this project, the protocol id 0x07 was chosen since this protocol id was not used in the gr-scapy_radio implementation yet. Connecting the output of the packet sink with the input of the Add GR Header blocks, results in network packets which each contain an 8-byte prefix with the last byte equal to 0x07. These packets can then be transmitted via a socket to Scapy-Radio (which listens on this socket) using the network socket block in GNURadio [4].

The resulting GNURadio flowgraph is attached as appendix (see appendix A). Refer to figure 4.2 for an overview of how network packets are parsed using the bladeRF, GNURadio and Scapy.

Implementing WirelessHART in Scapy

The next step is to build a protocol specification in Scapy such that it can interpret bytes in the stream. As stated earlier, there was already a very basic implementation of 802.15.4 in Scapy [24]. This implementation was taken as a basis and extended to include the WirelessHART-specific Data-Link and Network layers. Furthermore WirelessHART uses specific (static) values for some fields in the MAC layer of the IEEE 802.15.4 standard. These values have also been programmed

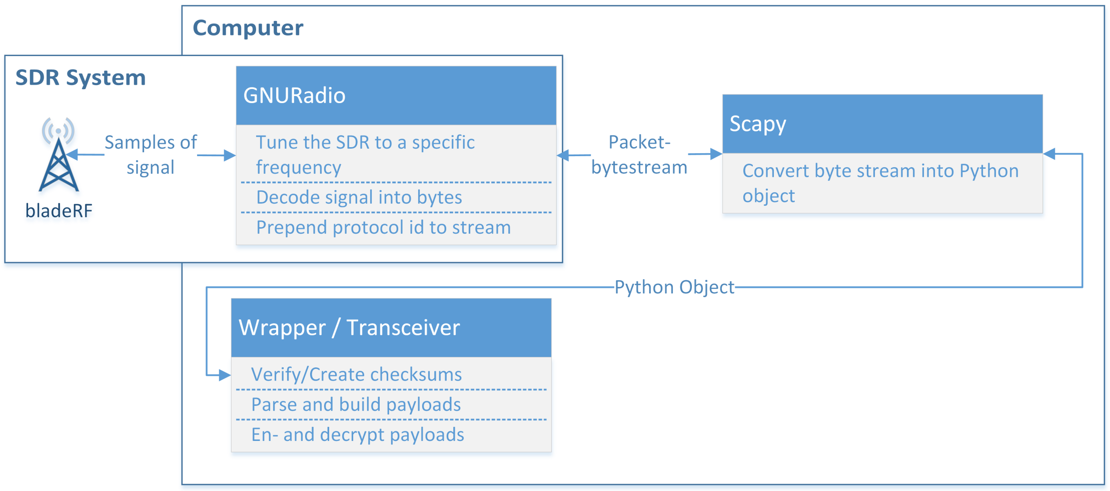


Figure 4.2: WirelessHART packet sniffer component structure

into the Scapy layer. The programming language used is Python since Scapy is built in Python. The layer does not include any processing of Transport and Application layer payloads.

Components of a Scapy Layer A Scapy layer (or protocol specification) consists of a Python file which inherits the `Scapy.Packet` superclass. It then overwrites and defines several functions which are called by Scapy. Scapy expects these functions to do a certain level of processing and return specified values (for example the full packet, the payload or some header fields) in specific formats (binary, integer, etc.). Which functions are required is defined very roughly in the Scapy documentation. Since there is no debugging functionality, implementing these functions is quite hard. Therefore the easiest way to build a layer is to take a pre-existing layer and modify it to suit the WirelessHART protocol. As explained earlier, the `Dot15d4` class (part of the scapy-radio project) was used for this purpose.

How a Packet is Parsed By The Layer The parsing of a packet using a Scapy layer is achieved using the "main" layer. In our case this layer is called `Dot15d4`, and is defined in the Dot15d4 class. This class inherits the `Scapy.Packet` superclass to define default methods. In each class, at least the `name` and `fields_desc` variables need to be defined. The name variable is a simple string containing a display name which will be displayed to the user, where the fields_desc variable defines a list of fields. These fields can be many different types (as defined in the `Scapy.Fields` class). The types define the length in bytes and how the data is displayed. For example, a `BitField` can be used to grab a single bit from the byte stream. This bit can then be given a name and a default value. Custom data types can also be added by adding a class which inherits the `Scapy.Fields` superclass.

When parsing the packet, Scapy reads through these fields and grabs the corresponding bytes and adds them to the Python object representing this packet. Before it does this, the pre_dissect() function is called. This function can perform checks or modify the packet before it is parsed. In the WirelessHART layer we will use this function to verify the Frame Checksum (CRC16, see chapter 4.1.1).

When all defined fields in this main layer are parsed, the guess_payload_class() function is called. This function receives the network packet parsed so far, and should make an educated guess on which sub-layer should be used to parse the rest of the packet. It should return this sub-layer class which Scapy will then use to parse the rest of the packet. This class can be chosen based on values of fields in the packet. Each of these sub-layers then has the same structure defining the same functions. There can be as many sub-layers as needed.

At some point there will be a sub-layer which does not define the guess_payload_class() function (or has this function return False). When this sub-layer is reached, processing of the byte stream stops and the packet object will be returned.

The output of this process is a Python object which contains all fields with their respective values as they are described in the layers and sub-layers. If there are remaining bytes in the stream, these will be stored in the payload field.

Defining WirelessHART Layers and Fields As mentioned earlier, the "main" layer is called Dot15d4. This layer first parses the 802.15.4 headers of the packet. The guess_payload_class() function returns Dot15d4Data if the IEEE802.15.4 *fcf_frametype* field (third bit of the first byte) equals 1. This means the packet is a 802.15.4 Data packet. In WirelessHART, all packets are 802.15.4 data packets since the first byte of the headers is fixed to 0x41.

Furthermore the Dot15d4 class performs CRC checking using the pre_dissect() function. It also implements CRC generation using the post_build() function which is called immediately after the packet is fully built (using all the sub-layers defined later).

It is also important to define a custom field. This field inherits the Scapy.Field superclass and defines a way to interpret WirelessHART network addresses. Since these can be variable in length (either a full EUI-64 address, or a nickname), it is important to convert these addresses to readable format correctly. The dot15d4address field was defined to do this, meaning in all other sub-layers we can simply call this field and have the bytes interpreted correctly. The dot15d4address field takes one parameter, namely the length of the address in bytes. Based on this length it decides how the address should be interpreted.

The Dot15d4 layer calls the Dot15d4Data sub-layer which parses the Data-Link layer headers of the WirelessHART packet. The guess_payload_class() function of this sub-layer determines the type of the WirelessHART packet based on the *wh_frametype* field (the last 3 bits of the last byte of the Data-Link layer headers). This field has the following valid values:

If the packet has the frame type Advertisement, the sub-layer WHAdvertisement is returned. If the frame type is Data, the WHPayload sub-layer is returned. Furthermore, if the frame type is Data a decision is made based on the format of the source and destination addresses to determine whether this packet is perhaps a join request (WHJoinRequest) or a join response (WHJoinResponse). The corresponding sub-layers are returned based on these values.

The WHAdvertisement sub-layer parses the fields in the WirelessHART network advertisement. This sub-layer needs a helper class to extract the superframe details. This helper class is

Table 4.2: WirelessHART Frametype Field

HEX Value	Packet Type
0x00	Acknowledgement
0x01	Advertisement
0x02	Keep-alive
0x03	Disconnect
0x07	Data

needed in order to extract as many superframes as there are superframes in the packet. How many superframes are in the advertisement is defined by the *wh_nr_superframes* field. For each superframe, the id, size (amount of slots) and link details are extracted. The link details contain details needed for the transceiver to operate successfully (slot numbers in which join requests can be transmitted and channel offsets required for channel hopping).

The `WHPayload` class parses the Network layer headers of the WirelessHART protocol. This class is very straightforward since the header fields are static in length. It does not parse the Transport and Application layer payloads and leaves those payloads encrypted in the `payload` section of the packet such that the wrapper script can decrypt them using the correct keys later.

The result of implementing WirelessHART as a Scapy layer is that the network packet bytes from GNURadio can now be displayed in a user-friendly manner. This means that the values of the different fields in the packet can be examined by the user. However, since the WirelessHART protocol uses encryption of the payloads and "random" channel hopping, additional parsing of payloads inside the packet is required.

Many problems arose during the building of the Scapy layer. The issues were mainly caused by the poor structuring of the WirelessHART standard. Detailed information required to parse WirelessHART packets is spread across the entire standard. The problems that occurred during this phase are described in chapter 4.1.6.

4.1.5 Parsing WirelessHART Payloads

Since the WirelessHART protocol exchanges information during the initial join-handshake which is crucial to the network operation and the possibility to sniff network traffic, we want to capture and decrypt this handshake. A wrapper script was developed which uses Scapy to receive (and later transmit) network packets via GNURadio. This wrapper is written in Python, since it's the easiest way to communicate with Scapy. The wrapper receives an interpreted packet as a Python object, allowing it to parse payloads, etc. We call this wrapper the WirelessHART transceiver, since it receives, parses and will later be extended to transmit WirelessHART packets.

Since the session key used to encrypt all subsequent traffic is exchanged during the join handshake, an attacker has to know the key to encrypt that handshake: the join key. It is possible to decrypt all subsequent network communications by decrypting a valid handshake between a legitimate device and the Network Manager. Therefore the assumption was made that the attacker has obtained the join key for the network. This assumption is actually a reasonable assumption since some network devices come pre-configured with a join key and the network administrator is not forced to change this key. Furthermore the attacker does not have to obtain this key using purely technical means. Social engineering can also be used for example.

First the sniffer imports the different modules made available by Scapy (specifically `scapy.main`, `scapy.layers.Dot15d4` and `scapy.GnuRadio`). By importing these modules, the sniffer is able to communicate with GNURadio and interpret packets it receives as Python objects. Also the `numpy`, `time` and `pycrypto` modules [27] are required to do the decryption of network packets, to be able to wait/sleep and do mathematics on timestamps and convert them into slot numbers relative to the superframe size.

