Out-of-band transfer with Android to configure pre-shared secrets into sensor nodes

JOHANNES NILSSON
JAMEEL HABBOSH
MASTER´S THESIS
DEPARTMENT OF ELECTRICAL AND INFORMATION TECHNOLOGY
FACULTY OF ENGINEERING | LTH | LUND UNIVERSITY

Out-of-band transfer with Android to configure pre-shared secrets into sensor nodes

Johannes Nilsson
dat12jn2@student.lu.se

Jameel Habbosh
dat12jh1@student.lu.se

Department of Electrical and Information Technology
Lund University

Supervisors:
Martin Hell, Lund University
Einar Vading, Axis Communications
Henrik Springfors, Axis Communications

Examiner:
Thomas Johansson, Lund University

March 21, 2018

Printed in Sweden
Tryckeriet i E-huset, Lund

Abstract

Applications based on Wireless Sensor Networks are making their way into all kinds of industries. Today, they can do anything from off-loading hospitals by monitoring patients in their homes to regulating production lines in factories. More often than not, they perform some kind of surveillance and tracking. Thus, in most cases the information they carry is sensitive, rendering good encryption schemes suited for performance-constrained sensor nodes a valuable commodity.

As traditional encryption is not well suited for performance constrained environments, there are many new "lightweight" encryption schemes emerging. However, many of the popular up and coming schemes make the assumption of already having a pre-shared secret available in the sensor node beforehand which can act as the base for their encryption key. The procedure of configuring this pre-shared secret into the sensor node is crucial and has the potential of breaking any scheme based on that assumption.

Therefore, we have looked at different procedures of configuring this pre-shared secret into a sensor node securely, using nothing more than a smartphone to configure the sensor node. This would eventually eliminate the assumption of *how* the pre-shared secret got into the sensor node in the first place. We used an Arduino Uno R3 running an Atmega328p MCU as a simulation of a potential sensor node. Moreover, using a smartphone as the configuration device, we chose to base the communication on two types of OOB based side-channels; Namely, a visual-based using the flashlight and screen as well as audio-based, using the loudspeaker.

We concluded that using a smartphone as configuration device has its difficulties, although, in this specific environment it is still a viable choice. The solution can decrease the previous knowledge required by the user performing the configuration while simultaneously upholding a high security level.

The findings of this thesis highlight the fact that: technology has evolved to a point where the smartphones of today can outperform the specialized devices of yesterday. In other words, solutions previously requiring specialized hardware can today be achieved with much less "specialized" equipment. This is desirable because with less specialized equipment, it becomes easier to further develop and improve a system like this, increasing its viability.

Keywords — performance-constrained sensor nodes, out-of-band communications, pre key-exchange problem, smartphone-based transmitter

Popular Science Summary

Using your smartphone to blink secrets into sensor nodes

Have you ever wondered what would happen if somebody could access your refrigerator? Might seem silly, but how about your front door's lock? With the ever increasing connected society, you might have to think about these questions sooner rather than later. The establishment of our connected society is heavily dependent on sensor nodes. There is currently no rigid way of loading the necessary cryptographic keys into these sensor nodes. Now, to enable these sensor nodes to communicate securely, we have studied alternative ways of using your smartphone to transmit these keys to the sensor nodes.

In this thesis, we have shown alternative ways of using a smartphone to transmit cryptographic keys into sensor nodes. These alternative ways were achieved by using components not otherwise thought to be used for communication. For instance, we built prototypes that used the flashlight; the screen and the loudspeaker to successfully transmit the keys. Doing this we were able to make the transmission easy to use while at the same time upholding a high level of security.

Currently, the sensor nodes have many protocols available to use for secure communications. However, these protocols often lack information about how one should load the sensor nodes with the keys, to begin with. In essence, they provide you with the car but not the key to start it. This is a problem that needs a concrete solution.

The result of this thesis can be used as a guideline for further development of this type of solution. Our prototypes indicate that this type of solution is not only viable but can be secure as well. Using nothing more than a smartphone and small additions to the sensor nodes hardware.

Briefly, the prototypes are built using an Android-powered smartphone as "key-transmitting device" while the receivering "sensor node" is equipped with a microphone or a photo-transistor. The additions to the receiver enable detection of both light and sound waves sent from the smartphone. Then, using the smartphone, the user is able to transmit data by blinking with the flashlight or screen; or sending tones with the loudspeaker, which the receiver interprets.

Acknowledgement

To begin with, I would like to express my gratitude to Axis Communications for sponsoring this Master's thesis work and providing a great workplace. Then, I would like to thank our supervisors Martin, Einar and Henrik for the support and navigation needed to complete this thesis work. Finally, I would like to send a special thanks to my friends, family and my lovely fiancee.

Johannes Nilsson

I would like to thank Axis Communications for the opportunity of researching this Master's thesis, for their hospitality and for the opportunity to work under the freedom of supervision. I would like to thank my supervisors at Axis, namely Einar Vading and Henrik Springfors and my supervisor at LTH, namely Martin Hell. Their combined insightful guidance and support was tremendously appreciated. Lastly, I would like to express my humble gratitude to my family and friends, their support during this work has been truly blissful.