The sniffer then defines various functions required to parse network packets as can be seen in table 4.3.

Table 4.3: Overview of Functions Defined in Transceiver

Function Name	**Arguments**	**Description**
parsePacket	packet	Parses the (scapy-)packet. If it's a network advertisement it extracts the crucial network parameters (superframe details, channel hopping offsets and slot numbers). If the packet is a data packet (and thus encrypted), it calls the parsePayload() function.
parsePayload	packet	Extracts and decrypts the Network layer payload (NPDU). For that it has to extract the counter and source address in order to compute the nonce. It then decrypts this payload by calling the decrypt() function, and does minimal parsing of the payload. It checks whether there are any routing details exchanged and stores these routing details (superframes and links). It also extracts the session key and stores that for later use. Finally, the time slot within the superframe in which this packet was received is calculated and compared with the value of the Data-Link layer sequence number (the least-significant byte of the sequence number that is) contained in the packet type. These values should be roughly equal.
decrypt	ciphertext, counter, srcaddr, packet type, key type	Decrypts a ciphertext using counter and src. addr. to build the nonce. Decryption is dependent on the packet type (join request, response, data packet, etc.) and key type (join key, well-known key, session key, etc.). It uses pycrypto to do the actual AES-128 CCM mode decryption.

We now have a functional sniffer which is able to receive WirelessHART packets using Scapy and GNURadio in combination with the bladeRF. Furthermore this sniffer can decrypt the join handshake and extract crucial routing details which are required to sniff subsequent communications between the joining device and the network.

4.1.6 Implementation Problems

There were many problems while building the protocol layer in Scapy. The protocol specification does not define some fields well enough, so there is room for interpretation by the developers of the WirelessHART hardware. One example is the link details in the network advertisements. Paraphrased, the specification states that it should look as follows:

First a byte containing the number of superframes is transmitted. The following should be performed as many times as indicated by this counter: The next 4 bytes indicate the superframe id, amount of slots and amount of links. The link details are transmitted following this counter, as many times as there are links.

In the standard it is not clear whether the link details for this superframe are to be transmitted after each superframe id, or after first sending all superframe id's. It was determined by reverse engineering that for each superframe, all link details are transmitted for all links belonging to that superframe. After this the next superframe is sent including details for the links belonging to that superframe, etc.

Another problem was endianness. The standard specifies that a byte should be transmitted most significant bit first. Multi-byte fields should be transmitted little-Endian meaning the least significant byte of the field is transmitted first. It turned out that this convention was not always followed, leading to confusion. It took a lot of time to figure out the exact byte and bit-order for each field in the packet. As an example the hardware address is a multi-byte field and should be transmitted most-significant byte (MSB) first. But this is against the "general" recommendation that each multi-byte field is transmitted least-significant byte (LSB) first. In some sections of the specification it states that a network address should be transmitted MSB first while in other parts of the standard it's left implicit. The network address at the Network layer is transmitted MSB first by the Dust chip, while it is not specified explicitly in the specification that this should be the case. A developer implementing the WirelessHART protocol could also follow the general definition and have his device transmit the network address LSB first. This is legitimate since it is a multi-byte field and the specification does not explicitly mention it is to be transmitted MSB first.

Encryption and CRC checking had first been implemented in the Scapy layer, but later it turned out to be more convenient to do the encryption in the wrapper that "reads" packets via Scapy and do the checksums in Scapy. See chapter 4.1.3 for more details about the different levels of encryption used.

Another big problem was the structure of the WirelessHART specification. It reads as a "story" and the structure is not very clear for implementors of the protocol. As a result, while reading the chapter on the structure of the Data-Link layer headers, the developer has to go back and forth through the specification in order to find detailed descriptions of how each field should be computed.

4.1.7 Conclusion

Following the procedure described in this chapter, a WirelessHART sniffer has been built using a bladeRF which sends digitised samples of an analog radio signal to GNURadio running on a laptop. GNURadio de-modulates the signal into bytes which are sent to Scapy over a network socket. The wrapper picks up these packets and converts and displays the packets in human-readable form using a WirelessHART layer (specification).

As a result, it is now possible to intercept any WirelessHART traffic and display the contents of the network packets in a readable form. If a handshake is caught, this handshake is decrypted using the join key (if known by the attacker) and the session key extracted from this handshake. All subsequent traffic originating and destined for the device that performed the handshake can now be decrypted and displayed by the WirelessHART wrapper.

4.2 Transmitting Traffic

This section describes how to transmit traffic to the WirelessHART gateway. This is the fourth phase of the project. Transmitting traffic is much more difficult than receiving (sniffing), and many problems occurred when building the transmitter. From a high-level perspective the steps are very clear:

- Sniff a valid handshake
- Get the link details and encryption keys from that handshake
- Transmit packets on those links with spoofed source address of the node to which that link belongs

This chapter will go into detail for each of these steps. Problems that occurred are described at the end of this chapter. It was not possible to build a successful transmitter, but the work presented in this section could prove useful for other researchers looking to transmit traffic onto a WirelessHART or other IEEE 802.15.4 based network.

First off, sniffing a legitimate handshake is straightforward using the sniffer that was built in section 4.1.4. However, the interesting information is encrypted using AES-128 in CCM mode.

4.2.1 Interpreting the Handshake

When the attacker has obtained the handshake and decrypted it using the join key, timing information can be extracted such that it is known to the attacker at which time slots a specific node is designated to transmit a packet. It took me a lot of time and effort to figure out how the time slots actually work (detailed in chapter 3.3.1). The goal was to create a tool that would extract all information from a handshake concerning timing information, channel hopping information and session keys (unicast-gateway, unicast-network manager and the two broadcast keys). This tool would later be integrated with the WirelessHART transceiver.

Since the Linear WirelessHART Development kit is closed-source and the specification did not specify explicitly which commands are to be used during a join handshake, this had to be reverse engineered from the byte stream of the payload. The payloads are HART commands. These commands consist of a unique command identifier (first two bytes), followed by parameters which are specific to each command. The commands have logical names like “Write session” and “Write device nickname” but it is not obvious exactly which commands have to be used during a join handshake.

This problem was solved by manually reverse engineering a join handshake. The HART commands related to WirelessHART network management (timing, channel hopping, session keys, address assignment, etc.) have a command identifier ranging from 777 (0x0309) until 977 (0x03D1). By reading through the bytes of the join handshake in hexadecimal representation, it is a straightforward process to identify two-byte values that start with 0x03. These identifiers were then converted to decimal and searched for in the WirelessHART specification. The parameters to these commands are described in the specification and can be implemented in the handshake interpreter tool.

It turned out that at the start of the payload there are some additional bytes which indicate a status. These bytes indicate whether this payload is a request or a response and whether the previous payload was processed successfully (see section 4.1.1).

The interpreter was built such that it only extracts specific information. To be precise it extracts the following information from a join handshake:

- Timestamps from Advertisement packets. These are required to synchronise the transmitter’s ASN if it wants to transmit packets later on;
- Superframe information (command 965). This includes the size of each superframe and the amount of active slots within that superframe;
- Link information (command 967). This includes for each superframe: time slot numbers, link type (transmit or receive), time slot type (normal, broadcast, join, discovery) and channel hopping information;
- Encryption keys. This includes the unicast session keys (command 963) and broadcast keys (command 961);
- Assigned nickname for the device (command 962). This is used to keep track of this device after the handshake is performed, since as of that moment the source address in the packet will be the device nickname.

This information is stored in a table, such that it can later be referenced to by the transmitter.

4.2.2 Preparing to Transmit a Packet

Now that the attacker has the link details, it is possible to transmit a packet. As a first (test) packet, a join request was used. In this join request, a fresh (not previously used on this gateway) EUI-64 address was used to imitate a new device joining the gateway. All the MICs as well as the encryption of the join request payload (which was extracted from a legitimate join request captured earlier) were calculated according to the specification (see section 4.1.3).

However, the next problem arose immediately: timing. Since timing requirements in this protocol are so strict (accuracy < 2.2ms required), it turned out to be very difficult to transmit a (spoofed) packet at the exact time slot in which the gateway accepts join requests. The receiver listens to his radio at the exact start of the time slot. He then activates a timer which lasts 1.12 ms, after which the receiver expects a packet to arrive. The receiver waits for another 2.2 ms to start receiving the synchronisation header. If it does not receive anything it turns off the radio for the remainder of the time slot. To solve this accuracy issue, various solutions were attempted.

Up until this point, a virtual machine had been used to run GNURadio and communicate with the bladeRF. GNURadio was moved to the physical machine instead of inside a VM. This appeared to run slightly faster but packets were still not received by the gateway, leading to believe the timing was still not accurate enough. It was not known whether timing was the only issue causing the gateway to not receive the join request. However, if timing accuracy was the only issue, it would mean the bladeRF was not suited for communicating with a WirelessHART network.

Next, an attempt was made to find out if there was a delay in the transmission by shifting the time of transmission such that the command to transmit a packet was sent to the SDR before the actual time slot started. The transmission was shifted in steps of 0.2 ms. Making steps of 0.2 ms until the transmission started 10 ms (duration of one slot) before the actual start of the target time slot, did not give any result. This means the gateway did not receive the join request.

According to [28], a GNURadio implementation working with a USRP, has a delay of about 4.2 ms when transmitting traffic. It is safe to assume the delay when sending a packet through the bladeRF is in this same order of magnitude. Taking steps of 0.2 ms and taking into account the 2.2 ms tolerance, we should have come near the correct starting point of a slot at some point. However, the gateway never accepted the packet, indicating the timing may actually not be the issue.

As a last resort to solve the timing issues, different hardware was used: an AT86RF233 chip in combination with a Raspberry Pi.

4.2.3 Raspberry Pi with AT86RF233

To use an Atmel AT86RF233 chip connected to a Raspberry Pi model B, a driver is needed to facilitate communication between the two devices. The Linux-WPAN-Next (a custom Linux kernel) was used for this purpose. This kernel includes a driver for the AT86RF233 chip, but was not developed to support the Raspberry Pi. The Linux-WPAN-Next kernel is based on a Vanilla Linux kernel (the mainstream kernel), while the Raspberry Pi requires specific modifications to this Vanilla kernel (distributed as the raspberrypi-linux kernel). When running a Vanilla(-based) kernel on the Raspberry Pi, the speed is decreased significantly because of the lack of these modifications. An attempt was made to get this set-up to work.

First, the Linux-WPAN-Next kernel had to be compiled for the Raspberry Pi. Since compiling a kernel requires a significant amount of processing power, it was cross-compiled from a server running Debian 8.0 using the tools distributed by the Raspberry Pi Foundation [29]. After the kernel is compiled, the result is a zImage file, which is a gzipped kernel image. Furthermore a modules folder is created, which contains (amongst others) the kernel driver for the AT86RF233 chip. Finally, a device-tree had to be generated in order for the Raspberry Pi to recognise the OpenLabs 802.15.4 board which holds the AT86RF233. These files have to be transferred to the Raspberry Pi's SD card for it to boot this kernel. Booting a vanilla kernel is not possible out of the box. A bootloader has to be used. For this project, a modified version of U-Boot was used [30] since it provided a Raspberry Pi version which was straightforward to compile and run. Transferring everything to the SD card, connecting the AT86RF233 to the Raspberry Pi and giving it power, would boot up the new kernel. As expected, the speed of this kernel is really slow. Writing any random file to the SD card will lock up the kernel.

Once booted into the kernel, the sniffer implemented in section 4.1.4 was installed on the Raspberry Pi. It turned out the kernel module did not support promiscuous (sniffer) mode for the AT86RF233 chip. So, the Linux-WPAN-Next kernel had to be modified in order to support this sniffer mode (also known as monitor mode in the AT86RF233 data sheet [17]). In order to support the sniffer mode, the driver has to be modified. Two configuration flags have to be set: disable the verification of the CRC checksum and disable the incoming packet filter. This packet filter is a built-in function of the AT86RF233 chip which will drop packets not destined for its own EUI-64 address and packets that have an invalid checksum, amongst other things. These features can be turned off by setting specific configuration registers in the AT86RF233 chip. The exact memory addresses and possible values of these registers can be found in the AT86RF233 data sheet [17].

This modified kernel was compiled and installed on the Raspberry Pi. Now that it is possible to put the AT86RF233 chip in monitor mode, it is also possible to run the sniffer. We can now use the Raspberry Pi in combination with the AT86RF233 to capture packets traveling back and forth between the legitimate network devices.

However, after about 200 captured packets (this number fluctuates, but was always less than 1000) the chip stopped working. Once the chip was restarted (by reloading the kernel module), it would work again for a couple hundred packets before crashing again. This took a lot of time to debug, since I was not familiar with the chip and debugging network drivers in the linux kernel. It turned out that the interrupt sent from the AT86RF233 to the Raspberry Pi indicating a packet was ready to be read from the buffer, sometimes was not received correctly causing the chip to wait for the Raspberry Pi to read the packet, while the Raspberry Pi was waiting to receive the interrupt. As a solution, level-triggered interrupts were used instead of edge-triggered. Edge-triggered interrupts can be visualised as a short pulse given on the interrupt line. The Raspberry Pi sometimes missed this pulse causing it to not see the interrupt. When using level-triggered interrupts, the chip would maintain the interrupt line at a high level until it is serviced causing the interrupt to stop and normal operations to continue. This way, the Raspberry Pi does not miss any interrupts and as a result the chip is now stable.

As soon as the chip was stable, I noticed that the AT86RF233 chip missed packets while sniffing. It appeared as if it could not keep up with the speed of the WirelessHART network. In the data sheet of the AT86RF233 [17] it is stated that this chip can handle the same speed as the maximum speed of the WirelessHART network (250kbps). However, this did not seem to be the case when running it with a Raspberry Pi. The cause is the vanilla kernel, which slows down the Raspberry Pi tremendously causing it to not read packets fast enough from the AT86RF233's buffer, which causes packets to be missed. The AT86RF233 cannot receive anything while it is being serviced by the Raspberry Pi (a packet is being read from the buffer for example).

Since the AT86RF233 was not working well enough as a sniffer, transmission of the join request constructed earlier was attempted. The transmission of the join request works (can be captured by the bladeRF) and is transmitted correctly without bits getting flipped, etc. However, the

WirelessHART gateway did not accept the packet transmitted by the Raspberry Pi. The suspected cause of this, is the speed of the Raspberry Pi running the vanilla kernel. It is simply not fast enough to transmit packets in correct time slots: when a packet needs to be transmitted, there is a significant (more than one slot) delay before it is actually transmitted. This means this set-up is not able to transmit packets to the WirelessHART network since such a delay is simply too much.

The Raspberry Pi setup was used for many other debugging purposes. It was very convenient to have, since it could intercept packets transmitted via the bladeRF which allowed for a lot of debugging possibilities. The WirelessHART gateway had to be configured not to send network advertisements, since otherwise the chip would miss the packets sent from the bladeRF. This allowed for debugging the issues with the bladeRF's transmission.

4.2.4 Transmission Accuracy

The packets transmitted via the bladeRF still did not get accepted by the WirelessHART gateway and network manager. Using the AT86RF233 chip as a receiver, it was determined that the packets transmitted via the bladeRF actually had a lot of bits flipped. Meaning the packets where being transmitted totally scrambled. Using trail and error it was determined that the sampling rate of the SDR system was the problem. The bladeRF is connected via USB to the host computer. The USB port is a USB2.0 port, which has a maximum data-rate of 35MB/s which turned out to be not enough for a sampling rate of 8MS/s. 8MS/s was determined using trail and error to be the minimum amount of samples per second that need to be sent from the computer in order to successfully transmit a packet without it getting scrambled. This problem could have been expected since the speed of the USB2.0 port is known, however it was not known how many samples per second were required to successfully transmit a packet, this had to be determined by trail and error.

The solution to this problem was to alternate between receiving and transmission modes on the bladeRF, such that a full 8MS/s was available when transmitting packets as well as when receiving. The problem with this solution is that it is now no longer possible to have full-duplex operation (receive and transmit at the same time). So this solution was not great, but at least it was now possible to transmit packets successfully.

Since this solution was not ideal, the computer was exchanged for a laptop with a USB3 port. USB3 supports much higher data rates and allows for the transmission plus receiving of samples at 8MS/s (so 16MS/s in total) without any problems. Now it is possible to transmit while also receiving WirelessHART packets at full speed, and the packets are no longer transmitted scrambled.

4.2.5 Transmission Failures

It was now possible to transmit packets with a spoofed source address of a node, but they still don't get accepted by the gateway. The reason for this is that the network chip in the gateway which takes care of decoding the radio signal and sending bytes to the kernel (the Dust Networks chip), performs some processing of the packets. It appears this chip processes the Data-Link layer. This was found by analysing a valid join handshake with both the SDR and a built-in sniffer, which runs on the gateway. The sniffer running on the gateway could only see the Network and higher-up layers of the protocol and not the Data-Link layer. Since the Data-Link layer contains a CRC and an integrity code (MIC, refer to chapter 4.1.3), it makes sense that if these values are invalid that the packet is dropped at the network chip and never forwarded to the kernel of the gateway. This made it very difficult because it was not possible to see exactly which keys are used for the Data-Link MIC computations. The reason for this is that keys are always conveyed to the joining device using Command 963 for unicast keys and Command 961 for broadcast keys. There are two unicast keys (unicast-network manager and unicast-gateway). The goal is to find the unicast-gateway key to compute the MIC at the Data-Link layer, so it is expected to be one of the Command 963 payloads.

As a solution, all keys that were extracted from a previously sniffed handshake were tested to see which one is used for computing the Data-Link layer MIC in subsequent packets. It was determined that this unicast key is always the second unicast key (the second Command 963) sent to the device during the handshake. I did not see any other differences in these HART payloads that could indicate what the purpose of the key being sent to the device is.

After implementing this MIC computation using the correct keys, the gateway still did not receive any packets in its kernel transmitted via the bladeRF. The decision was made to try to get a join request packet accepted by the gateway since this packet was encrypted using the join key. This key was known, hence it should be trivial to transmit a valid join request in a time slot allocated for joining.

4.2.6 Transmitting a Join Request

The join request is the initial packet a device would send to the network manager (via the gateway, see section 3.2) in order to join the network. The network manager will reply to this join request with a session key, after which all subsequent communication is encrypted using this session key. The join request packet is a good test since it triggers a log entry in the gateway and therefore should be easy to debug.

The payload of a join request contains a single HART command (0x40) followed by some parameters of the device like the battery level and signal strength. In order to successfully transmit this join request, the attacker has to first sniff three advertisements to synchronise his local ASN with the network. This can be done using the sniffer built in the previous chapter. After the attacker's network clock is synchronised with the network, the actual time slot and channel on which join requests are accepted by the gateway can be obtained from the advertisements (see chapter 3.3.2). After the attacker has this information the join request can be transmitted.