Jameel Habbosh

Table of Contents

List of Figures

List of Tables

List of Abbreviations

ADC — Analog-to-Digital Converter
AES — Advanced Encryption Standard
API — Application Programming Interface
ASCII — American Standard Code for Information Interchange
ASK — Amplitude Shift Keying
BOD — Brown Out Detector
CRC — Cyclic Redundancy Check
DFT — Discrete Fourier Transform
EEPROM — Electrically Erasable Programmable Read-Only Memory
FFT — Fast Fourier Transform
FSK — Frequency Shift Keying
GPS — Global Positioning System
IEEE — Institue of Electrical and Electronics Engineers
INT — Interrupt
IOT — Internet of Things
ISR — Interrupt Service Routine
JVM — Java Virtual Machine
MCU — Micro Controller Unit
ME — Manchester Encoding
NFC — Near Field Communications
NRZ — Non Return to Zero
NTP — Network Time Protocol
OOB — Out-of-Band
PCB — Printed Circuit Board
PCINT — Pin Change Interrupt
PWM — Pulse Width Modulation
RTOS — Real Time Operating System
SRAM — Static Random Access Memory
TWI — Two Wire Interface
VLC — Visual Light Communication
WDT — Watch Dog Timer
WSN — Wireless Sensor Network

Chapter 1
Introduction

By now, you have probably heard about "Internet of Things", one way or another. Its growth in popularity is evident, recent studies [1] suggests that by 2020 there will be over 46 billion connected *things*. These, so called things, can practically be any type of device, as long as it has a way of accessing the Internet. Since the devices are increasingly integrated in all parts of society, it is quickly becoming a part of life, whether you like it or not.

Contributors to the fast growth of connected things are advances in the fields of embedded computing and network communications. These advances enable the engineers of today to create sophisticated Wireless Sensor Networks, often called the "eyes and ears of IoT". WSN:s are networks that consist of, up to thousands of small sensor nodes. These nodes are interconnected wirelessly and are heavily performance constrained. Their objective is often to measure different types of physical attributes in their environment with the use of onboard sensors, translate the information into digital form and transmit it to the network. There are even some cases where the sensor has the ability to control the environment when it is equipped with actuators. This technology seems to have endless potential and is today used in many types of applications, such as, military, environmental monitoring, logistics, robotics, healthcare and much more [3][44].

As these technologies steadily are given extended responsibility in critical systems, the repercussions of potential malicious attacks is increasing equally. Gaining access to one of these systems could possibly mean that an intruder, for instance, would have the power to control an entire city's electrical grid or even shut down critical machines used in healthcare applications. To mitigate these risks from materializing, continuous research is crucial to maintaining both the technology and frankly, society itself.

1.1 Background

Axis Communications is always looking into new ways to extend the functionality of their access control system. One possible extension is to detect when an intruder tries to break in via a window. This can be detected with the help of a so called *Glass Break Sensor (GBS)*, which shares many characteristics of a resource-constrained wireless sensor node. The sensor node should be able to detect glass breaking and send information to the base station of the access control system

wirelessly. See Figure 1.1. This base-station is responsible for many other functions as well, such as communicating with other types of devices. In other words, the glass break sensors is a small sub-system of the larger access control system.

Now, due to the nature of this system, it is very important to provide high levels of security. Not only is it a security-centered system itself, it also has the ability to be connected to the Internet, making it vulnerable to outside attacks. Although the information transmitted from the glass break sensor might be considered as non-sensitive, a secure and encrypted connection is still prefered. Upholding high security standards of the overall system is important, as a system is only as strong as its weakest link.

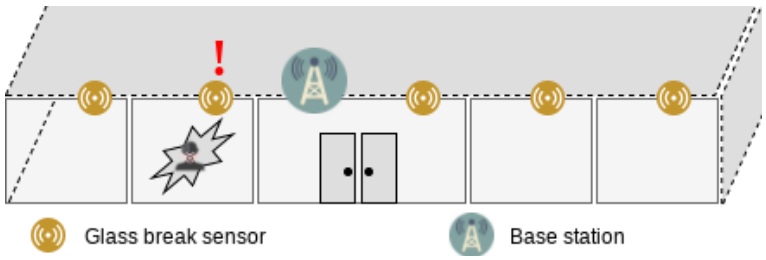

Figure 1.1: Use-case: intruder breaking a window and connected sensors alarming a base-station.

Naturally, this use case has its roots in a more general problem. This general problem is that conventional security mechanisms often require large amounts of resources within the areas of computational power and communications. As a result, these mechanisms cannot be applied to performance constrained wireless sensor nodes. Consequently, extensive research on novel mechanisms that tries to achieve the same level of security but are less dependent on resources has been conducted. This research has resulted in the proposal of a lot of new "lightweight" security schemes, such as [17][11][43][25][12].

In addition, mechanisms based on symmetric encryption have gained a lot of attention, almost all of the most promising up and coming schemes rely on some type of symmetric key, also known as a *pre-shared secret*. Most of the time, these schemes rely on that the symmetric key is created on a machine with more resources available and then transported and stored onto the sensor node; Put simply, offloading the computational demands of key generation to a more powerful machine. However, they do not specify a procedure for loading the pre-shared secret from the machine into the sensor nodes.

For instance, Eschenauer et al. [17] describes their configuration procedure merely as: "loading of the key ring into the memory of each sensor". Similarly, Chan et al. [12] describes their process as: "... and store them into the node's key ring". Naturally, the method of configuring this pre-shared secret into the device impacts the overall security of the system. If the pre-shared secret is compromised, so is the system, no matter how secure the protocol using this secret is. Basically, it does not matter how advanced a vault lock might be, it is always possible to open it with a (compromised) key.