This transmission was successful, meaning the packet was encrypted correctly, the MIC was correct, the CRC was correct and all fields contain legitimate values. However, the packet was still not accepted by the gateway. The only reason this could be was timing. This could be debugged using the AT86RF233 as a receiver while transmitting packets via the bladeRF. An attempt was made to find the delay in the bladeRF's transmission by sending 2000 packets with a delay between each packet of 5 ms. If the delay between the receive time (at the Raspberry Pi) of the packets was static, that would mean the delay of the bladeRF transmitting packets is also static. The result of this test was that the packets were received by the Raspberry Pi at intervals of about 5,3-5,6 ms. This means there is a 0,3-0,6 ms delay in the transmission of the bladeRF and this delay is pretty much static. This meant that the delay between the start of the time slot and the actual transmission was at most 0,6 ms, which is accurate enough for the WirelessHART network (<2,2 ms is required). Furthermore, as described in section 4.2.2 and in [28] the latency should be static and hence it should have been identified using the procedure in section 4.2.2. It was now obvious to conclude that timing was not an issue with the bladeRF and there had to be some other problem.

Since the gateway does not provide any mechanism to debug the Data-Link layer (in which the unidentified problem occurs), it is not possible to further debug why the gateway does not accept network packets transmitted by the transceiver. Therefore the decision was made to abort this phase of the project and continue studying attacks.

4.3 Conclusion

In this chapter a WirelessHART transceiver was developed using a bladeRF. Sniffing and transmitting valid network packets was achieved. However, there was still a bug in the transmission of the packets causing the WirelessHART gateway to not accept a join request required to initiate a handshake to join our transceiver onto the network.

Furthermore, an AT86RF233 chip was used in combination with a Raspberry Pi in order to test whether transmission using this set-up would work. Unfortunately, it did not work since the kernel used for the Raspberry Pi was not fast enough to support the AT86RF233's 256kbps speed resulting in loss of timing accuracy. The AT86RF233 chip in combination with the Raspberry Pi was then used to debug the bladeRF's transmission. It was determined that the transmission timing was accurate enough. The packet structure was exactly according to the WirelessHART specification but still did not get accepted by the gateway. Furthermore the payload of the packet was copied from a legitimate packet so it should be correct as well. From this observation we can conclude there must be some undocumented behaviour in the gateway which causes it to drop the network packets transmitted by the bladeRF. Since there were not enough debugging capabilities in the gateway, it was decided to stop trying to get a join sequence to work and focus on the attacks on the WirelessHART protocol.

Chapter 5
Attacks on a WirelessHART Network

This chapter describes attacks on the WirelessHART network. First, a short introduction will be given with argumentation why these specific attacks are presented. Then, attacks without requiring the join key are presented followed by attacks which are only possible if the attacker does have the join key.

Attacks requiring physical access to the devices are not considered, so side-channel attacks and hardware hacking attacks are excluded from the scope of this chapter.

5.1 Setting the Scene

As described in chapter 3, the security of the WirelessHART protocol depends on the join key. If an attacker obtains the join key for the network its possible for him to decrypt all traffic, given that he can sniff the handshake between the network manager and each device in the network (the nodes). Its also possible to transmit packets with a spoofed source address, since the key used for calculating the MICs and performing encryption in each device is also exchanged during the handshake. Conversely if the attacker does not have the join key, he cannot decrypt the handshake and therefore cannot obtain the session keys used for encryption and computation of MICs for communicating with the nodes. Furthermore he cannot join his own node onto the network without encrypting a packet with the correct join key.

Since it was not possible to transmit traffic to the WirelessHART gateway (see chapter 4.2), the attacks presented in this chapter are theoretical and not validated on actual WirelessHART hardware (with the exception of a generic jamming attack described in section 5.2.1). The WirelessHART specification has been studied extensively in order to determine whether the attacks are actually possible. This chapter will only cover attacks which are possible according to the WirelessHART specification.

There are many well known attacks on wireless mesh networks. Although these attacks were originally excluded from the scope of this project, it was decided to include these known attacks in this chapter and to investigate whether they are applicable to the WirelessHART protocol or whether the WirelessHART security features thwart them. Please note that this chapter does not provide an exhaustive overview of all attacks possible. The attacks described in this chapter are focussed on breaking the availability and integrity of the WirelessHART network, since these are the most important security requirements of an industrial communication protocol.

There are a couple of papers written about the security of IEEE 802.15.4 networks and WirelessHART networks. For example the paper by Raza et al. [31] gives an overview of attacks that might be applicable to WirelessHART networks. However, these attacks are not described in detail. This chapter aims to go into detail of the applicable attacks to WirelessHART and explain why they will work with which restrictions.

5.2 Attacks Without the Join Key

This section describes attacks for which the attacker does not require the join key. If the attacker does not have the join key, it means he cannot know any of the session keys and can therefore not decrypt encrypted payloads or calculate integrity codes (MICs). He can however interpret header fields, since these are transmitted in the clear.

5.2.1 Jamming All Channels

Perhaps the most simple attack on the availability of a WirelessHART (or IEEE 802.15.4) network is jamming. Jamming is the intentional disruption of a radio signal by introducing another radio signal with the same modulation technique on the same frequency as the original radio signal. This creates a disturbance in the receiving of the legitimate signal. WirelessHART uses channel hopping in order to provide resistance against jamming attacks. The assumption from the WirelessHART designers is that an attacker can never jam all channels at the same time.

Furthermore, WirelessHART uses channel blacklisting [3] in an attempt to evade jamming attacks. As soon as another signal is detected on one channel, the channel is blacklisted by all devices and will be excluded from the available channel list causing that particular channel not to be used anymore for a certain period of time.

In order to perform a jamming attack, the attacker can use common devices which operate in the 2.4GHz band. For example cheap WiFi adapters (that support promiscuous transmit mode) can be used to continuously transmit arbitrary WiFi packets. This type of jamming is called *continuous power emission*. However, if he uses non WirelessHART-compliant devices like WiFi adapters, the signal strength of these devices must greatly exceed the one of the WirelessHART devices. This can be hard given the environments WirelessHART may be deployed in (factories, manufacturing plants, etc.). The goal is to transmit a strong enough signal on all channels, causing WirelessHART to blacklist all channels. This is hard because the signal strength of these network adapters must greatly exceed the legitimate devices to be able to interfere with the network.

A better way would be to use an AT86RF233 chip (or any other IEEE 802.15.4 compliant radio) which can actually transmit valid IEEE 802.15.4 packets at a fast enough rate. This type of jamming is called *concurrent packet transmission*. The attacker would need to develop a PCB (Printed circuit board) which contains (or connects to) a large enough antenna and a signal amplifier in order to gain enough signal strength to have a signal strength which is comparable to the legitimate devices' signal strength. The WirelessHART specification states that the transmit power of legitimate devices should be a nominal 10 mW, ±3 dB (in line with the Equivalent Isotropic

Radiated Power or EIRP) [3]. An attacker could calculate based on the surroundings and rough layout of the WirelessHART network what the average signal strength will be from a legitimate device to the gateway, and then adjust his radio for this value. Using an amplifier he can get much higher transmit power to overpower the other devices. The receiver sensitivity defined in the WirelessHART specification is not relevant here, since the goal is to overpower the other devices and disrupt the legitimate communications.

A study on the effectiveness of these two types of jamming (continuous power emission and concurrent packet transmission) has been done in [32]. It was found that when the jamming device transmits 802.15.4 compliant packets, a little over 60% of the legitimate packets are still received. This number is quite high and is thus this type of jamming is not very effective. It is not documented in the WirelessHART specification exactly when a channel will be blacklisted. Testing is required to see whether the 40% packet loss is enough to get the network manager to blacklist a specific channel.

Since there are 15 channels in use by default, an attacker would require (approximately) 15 devices to cover all channels simultaneously (its possible that some other devices transmit packets with channel overlap, so less than 15 devices may work). However, the AT86RF233 chips cost around 4 euro each. Add a little bit of cost to develop and print a PCB and antenna and the total cost of one unit to jam a single channel would be roughly 15 euro. The total cost of 15 units would be: 15 euro ∗ 15 units = 225 euro which is relatively cheap considering a software-defined radio which can only handle one channel simultaneously costs upwards of 600 euro. These 15 devices must be controlled by a micro-controller, an FPGA or a computer. Using a cheap system on a chip like an Arduino or a Raspberry Pi would make this attack device fit in a shoebox and make it highly portable and hard to detect. Furthermore these mini computers are cheap and do not consume large amounts of power. A much nicer solution would be if the attacker included a micro-controller on the PCB that holds the AT86RF233 chip with required components. He has to manufacture these PCB's anyway. Then, the size of this attack device could be made very small.

It is unknown whether the WirelessHART Network Manager will always keep one channel available. In the best case, the Network Manager will keep one channel available to communicate with the legitimate devices, and keep checking the other channels whether the interference has stopped. However, this is not specified in the WirelessHART specification so it is unclear how this scenario is handled by the WirelessHART network manager. Therefore it is also unclear whether this attack would actually disrupt the network. It will however decrease the total throughput of the network since not all channels are available for transmission and a smaller set of channels has to be shared by the same amount of devices.

5.2.2 Jamming Join Slots

It is possible to further optimise the jamming attack by jamming only the slots allocated for new devices to join the network. This will prevent legitimate devices from joining the network. Note that this attack will only disrupt the network if combined with a de-authentication attack forcing legitimate devices off the network. Such an attack does not appear to be possible without knowledge of the join key, and this jamming attack might therefore be ineffective. This attack has not been tested since it was not known whether the transmission of packets via the transceiver was actually accurate enough.

Since the network advertisements of a WirelessHART network are not encrypted, it is possible for an attacker to gather information on the exact time slots during which the network manager (and gateway) expects join requests from legitimate devices. These time slots are the links and slot numbers mentioned in the WirelessHART network advertisement (see chapter 3.2). Also the channel offset required for the attacker to follow the channel hopping sequence is included in the advertisement so the attacker has all information required to transmit (invalid) join requests. The join requests transmitted by the attacker are invalid since he does not have the join key required for encrypting the payload and calculating the MIC on the Network Layer. However, a valid Data-Link layer can be assembled by the attacker since this layer has a MIC calculated using the well-known key which is public.