1.2 Scope

Utilizing out-of-band transfer is an alternative approach that has previously often been discarded due to the fact that additional interfaces on each of the sensors are required. These interfaces have been considered to influence the manufacturing costs of the sensors too much [10]. However, as the prices of electrical components have been greatly reduced within the last decades, an additional interface could have a smaller impact on the manufacturing cost of the node [19]. In addition, it may yield a more user-friendly configuration process compared to conventional wired processes, as well as upholding a high level of security.

The aim of this Master's thesis work is to investigate whether out-of-band methods are a viable solution for configuring the pre-shared secrets into a resource-constrained wireless sensor. The proposed solution must have as few prerequisites as possible to keep down cost and at the same time promote usability. The tests performed on the prototypes will be based on the categories ease of use, security, power consumption and resource demands.

It mainly focuses on out-of-band methods that are visual- and audio-based. Prototypes utilizing both of these out-of-band channels will be produced for the opportunity to further test, analyze and build upon. The results of this project will also be used as a collection of measuring data and provide guidelines for the development of the wireless glass break sensor. Hopefully, the data retrieved from these prototypes along with the report can act as a source of information to anyone trying to achieve any similar solution.

This thesis' scope does not include implementing another "lightweight" protocol for establishing secure connections between wireless sensors and base stations. The line is drawn by having placed the pre-shared secret into the wireless sensor node. In turn, the "lightweight" protocols mentioned in Section 1.1 could then use that key. Implementing an entire protocol like such will require numerous sensors and more time. Therefore, this should be considered as future work.

1.3 Related Work

Rapid development within IoT and WSN have resulted in a lot of research being conducted within these areas, some of which distinctively deal with the problems regarding security. Even if there are many aspects of security to consider from a system perspective, almost all of them have a common divider. This common divider is the fact that they depend on some type of cryptographic key. This means that some of the security research within the WSN context is concerned with loading the pre-shared secrets (cryptographic keys) into the sensor nodes, as described in Section 1.1.

To begin with, in 2005 *Seeing-is-Believing (SiB)*, [34], introduced a procedure to achieve authentication between two devices with no previous knowledge of each other, utilizing a visual-based OOB channel. The devices who want to establish a secure connection between each other are required to have either a camera and/or display to be able to use the procedure proposed in *SiB*. They introduced a user scenario where a user wants to connect to a wireless public printer. The printer should then have its public key physically imprinted on it as a barcode. Then, the

connecting user can take a picture of the barcode with their device and extract the public key. Now the user verifies the authenticity of the public key sent to their device wirelessly by the printer by comparing the two public keys. The one sent wirelessly with the one extracted from the barcode. As mentioned earlier, this system is not limited to this specific use case but can be applied to any devices that meet the interface requirements.

A year later, Michael T. Goodrich et al. produced a system called *Loud-and-Clear (L&C)*, [23]. Even though *L&C* is quite similar to *Seeing-is-Believing* the major difference is that they changed the OOB channel from visual-based to an audio-based one. Naturally, now requiring the devices to have a display and/or a loudspeaker instead. Using a Text-To-Speech engine they managed to enable the device to generate a "human-understandable" sentence, which was uniquely based on the key. This sentence was then played through the loudspeaker of the device. Next, the user listens to the sentence spoken by the device they want to connect to and verifies whether it is authentic by comparing the sentence and the already stated key. Looking at the same use case as in *SiB*: a public printer can have the public key (sentence) physically printed on the device somewhere. Then, with the help of a button the device can be made to *speak* the key as well. Now, in the same manner as *SiB*, the user verifies that the printed public key matches the one heard through the loudspeaker.

In 2007 C. Kuo et al. presented *Message-in-a-Bottle (MiB)*, [29]. They introduced a novel procedure to securely load the pre-shared secret into a sensor node. They utilized the node's already present wireless interface. To protect their key establishing communication from outside interference, they placed both the key-transmitter and the key-receiver (sensor node) inside a Faraday cage. In other words, this paper proposes a solution for configuring pre-shared secrets into a wireless sensor node, yet not through an OOB channel. Since a perfect Faraday cage is very difficult to manufacture, the researchers had to consider any leaking signals. To deal with the imperfect Faraday cage they also had a device outside the cage that transmits non-sensitive information on high power, acting as a jamming device. This makes it very hard for an eavesdropper to interpret any leaking signals.

Furthermore, Matthias Gauger, Olga Saukh and Pedro José Marrón proposed to utilize a side-channel to solve configuration problem in *Enlighten me!* published in 2009, [21]. They predicted that future sensors would be delivered pre-programmed by the manufacturing company without any wired interfaces creating a demand for wireless transferring capabilities. They proposed a system which used a visual-based OOB interface to transmit and receive secret information in the configuration procedure. They presented solutions based on both dedicated hardware (specialized hardware) and a Pocket-PC as device acting as a "key transmitter".

Lastly, in 2011 *Rainbow Crypt*, [7], introduced a solution to provide secure communications between resource-constrained devices. The communication is also based on a visual-based OOB channel. This system requires the devices to have a small color screen and a camera as prerequisites. To communicate, the transmitter and receiver have to be placed in direct contact with each other. On the one hand, having the components so close to each other makes communication difficult, since the camera cannot focus on objects too close. On the other hand, with this setup, it

is impossible to eavesdrop on the channel. Their experiment resulted in a severely limited communications channel for general purpose but viable in the context of transmitting pre-shared secrets.

1.3.1 Conclusion of Related Work

Even if all the related work mentioned have slightly different purposes. They all are dependent on the same thing, achieving some type of OOB based communication.