The attacker can transmit invalid join requests (encrypted with a different key) during all time slots allocated for joining. If the attacker can transmit such a well-formed but erroneously encrypted packet, the gateway will send back an Acknowledgement packet to the attacker acknowledging the receipt of the packet. This behaviour happens because at the Data-Link layer, the packet appears to be legitimate. The decryption will then fail at the network manager's Network layer processor, since the Network layer payload must be encrypted and signed using this network's join key (which the attacker doesn't have). It has not been tested whether the Network Manager will actually transmit a failure report back to the attacker, but this behaviour is not documented in the WirelessHART specification so it is unlikely that the Network Manager will transmit any packets as response to this false join request.

The attacker could use the Scapy implementation built in chapter 4.2. Combining this Scapy implementation with a Raspberry Pi and any IEEE 802.15.4 compatible radio (such as the AT86RF233 chip), will result in the attacker being able to transmit (invalid) join requests. He should time these join requests to start transmission sooner than the legitimate devices, since the gateway will accept the first packet it receives.

Since the WirelessHART protocol makes use of a back-off timer in case of a collision (briefly explained in chapter 3.3.2), any legitimate device wanting to join the network will detect a collision if the attacker's packet reached the gateway earlier than the legitimate device since the gateway will only acknowledge the first packet it receives in a time slot. If a device detects a collision, it will wait a variable amount of seconds before re-transmitting the join request in another time slot. But after waiting, the attacker will flood that slot too, so the legitimate device will wait another variable amount of seconds. This creates a loop that will run as long as the attacker is able to transmit packets in these join slots.

For this attack to work it is required that the attacker can time the transmission of a packet very precisely. Since the gateway will accept the first packet to arrive (intact) within a certain time slot and forward it to the network manager, the attacker needs to beat the legitimate device by making sure the gateway will receive his packet first. Furthermore the attacker must make sure that his signal is more powerful than the legitimate device's signal since otherwise interference may happen and both packets may be dropped. If both packets are dropped, the attack is considered successful too since the legitimate device will activate the back-off timer again and wait for the next slot to re-transmit its join request. It is not known whether there is a maximum amount of retries before the legitimate device gives up. This does not appear to be documented in the WirelessHART standard and can therefore be dependent on the implementation.

In this attack, an attacker would require just a single IEEE802.15.4 radio. Since this configuration uses a very small amount of power, its possible to build a very small battery powered device that performs this attack.

5.2.3 Traffic Analysis

Sniffing traffic in itself is an attack on the confidentiality security requirement. As stated earlier, this security requirement is less important than the availability or integrity requirements. However, this attack can be used to gather information in order to make other attacks on availability or integrity possible.

Using a sniffer such as the one developed in chapter 4.1, it is possible for the attacker to analyse metadata contained in the packets. Since large parts of the packets are not encrypted (the Data-Link and Network layer headers to be specific), its possible to perform an analysis on which devices are currently active in the network. The headers contain source and destination addresses and routing information. Table 5.1 provides an overview of which fields of a packet can be seen by an attacker that does not have the session key (nor the join key).

Table 5.1: Information Leakage without Join Key

Field Name	Values
Slot Number	The attacker can obtain synchronisation with the absolute slot number using the Sequence Number in the Data-Link layer header.
Network ID	The attacker can get an overview of all WirelessHART networks in range. If there are multiple networks running in an area, its possible to detect this without having any access to any of the networks.
Src/Dst Addresses	The attacker can obtain a list of MAC addresses of the devices operating on the WirelessHART network. The MAC addresses tell the attacker which brand of devices are in use, and may give the attacker information about the join key (some brands use a preset key which does not have to be changed by the Administrator).
Src/Dst Addresses	The attacker can also obtain a list of "nicknames" (similar to IP addresses in a TCP/IP network). This does not give the attacker much information besides the amount of devices active on the network.
NLPDU Nonce	This field provides the attacker with information regarding the nonce used in the AES encryption and signatures generated at the Network layer.

5.2.4 Transmitting Fake Advertisements

Transmitting fake advertisements is another attack on the join procedure of the network. An attacker can transmit "valid" network advertisements since these advertisements do not require knowledge of the join key. Only the well-known key is required, which is documented in the WirelessHART specification.

Using traffic analysis, the attacker can determine the network ID of the legitimate network by sniffing legitimate network advertisements. Using this information, he can build spoofed network advertisements using the Scapy layer described in this thesis. It is not documented how a joining device decides which advertising device will be used to join the network. The standard specifies that any device which is taking part in the network is able to transmit network advertisements. Other devices can then join the network via these advertising devices. It's likely that the joining device chooses the advertising device which is nearest to itself. This can be done using the RSSI (Receiver Signal Strength Indicator), which indicates the signal strength of a received packet.

If an attacker can build a transmitter that can overpower the gateway (which transmits the legitimate advertisements), it might be possible for joining devices to start transmitting join requests to the attacker instead of the gateway. The attacker can *not* perform the handshake with the legitimate device since he does not have the join key with which the device expects the response packet to be encrypted. It is also very likely that the legitimate device will choose another advertising device to join the network if it does not receive a reply from the attacker. Therefore it is not likely that this attack will be very effective without the attacker having the join key.

5.2.5 Bruteforcing the Join Key

The initial join key for a network is usually configured by the vendor. Some vendors may use schemes to generate this join key, making it prone to brute-force attacks if the attacker knows this scheme. The problem for users is that changing the initial join key requires the network to be booted first. Once a WirelessHART network is formed, Command 963 (Write Session) can be used to change the join key for future handshakes over the air. However, changing the initial join key requires this command to be sent via *wired* HART. This involves opening the device and connecting it to a computer using the wired HART protocol. The problem with this procedure is that when opening a device, it voids any certifications the device has. For example a certification that the device can operate under high temperatures/pressures or in nuclear environments. Therefore it is expected that end-users will use the pre-configured join key to start the network the first time.

Since it is highly likely that vendors of WirelessHART devices use a scheme to generate this initial join key, a brute-forcer was implemented to brute-force the last 8 bytes of the join key. If the vendor uses a scheme where the last eight bytes are unique, this brute-forcer can find the join key used to encrypt a sniffed join request packet in a few hours on a quad-core computer. It requires maximum 2^{64} AES decryptions which can be completed in a reasonable amount of time. The input to the brute-forcer is a join request (or a join response). It is not necessary to have both the request and response to guess the join key, since the payload structure of both packets can be recognised when decrypted successfully.

5.3 Attacks Requiring the Join Key

This section will describe attacks in the scenario that the attacker has obtained the join key for the network. This can be done using non-technical means like social engineering or by guessing the key (some join keys might be predictable). It is noteworthy that if the attacker only has the join key it does not mean that he can communicate with all devices. He also needs to capture a handshake between the target devices and the legitimate network manager. He can decrypt this handshake using the join key in order to obtain the Network and Data-Link layer session keys. The Network layer session keys (used to encrypt the payload) are unique for each device so he needs to capture handshakes of all devices he wants to target.

If the attacker captures a handshake for a specific device and he has the join key to decrypt it, he can transmit packets in the name of the network manager by setting the Network layer source address to 0xF980 which is the address of the Network Manager and the Data-Link layer source address to 0x0001 and encrypting the packets with the keys extracted from the handshake. The target devices will accept these packets as if they are from the legitimate network manager transmitted via the gateway.

5.3.1 Mass De-Authentication and Denial of Service

Mass de-authentication is the most important attack found during the study of the WirelessHART specification. It is a very noisy attack: if the network manager keeps logs, it can be observed by the Network Administrator very easily. Furthermore this attack will disrupt the operations of the network, meaning the WirelessHART devices will not transmit any sensor values anymore, causing a process operator to immediately notice that something is wrong.

Command 972 (Write Network Suspend) can be used by the network manager to temporarily suspend a WirelessHART network for a specified number of time slots. Suspension means that all devices that receive this packet will cease all radio communication for the specified amount of time slots. In the specification it is stated that this command can be used when safety procedures do not allow radio communication in a certain area for example during mining blast operations. The command takes two arguments: the time (ASN) at which suspension should start and the time (ASN) at which the network should resume communications. Both these numbers are 40bit unsigned integers meaning the maximum value of the variables is 2^{40}. The devices will then go to sleep for this many slots, which have a duration of 10 ms each. Therefore, the network can be suspended for at most $(2^{40} * 10\text{ms})/1000$ seconds (which equals 348,4 years).

This command is allowed to be transmitted as a broadcast or as unicast, and can therefore be encrypted with the broadcast-network manager key at the Network layer. Also the MIC can be generated using this same key. At the Data-Link layer the broadcast-gateway key can be used. The attacker can obtain these keys by decrypting any sniffed handshake using the join key or joining his own device onto the network, since this key is the same for all devices. Therefore this attack can be used to force a mass re-authentication allowing the attacker to capture all handshakes.

The attacker transmits a packet with Network layer source address 0xF980 (the network manager) and Data-Link layer source address 0x0001 (the gateway). The payload of the packet should be the command 972, encrypted using the broadcast-network manager key at the Network layer and the MIC at the Data-Link layer should be calculated using the broadcast-gateway key.

In the WirelessHART specification it is specified that a device which receives a command 972 packet will "clear all its network information" [3]. This sentence is further explained in another chapter of the specification which describes the Network layer state machine. It is clear from that chapter that the target device will delete all session keys and disable its radio until the specified resume ASN is reached. Once the resume time is reached, it will re-join the network by performing a handshake with the gateway.

The value of the resume time depends on the attacker's goal. If he wants all devices to re-authenticate so he can capture the handshakes, he might set the resume time to a very low offset from the suspend time. If his goal is to disrupt the network, he should set the resume time to a very high value causing the devices to go to sleep permanently (until the administrator reboots them). Practically this would mean all sensors have to be power cycled, causing massive disruption in the plant processes.