Firstly, *SiB* and *L&C* does not use their OOB based side-channel to actually transmit any new data. They use it more as a secondary channel to provide the same type of information twice. Consequently, it relates to this thesis by generally merging the concepts of OOB communication and security together. From this, we have drawn inspiration as to how we could improve this as well as moving the actual communication to the OOB based channel completely.

Secondly, *MiB* and *Enlighten me!* have a very similar scope compared to this thesis. They are also concerned with loading secret information securely into resource constrained sensor nodes. Remarkably, parts of this thesis' scope are mentioned as "future work" in one way or another in both of those papers. *MiB* mentions the possible benefits of moving their system's communication from wireless to an OOB based channel. While *Enlighten me!* discusses extending their findings onto phones as well. Briefly, they have already built a similar system compared to this thesis' aim. The difference between them is that *Enlighten me!* are mainly focused on a specialized device used for transmitting the pre-shared secrets. Furthermore, *Enlighten me!* acts as a baseline for further development and comparison of results.

Thirdly, as we entertained the idea of using colors in our visual based OOB channel, to begin with, *Rainbow Crypt* provided some valuable insights into what would be required. However, as for now, introducing colors into the visual-based solutions should be seen as future work.

Lastly, combining all of this research has given us many valuable insights by looking at their successes and failures. In addition, placing the related work in chronological order highlights how advances in technology have increased the viability of a system utilizing OOB as either a primary or a secondary communication channel. Compared to today, as our last reported related work was published 2011, it has been seven more years of technological advances.

1.4 Contributions

Although the work have been divided between both authors, we have been equally involved in the work done. We have used each other as a sounding board for both thoughts, ideas and assistance. The thesis has been divided into four larger parts of work, namely:

1. Implementing the application used by the key-transmitter (smartphone) which transmits the pre-shared secret using a visual-based OOB channel.
2. Implementing the firmware of the receiver, utilizing the visual-based OOB channel.

3. Extending the application of the key-transmitter with capabilities of utilizing an audio-based OOB channel.
4. Implementing the firmware of the receiver, utilizing the audio-based OOB channel.
5. Writing the thesis report.

1.5 Outline

The rest of this thesis will be organized as follows:

- *Chapter 2 Preliminaries:* Introduces some brief background information regarding relevant technologies and concepts. Providing a quick way for the reader to brush up on the subjects if needed.
- *Chapter 3 Architectural Overview:* Provides an overview of the main components used to build the prototypes. Also, a further explanation of how the scope of the thesis affected the choice of components and why we decided to use them.
- *Chapter 4 Implementation:* Goes more into detail on how the software and firmware of the prototypes were created. Based on the assumptions and choices of hardware explained in the previous chapter.
- *Chapter 5 Results:* Presents the results and briefly explains how the tests were conducted on the prototypes.
- *Chapter 6 Discussion:* Puts the entire solution and the results presented in the previous chapter into a wider perspective, discussing what one can gain by utilizing the methods presented in this thesis.
- *Chapter 7 Conclusion:* Concludes whether the thesis was of success or not.
- *Chapter 8 Future Work:* Presents possible improvements and subjects we think should be further researched.

Chapter 2
Preliminaries

This chapter is dedicated to provide the reader with a brief summary of the key technologies and concepts mentioned throughout this report.

2.1 Digital Communication

Digital communication is a wide area that contains many subjects. This section will only mention subjects which relate to the scope of the thesis. The reader is encouraged to read [32] for more details about these and many other subjects regarding digital communications.

2.1.1 Out-of-Band Channels

There are several types of ways to achieve OOB communication. Generally they fall into one of three categories, namely *audio*, *visual* or *tactile*. However, it should be noted that OOB communication is not limited to only these types. Since as long as the communication uses a "side-channel" compared to the "main-channel" used for communication, then it could be classified as an OOB channel.

Naturally, there are both benefits and drawbacks in utilizing an OOB channel for communication. On the one hand, from a security point of view, an OOB channel could provide strong authentication and integrity of the messages being transmitted. In addition, OOB channels are generally not regulated, meaning they are free to use, compared to the heavily regulated radio-wave spectrum. On the other hand, eavesdropping on the channel is still a problem. For example, Alice and Bob are having a conversation. Alice is sure that she is talking to the real Bob since she can verify that he is standing in front of her and vice versa. Similarly, she is certain the spoken message has not been modified since it is coming directly from Bob's mouth. However, there is always a risk of Eve hiding somewhere close by eavesdropping on the conversation.

Moreover, OOB channels typically cannot support massive throughput, have a short range of communication and tend to require extra or special interfaces to be able to utilize the channel.

Visual Based Channels

Visual based OOB channels are often called visual light communication, abbreviated VLC. This type of OOB channel uses visible light as its channel of communication. Hence, light with wavelengths between 390 and 700 nanometers. Visual light is emitted by a source, often an LED, and received by a target, using a photo-diode or photo-transistor. This type of system often requires the source and target to be quite close to each other since the propagation of light suffers from high degradation. It is also often high levels of ambient noise. However, although this might be seen as a drawback it can actually be turned into a good thing. As mentioned previously, eavesdropping still presents a problem when utilizing OOB. Therefore, having a low range of communication with high signal degradation results in it being harder to eavesdrop on the channel. The eavesdropper has to be very close and therefore also risk being detected.

Audio Based Channels

Often consisting of a loudspeaker as transmitter and microphone as a receiver. What makes this type of OOB method somewhat special compared to the other types of OOB channels, is that today, many devices such as smartphones already contain the hardware needed to utilize this type of communication.

2.1.2 Digital Data Transmission

Digital data is a sequence of `0`'s and `1`'s. However, to be able to transmit this data over a medium some type of signal is necessary. In other words, signal processing of the digital data has to be done to produce such a signal. The kind of signal processing that is needed depends on what type of medium the signal is supposed to be sent through.

To separate the two, the process of converting digital (or analog) data into a digital signal is called *line encoding*. While converting digital (or analog) data into an analog signal is called *modulation*.

Line Encoding Techniques

There are several techniques available to achieve line encoding of a binary data stream. The ones most relevant to this thesis are:

Non Return To Zero which is the simplest form of line encoding. The binary stream of digital data is directly represented as either *positive level* if the bit to be transmitted is `1` or *negative level* if the bit were `0`. An example of NRZ-encoded data is illustrated in Figure 2.1.

NRZ is a pretty straightforward technique which uses as little bandwidth as possible. Nevertheless, it is also highly dependent on a separate clock signal between the transmitter and receiver to be able to synchronize. For instance, if there is a part of the binary data stream which contains multiple bits of the same kind in a row, it can become hard to discern between how

many bits were actually sent. Due to the receiver only seeing one signal for a prolonged time.

Manchester Encoding which was invented at the University of Manchester in the late 1940s. Manchester Encoding, *ME*, is a technique which provides a *self-clocking signal*. This means that a separate clock signal otherwise used for synchronization is not required. In fact, it is embedded in the signal itself. ME works by encoding a `0` from the bitstream as a pulse from *positive level* to *negative level*. While a `1` is encoded as a pulse from *negative level* to *positive level*. The bits are, in other words, encoded as *transitions* from either high to low (`0`), or low to high (`1`). An example of data encoded with ME is illustrated in Figure 2.1.

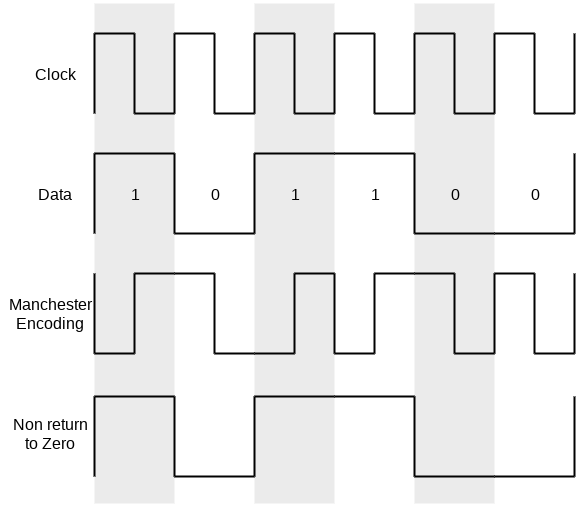

Figure 2.1: Data encoded with the ME and NRZ techniques.

Since good synchronization can be hard to achieve, having a self-clocking signal is a major advantage. Another advantage of ME is that the signal produced does not contain any long sequences of the same signal level. This means that the situation where the receiver easily can lose synchronization with the transmitter never happens. Conversely, using ME does add some overhead because each bit is encoded with two states the signal uses twice the bandwidth.

Pulse Width Modulation is easily confused as it is more often referred to when talking about controlling the power supply to electrical circuits. Regardless, PWM can be applied to line encoding as well. It works by mapping a certain width of a pulse to represent a specific symbol. The symbol is not restricted to the binary case, [32]. By having a short pulse representing bit `0` instead of the absence of a pulse the implementation of the receiver becomes less complex as you no longer have to differentiate between a `0` and *nothing*.

As stated previously, PWM is not restricted to the binary case. Moreover, it is easy to include more symbols. Instead of having only two different widths

of pulses representing `0`'s and `1`'s, one could implement several widths where each one represents a different symbol.

For example, a pulse of 10ms can be mapped to the bits `00`, 20ms to `01`, 30ms to `10` and finally 40ms to `11`. Which results in a 4-ary implementation of PWM. This is illustrated in Figure 2.2.

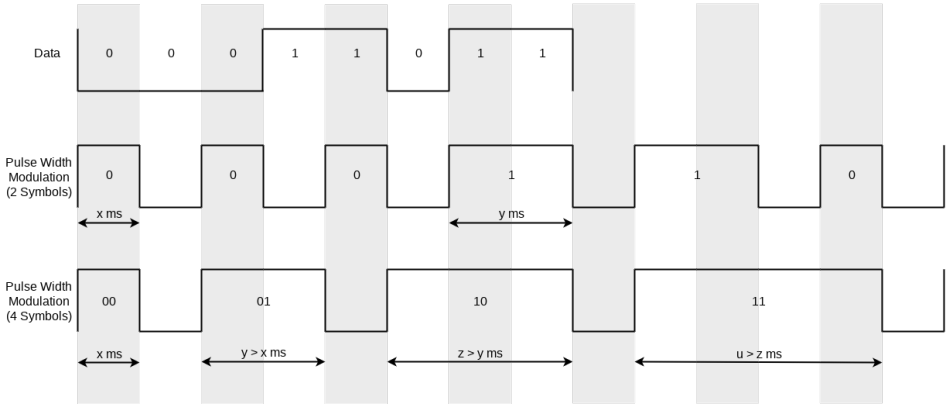

Figure 2.2: Data encoded with both 2-ary and 4-ary PWM.