There is a possibility that there exists a resume command (although it is not documented in the WirelessHART specification). The HART Communication Foundation has a patent [33] which describes how a wireless network can be suspended temporarily. The patent also describes that the devices will keep on listening while they are in suspended state such that one device (presumably the Network Manager) can transmit a resume broadcast command. This will cause the network to wake up from its suspended state. However, this resume command is not documented in the WirelessHART specification. Furthermore, the Network layer state machine clearly states that the device will turn off its radio upon receiving the suspend command. This means there can not be a resume command, since no device would receive this command as all the radios are turned off.s

This attack has a very high potential. The attacker can force devices to perform a handshake, knowing only the join key for the target network. Since handshakes are required for almost all other attacks, this is a very good entry point. Furthermore it is possible to selectively bring down a certain node if the attacker already has the unicast session keys for that node. He can then transmit the attack packet directly to the target node instead of sending it as broadcast.

Care must be taken, since the attack will generate a lot of noise. There might be log entries in the Network Manager and a process operator will notice the devices going down for a short (or long) period of time since his operating interface will not receive any sensor values until the network has converged again (all devices are joined).

5.3.2 Nonce Exhaustion

As described in chapter 3, WirelessHART uses AES-128 in CTR mode to encrypt Network layer payloads. Since this algorithm uses a nonce, it also provides replay protection. The network manager keeps a list of all Network layer nonces associated with all devices joined on the network. If the network manager receives a packet from a device, with a nonce counter lower than the nonce it has on record it will drop the packet [3].

An attacker can abuse this feature if he has the session key between the network manager and a target device. He then waits for the device to transmit at least one packet using this key, so the attacker can extract the nonce from that packet. The attacker should aim to obtain the nonce used in combination with the Unicast-network manager key used by the device to communicate with the network manager at the Network layer.

The attacker can then transmit a packet using the Scapy layer described earlier in this thesis with a spoofed source address of the target device, and destination address of the network manager. The nonce (aka. counter) field (1 byte, see 4.1.3) in the Network layer of this attack packet can be set to the maximum value minus one (which is 0xFE). It must be set to the maximum minus one, because when the nonce counter in the session table of the network manager reaches the maximum value it will roll-over to 0x00. The problem with the roll-over is that if the network manager has a nonce counter of 0x00 in its session table, any nonce transmitted by the legitimate device will be accepted since it is larger than the one stored in the session table.

The payload of this packet does not have to be valid, but the packet has to be encrypted using the correct session key (unicast-network manager key) otherwise the packet will be dropped. All other fields in the Data-Link and Network layer headers need to be set correctly, like the ASN and sequence numbers.

Upon receipt, the gateway will first check the checksum (CRC-16), parse the Data-Link layer headers and then the network manager will start parsing the Network layer headers. It will extract the source address (which is the attacker's target) and the nonce (aka. counter) field from the packet which it uses to reconstruct the full nonce. It will perform a lookup to see whether the reconstructed nonce is greater than the nonce it has on record in its session table. This check will pass since the attacker knows the nonce which the network manager has in its session table and knows that it is less than the maximum value (namely the maximum value minus one). Next, the network manager will decrypt the payload using the nonce and unicast-network manager key associated with the source-address from the packet. Upon decryption, the nonce record will be updated.

It is unknown whether the nonce record in the session table is incremented *before* parsing the payload or whether it is incremented *after* parsing the payload. This is not specified in the specification. If it is incremented before parsing the payload, the payload does not need to be valid. However, since the attacker can build valid payloads using the Scapy layer described in this thesis and the WirelessHART specification, he might as well construct a valid payload. This could be for example a device health report (WirelessHART command 779) or a ping request.

As soon as the nonce counter in the session table of the network manager is updated to its maximum value minus one, the network manager expects the next packet from this device to have a nonce counter of 0xFF. All packets subsequently transmitted by the legitimate device will have a nonce counter lower than this value since the legitimate device increases its local counter by one upon each new packet. The expected result is that the network manager flags the legitimate packets as replays since it cannot successfully reconstruct the nonce (see section 4.1.3), and will drop the packets. It is not documented after how many failed retries from the legitimate device it will initiate another handshake. It is expected that at some point the device will notice that it cannot reach the network manager anymore and will re-join the network. This will likely happen before it reaches the maximum nonce value.

In [34] this attack is also described. The authors claim that this attack can be used to perform a permanent disruption of the communication between a single device and the network manager. However this might not be true since the device might keep on trying to reach the network manager, increasing its nonce upon each transmission. At some point it will reach the maximum value of the nonce counter, causing the packet to be accepted by the network manager (this process will take millions of packets). This behaviour is not documented in the WirelessHART specification.

In order for an attacker to make this into a permanent attack, he constantly needs to sniff the network traffic monitoring the nonces transmitted by the target device. As soon a the target device transmits a packet with the maximum nonce value, the attacker must transmit the denial of service packet again with the nonce counter of the maximum minus one.

Testing of this attack is required to observe how a legitimate device acts when it cannot communicate with the network manager anymore (because the nonce is out of sync). It might be possible that the device initiates a join handshake, in which case this attack may be interesting to perform in combination with other attacks mentioned like transmitting fake advertisements.

5.3.3 Time Slot Saturation

If an attacker has obtained the unicast-gateway key for a target device, he can then transmit network packets to the gateway in each time slot allocated to the target device. It is also possible to use statistical analysis to determine which time slots are allocated to which device, however it cannot be determined with absolute certainty whether the time slots observed are all the time slots allocated to the device. Assuming the attacker can time the transmission of these packets accurately enough, a denial of service will happen.

When a legitimate device wants to transmit a packet, it will detect energy on the channel causing it to wait for a specific amount of time (the back-off timer, see section 3.3.2). After this timer has expired, it will try to retransmit the packet in the next time slot, however the attacker will transmit a packet in that time slot too causing the legitimate device to go in back-off mode again. This way it is possible to completely block a specific device from the network without the gateway noticing that the device is unreachable. Testing of this attack is required to validate whether the legitimate device will attempt to re-join or perhaps try to communicate with the gateway outside of its allocated time slots.

5.3.4 Device Hijacking

As described in section 5.2.4 it is possible for an attacker to transmit valid network advertisements. If the attacker has the join key, it is possible for him to perform a handshake with a device that wants to join his malicious gateway. Up until that point, it is important that transmission of the packets between the attacker and target do not get interrupted by WirelessHART traffic of the legitimate network, since this will greatly slow down the joining process. If it is slowed down too much or there is too much interference, the target may choose another gateway to join the network.

Once the handshake has been performed with a target device, it will not be able to communicate with the legitimate gateway since the legitimate gateway does not have the corresponding session key. This gives the attacker full control over the target, since he has a one-on-one session with it. He can now issue management commands originating from a malicious Network Manager in order to change configuration settings on the target. This will work because the target accepts packets with a (Network layer) source address of 0xF980 as coming from *the* Network Manager (remember there is only one per network). If these packets are encrypted with the correct session key the target executes whatever command is in the payload of the packet.

One of the parameters that can be changed over the air is the network ID (using WirelessHART command 773). Once the network ID is changed, the device will not join the legitimate WirelessHART network anymore since it is now configured for a different network. This new network ID will be used upon the next join, so it is important to send a disconnect command to the target (command 960).

The attacker can now set up a WirelessHART network with this new network ID and start transmitting advertisements. He can then perform a handshake with the target that tries to join his network. The target is now fully joined onto the attacker's WirelessHART network and will not join the legitimate network anymore unless the Network Administrator (human) re-configures the device.

5.3.5 Time De-Synchronisation

If the attacker has the unicast-gateway and unicast-network manager keys, he can issue command 793 (Write UTC time mapping). This command has two parameters, the date and time at which the network's absolute slot number (ASN) was zero. If the attacker provides an erroneous timestamp, the device will fail to calculate the correct ASN and superframe slot numbers for the network causing its slot timing to be de-synchronised with the network. This command is only accepted by the target if the sender is the Network Manager, so the Network layer source address needs to be set to 0xF980.

This attack needs to be tested since it is not clear from the specification what should happen when a device loses time synchronisation. It is possible that the device tries to re-synchronise its time with the network manager however this does not seem likely since it can no longer communicate with the gateway (thus also not reach the network manager) because their slot timing is different. If this is not the case, it might re-join the network as if it got disconnected.

5.3.6 De-Authentication

If the attacker has obtained the unicast-gateway and unicast-network manager keys, he can use command 960 (Disconnect device) to force a device off the network. The Network layer source address of that packet must be equal to 0xF980 and the packet must be encrypted using the unicast-network manager key between the Network Manager and the target device. This command will cause the legitimate device to transmit an acknowledgement to the legitimate network manager after which it will disconnect from the network.

The documented behaviour is that a device will automatically reconnect to the network. Once it joins the network again, the attacker can capture the handshake including all session keys again. He can then simply send another command 960 packet with a spoofed source address of the network manager in order to disconnect the target again.

This behaviour was also observed by sending a command 960 packet from a real network manager to a specific device using the WirelessHART Development Kit. The target device announced it was going to disconnect to the rest of its neighbours after which it would disconnect. The target would then automatically re-join the network.

5.3.7 Transmitting False Data from Network Devices

If the attacker has obtained the unicast-gateway and unicast-network manager keys for a target device, it is trivial to inject network packets which may include forged sensor values. The result may be that the process operator makes wrong decisions, and for example speed up a motor or open or close a valve causing physical damage. This is an attack on the integrity security requirement. It has the potential to cause physical damage so depending on the attacker's motive, this might be an interesting attack.

5.3.8 Transmitting False Data to Network Devices

Packet injection is also possible in the opposite direction once the attacker obtained the unicast session keys. He can transmit packets as if they are coming from the process operator's workstation instructing the WirelessHART device to perform an operation like open or close a valve or speed up a motor. The downside of this type of packet injection is that the device will report the result of the command back to the process operator's console allowing the him to see that something is wrong (e.g. the valve closed without the operator issuing the command).