Modulation Techniques

The three fundamental modulation techniques are often considered to be: Amplitude Shift Keying (ASK), Frequency Shift Keying (FSK) and Phase Shift Keying (PSK). As ASK and FSK are the ones related to this thesis, only they will be further explained.

Frequency Shift Keying works by having a *carrier-frequency*, which acts as "base" for the signal. The signal is often a sinusoidal but the technique works with other types of pulse-shapes as well. Furthermore, based on the data to be sent, the *carrier-frequency* is altered creating a way for the receiver to detect data.

FSK can easily be modeled to any symbol-size. Often denoted (M-ary FSK), where M denotes how many symbols are being modulated. For example, binary FSK, BFSK or 2-ary FSK is the case where the symbol-size is equal to the bit-size as there are only two symbols, namely `0` and `1` which in turn are modulated by two different frequencies. Figure 2.3 illustrates how FSK can modulate digital data.

Amplitude Shift Keying ASK is similar to FSK in the sense that it also uses a *carrier-frequency* as well as that it translates into more symbols easily. The difference is that instead of changing the frequency depending on the data to be sent, the changes are made to the amplitude of the signal. ASK is easier than FSK to implement, both transmitter and receiver are less complex. However, ASK is much more sensitive to noise than FSK. Figure 2.3 illustrates how ASK can modulate digital data.

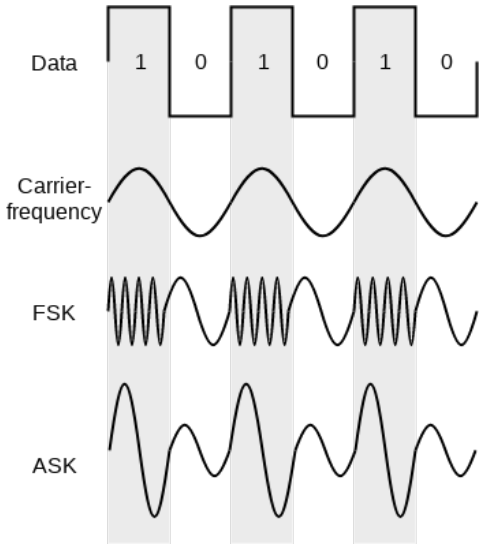

Figure 2.3: Example of how ASK and FSK can change the carrier-frequency, with regards to the data.

2.1.3 Discrete Fourier Transform

The Discrete Fourier Transform is the same as the Continuous Fourier Transform, except the signal is only known at N points which are separated by a specific sample time T. Put simply, the signal is *digital* instead of analog. Now, DFT is a mathematical tool that provides a way to determine the frequency content of a (digital) signal sampled in the time-domain, [36]. The DFT can be expressed as:

$$\underbrace{F_k}_{\text{k-th frequency bin}} = \underbrace{\sum_{n=0}^{N-1}}_{\text{Evaluating at n of N samples}} x_n * e^{-\frac{j2\pi kn}{N}} \tag{2.1}$$

Expanding the summation in Equation 2.1 renders the following:

$$F_k = x_0 * e^{-\frac{j2\pi k0}{N}} + x_1 * e^{-\frac{j2\pi k1}{N}} + \ldots + x_{N-1} * e^{-\frac{j2\pi kN-1}{N}} \tag{2.2}$$

Now, with the help of Eulers formula:

$$e^{jx} = cos(x) + jsin(x) \tag{2.3}$$

Equation 2.2 can be expressed as:

$$F_k = x_0[cos(-\frac{2\pi k0}{N}) + jsin(-\frac{2\pi k0}{N})] + \ldots + x_{N-1}[cos(-\frac{2\pi kN-1}{N}) + jsin(-\frac{2\pi kN-1}{N})] \tag{2.4}$$

which when evaluated become:

$$F_k = A_k + jB_k \tag{2.5}$$

By plotting Equation 2.5 on a complex plane, we can extract information like the magnitude of the vector as seen in Figure 2.4. This is accomplished by using Pythagoras' theorem:

$$Mag = \sqrt{A_k^2 + B_k^2}$$

This magnitude corresponds to the amplitude of the sinusoidal at that frequency bin, illustrated in Figure 2.5.

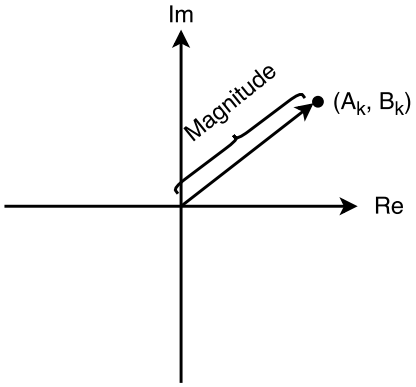

Figure 2.4: The result of a DFT calculation (Equation 2.5) plotted in a complex plane.

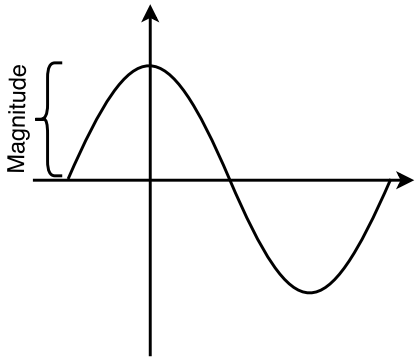

Figure 2.5: Connection between the result of the DFT from the frequency-domain (Figure 2.4) to the time-domain.

Fast Fourier Transform

Now, closely related to the DFT is the Fast Fourier Transform. This is not a new form of Fourier transform but a set of algorithms that provides a more computationally efficient way to calculate the DFT itself. For instance, in DFT, if you have one hundred samples ($N = 100$), you are going to end up with a hundred frequency bins. For each of these frequency bins, you will have to calculate a hundred values. This results in a complexity of $\mathcal{O}(n^2)$. The different FFT-algorithms decreases the complexity from $\mathcal{O}(n^2)$ to $\mathcal{O}(n \log n)$. The FFT-algorithms are rather complicated algorithms and therefore only the key details of the algorithms are described.

The key idea is to take advantage of the periodic and symmetric nature of the sinusoidal. It starts by dividing the sampled signal points into even and odd indexed summations:

$$F_k = \sum_{m=0}^{\frac{N}{2}-1} x_{2m} * e^{-\frac{j2\pi km}{N/2}} + e^{-\frac{j2\pi k}{N}} * \sum_{m=0}^{\frac{N}{2}-1} x_{2m+1} * e^{-\frac{j2\pi km}{N/2}} \tag{2.6}$$

Looking more closely at the exponential $e^{-\frac{j2\pi km}{N/2}}$ one can see that it is subject to symmetry identity which mean that if $k = 0, \ldots, N$ the result of calculating

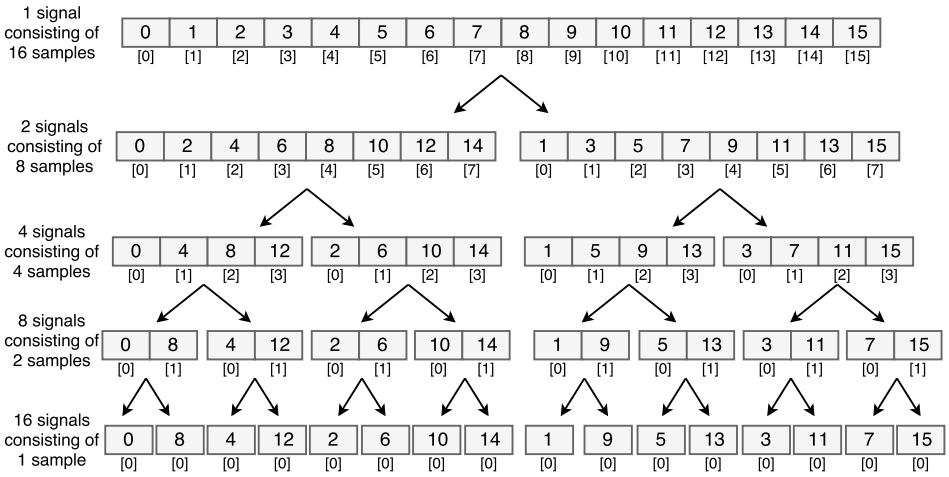

Figure 2.6: FFT decomposition, splitting a signal of 16 samples in halves separating the even and odd indexed samples [41].

$k = 0, \ldots, \frac{N}{2}$ is the same as $k = \frac{N}{2}, \ldots, N$. This means that the number of calculations needed is halved, since only one of the halves is needed.

Now, this process can be repeated which means that the summations are halved each iteration creating a final complexity of $\mathcal{O}(n \log n)$. The process of splitting the array of signal samples is illustrated in Figure 2.6. However, as a consequence of the number of samples taken, it should be based on the equation $N = 2^x$ to achieve the largest amount of efficiency.

2.2 Cryptography

As this thesis has its root in the area of security, this section provides a brief summary of the core concepts. This is primarily done to reintroduce the reader to Alice, Bob and Eve while getting a hang of the terminology again. And so, the objectives of cryptography is often described as providing functionality to satisfy four major goals [35][22]. These goals are:

Authentication which is concerned with the process of proving an entity's identity. If Alice and Bob wish to communicate, without previous knowledge of each other, then there has to be a mechanism that proves to Bob that Alice is who she claims she is and vice versa. Authentication is needed for both entities, namely Alice and Bob, but also for the data itself (messages). Therefore, the subject can be subdivided into *entity authentication* but also into *data origin authentication*, where the latter also provides integrity.

Confidentiality is the process of proving that another entity cannot read the messages that are being sent over the communication channel. If Alice and Bob communicate, there has to be a mechanism that protects the messages that they are sending from being read by anyone else.

Integrity which is the process of proving that the message that has been sent over the channel has not been altered in any way. If Alice sends a message to Bob, then there need to be a way of proving that the message Alice sent is *exactly* the same as the one Bob received.

Non-repudiation is the process of proving that a sender really sent a message. If Alice sends a message to Bob, then there has to be a way of proving that Alice did, in fact, send that message, so that Alice cannot claim that she did not send the message.

These are only the four major goals which many other smaller goals can be derived from, however, those will not be further discussed.

2.2.1 Encryption

Encryption is used to achieve confidentiality, by providing a way to scramble the message to be sent before the transmission occurs. If Alice wishes to send a message to Bob, she first *encrypts* the message based on a private-key before she sends it. The transmitted message is scrambled and even if another party gets a hold of it, they cannot read the message without knowing the private-key used to *decrypt* it. Bob, however, has the same private-key used to encrypt and therefore, can transform the scrambled message into readable form again.