5.4 Conclusion

In this chapter, several attacks have been described in detail. The attacks are focussed on the two most important security requirements in the WirelessHART protocol: availability and integrity. First, a number of attacks were presented that do not require the attacker to have any knowledge about the network. Disruption of the network is possible using jamming, however this attack does not seem to be very effective as proven in [32].

More effective attacks are possible when the attacker has knowledge of the join key. A mass de-authentication attack is described, allowing an attacker with knowledge of the join key to shut down the entire WirelessHART network (including gateway(s)) for either a short period of time or for a very long period of time. When shutting it down for a short period of time this attack can be conceptualised as a network reboot, allowing the attacker to capture session keys between all devices. These keys can then be used for other attacks.

Once the attacker has obtained the correct session keys, he has full control over the network. He can inject packets with false measurements (integrity) and disrupt communication to and from specific devices in various ways (availability).

Chapter 6

Conclusions and Future Work

6.1 Conclusion

The main research question this thesis answers is: Which attacks are possible in a WirelessHART network? In order to find the answer and to develop tools for security researchers, two sub-questions were answered.

1. Which (generic) hardware can be used to communicate with a WirelessHART network?

This question was covered in section 4.1 where a method is developed to communicate with a WirelessHART network. Using a Software-Defined Radio (SDR), it is possible to build any type of signal modulation scheme in a software program running on a computer, using a single piece of hardware (in this case the bladeRF). When building such an implementation, it is important to understand the physical layer of the target protocol. In the case of WirelessHART, this is based on IEEE 802.15.4, which is an open specification. Hence it was trivial to build the modulation scheme for this communication protocol as it is well-documented in the IEEE 802.15.4 specification. Care must be taken that the hardware component of the SDR (bladeRF) has a high enough throughput. For the WirelessHART protocol, 16 MS/s was determined to be the minimum amount of samples per second required to be able to receive and transmit simultaneously (8 MS/s for receiving and 8 MS/s for transmitting).

An easier solution was to use readily available radios which can receive and transmit traffic on a specific physical layer. In the case of WirelessHART, any IEEE 802.15.4 compliant radio can be used. An AT86RF233 transceiver was used in combination with a Raspberry Pi in section 4.2.3. If such radios are available in future protocol security assessments, it might be quicker to use such a radio to perform the security assessment instead of re-building the physical protocol in an SDR. Care must be taken that these radios must be able to be connected to a computer in order to dynamically transmit network packets. For some protocols this might not be the case and it might be more simple to use an SDR instead.

2. How can network packets of a WirelessHART network be interpreted and generated?

Interpreting WirelessHART network packets is described in section 4.1.4. The tool used to interpret network packets is Scapy. This tool translates a stream of bytes into readable network packets and vice versa. It is flexible in the sense that it can perform this task for any given network protocol. Each protocol must be implemented as a protocol specification called a layer. Scapy then allows the security researcher to interpret packets by taking an input byte stream which it will output in a readable format (python object) according to the given layer. It also allows the security researcher to execute attacks by translating a python object back into a properly formatted byte stream. This byte stream can then be transmitted over a wireless network via the SDR.

Transmitting traffic is more difficult than receiving traffic. Transmission of network packets is covered in section 4.2. Since WirelessHART has very strict timing requirements, and the available WirelessHART hardware had almost no debugging capabilities it was very difficult to transmit packets onto a WirelessHART network using the SDR. In this project, we did not succeed in transmitting packets such that they are accepted by the network. Both methods (SDR and the Raspberry Pi setup) were attempted, both did not succeed. Using both methods it was possible to transmit packets correctly and at the correct timestamp, but the packets did not get accepted by the WirelessHART gateway, meaning that there is still an open (unknown) issue. More research and testing is required to be able to successfully transmit WirelessHART packets using this generic hardware.

Main question: Which attacks are possible in a WirelessHART network?

In chapter 5, eleven attacks are listed. None of these attacks could be tested, since there was no way to get packets accepted by the gateway (or other network devices). However, the protocol specification has been studied extensively in order to determine that these attacks are possible.

Four of these attacks do not require the attacker to know the join key of the network, but these attacks are not really efficient or are expected to be ineffective. These attacks include a passive attack (sniffing network traffic in order to gain a picture of the network) and two types of jamming attacks. Furthermore it is possible to transmit malicious network advertisements without knowledge of the join key, causing the legitimate devices to try to join the attacker's network instead of the real network. This is expected to be ineffective since without the join key, the attacker cannot perform a handshake with the joining device and the joining device will simply try to join one of the other advertising devices which belong to the legitimate network.

Seven attacks are described which do require knowledge of the join key. Due to the use of per-device session keys, the attacker needs to capture a handshake between each target device and the network manager, which he can then decrypt using the join key to obtain the session key. The problem is that these devices generally do not perform handshakes anymore once they are joined to the network.

An attack was found which has no other requirements besides knowledge of the join key (and a working transmission device). It can be used to de-authenticate all legitimate devices by sending a single attack packet. There are two variants: it can be used for denial-of-service or to force all devices to perform a handshake. It works by having the attacker join his own device onto the network to obtain the broadcast session key between the network manager and the legitimate devices. He then uses this broadcast session key to broadcast a Command 972 packet in order to suspend the network for a specified amount of time. If he picks this time to be very short (must be greater than 10 ms), all devices will re-authenticate with the network causing all devices to perform handshakes. If the attacker then captures these handshakes, he obtains all the session keys by decrypting the handshakes with the join key. With these session keys the other six attacks become possible like injecting false data (measurements) or performing a selective denial-of-service for a single device instead of the entire network using Command 972 or one of the other attacks described in chapter 5.

If the attacker has the join key, he can also hijack devices. He can transmit malicious advertisements and then perform a handshake acting like the network manager. This allows the attacker to hijack and permanently disrupt service for specific devices as he can change the network configuration on those devices (network id or other parameters) causing it to not join the legitimate network anymore. The only way to fix the hijacked device after this attack is for the network administrator to re-configure the device manually which usually involves opening the device.

6.2 Future Work

Since it was not possible to build a transmitter capable of getting network packets accepted by the WirelessHART network, this is a future research possibility. If in the future it is possible to transmit arbitrary packets, all of the attacks mentioned in chapter 5 can be tested.

The AT86RF233 driver can be ported to the rpi-linux kernel, enabling the full speed of the Raspberry Pi device. The result of porting the driver to the rpi-linux kernel will be a dramatic speed increase, allowing transmission in specific time slots while also performing encryption/decryption and calculating MIC's. The Scapy implementation of chapter 4.1.4 can then be used to test the attacks.

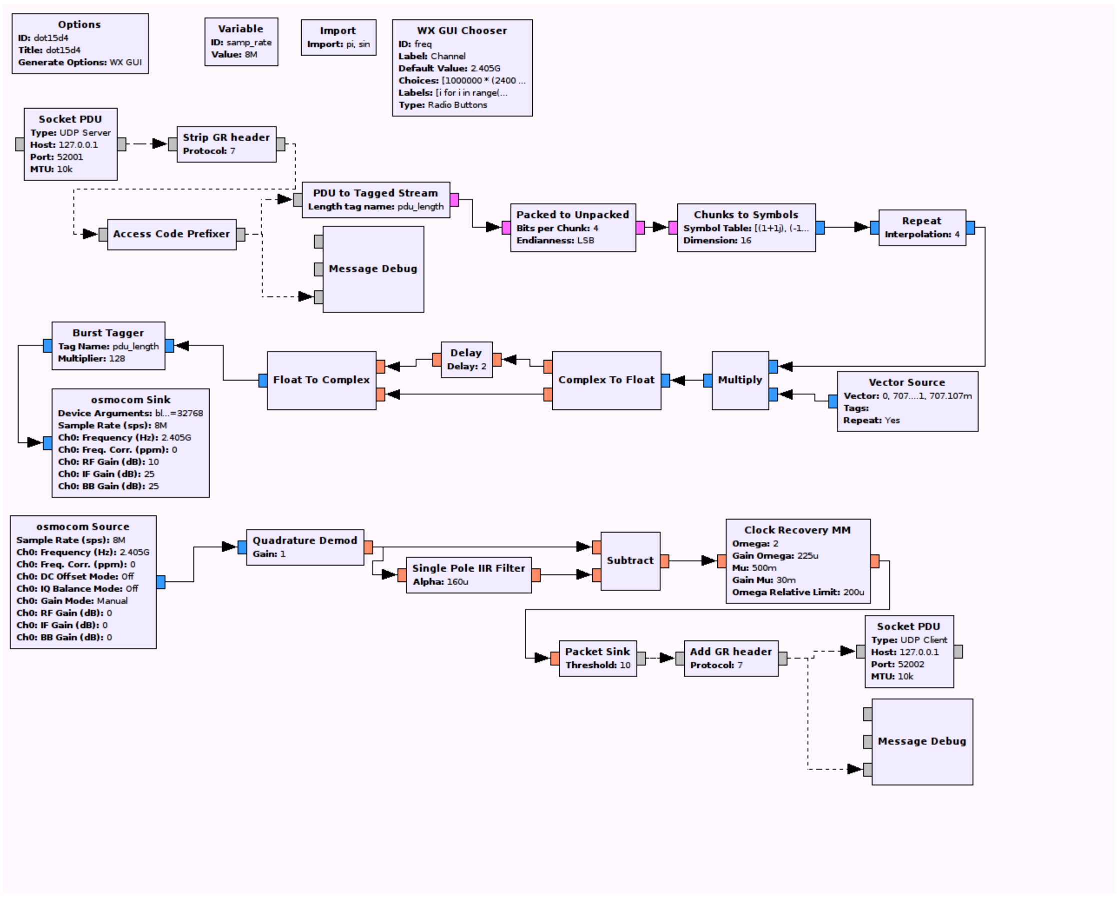
Channel hopping can be implemented using the AT86RF233 chip. This chip is capable of switching channels in 11 microseconds which is fast enough for communicating on a WirelessHART network. Channel hopping can also be built using the bladeRF. Since the 2.4 GHz frequency band is 83.5 MHz wide and the bladeRF supports a bandwidth of 28 MHz in optimal conditions, two bladeRF's are required to capture traffic on all 16 channels simultaneously without the need to actively hop in the hardware (change the bladeRF frequency). This is of course assuming that the computer connected to the bladeRF is fast enough to process this amount of data simultaneously.

Once it is possible to transmit network packets, the attacks described should be tested. These attacks should work if the implementation is done according to the specification. The impact of the attacks can be determined by testing the results of performing the attack.

Since the SDR (GNURadio) implementation of the IEEE 802.15.4 protocol allows for receiving and transmitting IEEE 802.15.4 compliant network packets, other protocols based on this standard can be evaluated as well (ZigBee, ISA100.A, 6LoWPAN, MiWi, etc.). As long as they do not include strict timing requirements like WirelessHART, this should be possible.

Appendix A
GNURadio 802.15.4 Transceiver

The figure below shows the implementation of the 802.15.4 transceiver used in this thesis. Behind each block is code which performs the actual signal processing.



Options
ID: dot15d4
Title: dot15d4
Generate Options: WX GUI

Variable
ID: samp_rate
Value: 8M

Import
Import: pi, sin

WX GUI Chooser
ID: freq
Label: Channel
Default Value: 2.405G
Choices: [1000000 * (2400 ...
Labels: [i for i in range(...
Type: Radio Buttons

Socket PDU
Type: UDP Server
Host: 127.0.0.1
Port: 52001
MTU: 10k

Strip GR header
Protocol: 7

PDU to Tagged Stream
Length tag name: pdu_length

Access Code Prefixer

Message Debug

Packed to Unpacked
Bits per Chunk: 4
Endianness: LSB

Chunks to Symbols
Symbol Table: [(1+1j), (-1...
Dimension: 16

Repeat
Interpolation: 4

Burst Tagger
Tag Name: pdu_length
Multiplier: 128

Float To Complex

Delay
Delay: 2

Complex To Float

Multiply

Vector Source
Vector: 0, 707....1, 707.107m
Tags:
Repeat: Yes

osmocom Sink
Device Arguments: bl...=32768
Sample Rate (sps): 8M
Ch0: Frequency (Hz): 2.405G
Ch0: Freq. Corr. (ppm): 0
Ch0: RF Gain (dB): 10
Ch0: IF Gain (dB): 25
Ch0: BB Gain (dB): 25

osmocom Source
Sample Rate (sps): 8M
Ch0: Frequency (Hz): 2.405G
Ch0: Freq. Corr. (ppm): 0
Ch0: DC Offset Mode: Off
Ch0: IQ Balance Mode: Off
Ch0: Gain Mode: Manual
Ch0: RF Gain (dB): 0
Ch0: IF Gain (dB): 0
Ch0: BB Gain (dB): 0

Quadrature Demod
Gain: 1

Single Pole IIR Filter
Alpha: 160u

Subtract

Clock Recovery MM
Omega: 2
Gain Omega: 225u
Mu: 500m
Gain Mu: 30m
Omega Relative Limit: 200u

Packet Sink
Threshold: 10

Add GR header
Protocol: 7

Socket PDU
Type: UDP Client
Host: 127.0.0.1
Port: 52002
MTU: 10k

Message Debug

Figure A.1: GNURadio Implementation of 802.15.4 Transceiver.

Bibliography

[1] Wikipedia, "Information security (CIA Triad)," 2015. [Online]. Available: https://en.wikipedia.org/wiki/Information_security

[2] "IEEE Standard for Information technology– Local and metropolitan area networks– Specific requirements– Part 11: Wireless LAN Medium Access Control (MAC)and Physical Layer (PHY) Specifications Amendment 5: Enhancements for Higher Throughput," *IEEE Std 802.11n-2009 (Amendment to IEEE Std 802.11-2007 as amended by IEEE Std 802.11k-2008, IEEE Std 802.11r-2008, IEEE Std 802.11y-2008, and IEEE Std 802.11w-2009)*, Oct 2009.

[3] Nederlands Elektrotechnisch Comité NEN, "Industrial communication networks - Wireless communication network and communication profiles - WirelessHART, NEN-EN-IEC 62591," pp. 1–446, 2010.

[4] AirbusDS, "Scapy-Radio." [Online]. Available: http://bitbucket.cassidiancybersecurity.com/scapy-radio

[5] E. Pietrosemoli, "Setting long distance WiFi records: proofing solutions for rural connectivity," *The Journal of Community Informatics*, vol. 4, no. 1, 2008.

[6] University Twente, "Security and Privacy in Mobile Systems." [Online]. Available: https://osiris.utwente.nl/student/OnderwijsCatalogusSelect.do?selectie=cursus&collegejaar=2014&cursus=201100023

[7] Wikipedia, "Digital-to-analog converter." [Online]. Available: http://en.wikipedia.org/wiki/Digital-to-analog_converter

[8] Open source project, "GNURadio." [Online]. Available: http://gnuradio.org/

[9] nuand, "bladeRF." [Online]. Available: http://nuand.com

[10] Ettus Research, "USRP B210." [Online]. Available: http://www.ettus.com/product/details/UB210-KIT

[11] "IEEE Standard for Information technology– Local and metropolitan area networks– Specific requirements– Part 15.4: Wireless Medium Access Control (MAC) and Physical Layer (PHY) Specifications for Low Rate Wireless Personal Area Networks (WPANs)," *IEEE Std 802.15.4-2006 (Revision of IEEE Std 802.15.4-2003)*, pp. 1–320, Sept 2006.

[12] Wikipedia, “Code division multiple access,” 2015. [Online]. Available: https://en.wikipedia.org/wiki/Code_division_multiple_access

[13] C. Langton, “All About Modulation,” 2002. [Online]. Available: http://people.seas.harvard.edu/~jones/cscie129/papers/modulation_1.pdf

[14] Recommendation, ITUTX, “ISO/IEC 7498-1: 1994,” *Information technology–Open systems interconnection–Basic reference model: The basic model*, 1994.

[15] P. Biondi, “Scapy,” 2011. [Online]. Available: http://www.secdev.org/projects/scapy

[16] A. Aring, “linux-wpan-next Kernel.” [Online]. Available: https://github.com/linux-wpan/linux-wpan-next

[17] Atmel Corporation, “Wireless MCU AT86RF233 Datasheet.” [Online]. Available: http://www.atmel.com/Images/Atmel-8351-MCU_Wireless-AT86RF233_Datasheet.pdf

[18] openlabs, “Raspberry Pi 802.15.4 radio.” [Online]. Available: http://openlabs.co/OSHW/Raspberry-Pi-802.15.4-radio

[19] A. N. Kim, F. Hekland, S. Petersen, and P. Doyle, “When HART goes wireless: Understanding and implementing the WirelessHART standard,” in *Emerging Technologies and Factory Automation, 2008. ETFA 2008.* IEEE, 2008, pp. 899–907.

[20] Linear Technology, “SmartMesh WirelessHART.” [Online]. Available: http://www.linear.com/products/smartmesh_wirelesshart

[21] K. Pister and L. Doherty, “TSMP: Time synchronized mesh protocol,” *IASTED Distributed Sensor Networks*, pp. 391–398, 2008.

[22] Winter, Jean Michel and Muller, Ivan and Pereira, Carlos Eduardo and Netto, João C, “Towards a WirelessHART Network with Spectrum Sensing,” in *Preprints of the 19th World Congress, International Federation of Automatic Control*, vol. 19, no. 1, 2014, pp. 9744–9749.

[23] P. Ferrari, A. Flammini, D. Marioli, S. Rinaldi, and E. Sisinni, “On the implementation and performance assessment of a wirelessHART distributed packet analyzer,” *Instrumentation and Measurement, IEEE Transactions on*, vol. 59, no. 5, pp. 1342–1352, 2010.

[24] L. Choong and M. Tadjikov, “GNURadio 802.15.4 demodulation.” [Online]. Available: https://github.com/septikus/gnuradio-802.15.4-demodulation.git

[25] L. Choong and M. Tadjikov, “Using USRP and GNURadio to decode 802.15. 4 packets,” 2008.

[26] L. Choong, “Multi-channel IEEE 802.15. 4 packet capture using software defined radio,” *UCLA Networked & Embedded Sensing Lab*, vol. 3, 2009.

[27] “PyCrypto - The Python Cryptography Toolkit,” 2010. [Online]. Available: http://www.dlitz.net/software/pycrypto/

[28] N. B. Truong, Y.-J. Suh, and C. Yu, "Latency analysis in GNU radio/USRP-based software radio platforms," in *Military Communications Conference, MILCOM 2013-2013 IEEE*. IEEE, 2013, pp. 305–310.

[29] Raspberry Pi Foundation, "Raspberry Pi Kernel Compilation Tools." [Online]. Available: https://github.com/raspberrypi/tools

[30] Open Source, "U-Boot." [Online]. Available: https://github.com/swarren/u-boot

[31] S. Raza, A. Slabbert, T. Voigt, and K. Landernas, "Security considerations for the WirelessHART protocol," in *Emerging Technologies & Factory Automation, 2009. ETFA 2009*. IEEE, 2009, pp. 1–8.

[32] A. Chakraborty and P. Banala, "An experimental study of jamming ieee 802.15. 4 compliant sensor networks."

[33] W. Pratt, M. Nixon, E. Rotvold, R. Pramanik, and T. Lennvall, "Suspending transmissions in a wireless network," Dec. 25 2013, eP Patent App. EP20,130,184,512. [Online]. Available: https://www.google.com/patents/EP2677699A1?cl=en

[34] N. Sastry and D. Wagner, "Security considerations for IEEE 802.15. 4 networks," in *Proceedings of the 3rd ACM workshop on Wireless security*. ACM, 2004, pp. 32–42.