Symmetric Encryption

Also known as *single key encryption*. This is due to the fact that it is based on a scheme that uses the same key both for encrypting and decrypting messages. As a consequence, the shared private-key, that both parties are supposed to use when encrypting/decrypting, has to be distributed safely somehow. This is referred to as the *key distribution problem*.

Asymmetric Encryption

Commonly referred to as *public-key encryption*, is a scheme that uses a key-pair for encryption. Each party has its own set of keys, one private and one public. As the name implies, the private key has to be kept a secret, however, the public key can be freely distributed. The private key is used for decryption and the public key is used for encryption. Let's say Alice wants to send an encrypted message to Bob, she then uses Bob's public key to encrypt the message and then sends it to Bob. Upon arrival, Bob can decrypt the message using his private key.

Comparison

There are many advantages and disadvantages to both of these approaches, the most important ones are:

- A symmetric key is generally much shorter than the keys in the key-pair of its asymmetric counterpart. The difference in size is often a factor of ten or more. The National Institute of Standards and Technology, (NIST),

recommends [6] using at least 128 bits for a symmetric key and 3072 bits for the factoring modulus. These lengths are deemed to be secure until 2031 *and beyond*.

- In symmetric encryption the shared private-key must be kept *private* on both ends, in other words, if either one of the concerned parties' key is compromised, then messages in both directions are compromised.
- Symmetric encryption schemes are not as computationally demanding as the asymmetric encryption schemes.
- Asymmetric encryption also requires some form of authentication scheme to prove the legitimacy of the public key. Further explained in Section 2.2.2.

2.2.2 Digital Signatures

Digital signatures try to uphold the same functionality as their analog equivalent. Which is to bind a signature to an identity, in such a way that if Alice presented her signature to Bob, he would not be able to deny Alice's true identity. In other words, digital signature schemes provide authentication and non repudiation but are not restricted to them.

In a digital signature scheme, there is most likely three different algorithms in use:

1. **Key generation algorithm** used to derive a private key K_{pr} and map it to a public key K_{pu}.
2. **Signing algorithm** which is responsible for producing a signature s, based on the message, m, and the K_{pr} from step 1.
3. **Verification algorithm** that has the functionality of either rejecting or accepting s, based on K_{pu}, m and s.

2.2.3 Hash Functions

Hash functions are a mathematical one-way function, that can compute a *hash-value* of a message. Given this hash-value, it is unfeasible to derive the original message. Hash functions are often used with digital signatures. If the message is very long, it might be better to compute a hash value based on the message and sign the hash-value instead.

Hash functions are tightly related to *Hash-based Message Authentication Codes*, [35], which provide data origin authentication as well as data integrity.

2.3 Wireless Sensor Networks

Combine multiple information collecting wireless nodes containing sensors and actuators, interconnecting each one of them to a base station and you get a Wireless Sensor Network, WSN. These networks are very versatile and can be deployed practically anywhere, anyhow. The nodes can be planted either manually by placing them in selected locations or randomly by, for instance, throwing them

out of a flying airplane. WSN:s are used everywhere, from home-automation to detection of possible threats on a battlefield.

The main purpose of the networks is to collect, process and pass the data along to a base station. The base station is a non-constrained device that acts as a storage unit of the data. Furthermore, the Internet connection in the WSN is often provided by the base station as well. The network topology of a WSN is often one of *star*, *mesh* or *tree*, depending on the environment where the network is to be deployed. Figures 2.7-2.9 illustrates the different network topologies mentioned.

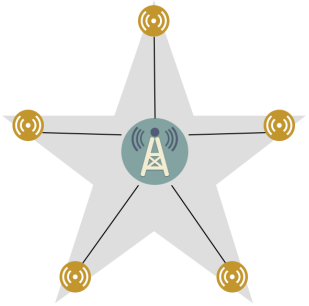

Figure 2.7: Star-shaped network topology.

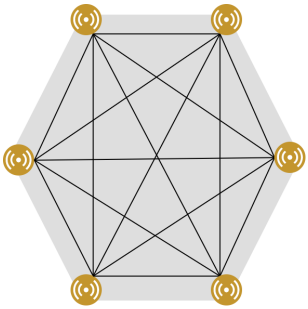

Figure 2.8: Mesh-shaped network topology.

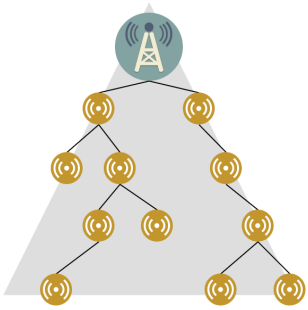

Figure 2.9: Tree-shaped network topology.

Even though the wireless sensor nodes in a WSN potentially could be *any* type of device, they are generally thought of as low-powered devices that consist of: one or more sensors; a processing unit; a power-supply; a transceiver and possibly an actuator as well. This is illustrated in Figure 2.10. Why they are generally thought of as this is because WSN:s are designed to contain up to thousands of nodes. Thus, if the nodes are expensive, the cost of deploying a WSN would be very high.

The fact that the nodes in these networks are so heavily resource constrained has had the undesired consequence that much of the prior research regarding networking and communication are not applicable to WSN based systems. Since previous research had vastly different underlying assumptions regarding the capabilities of both the network and its nodes, most previous distributed networks have assumed the availability of wired connections. These assumptions imply that the sensor devices have unlimited power and user interfaces such as visual screens as well as mice and keyboards, [42].

This highlights the need of continued research within WSN, not only in terms of security but within all aspects.

2.4 Arduino

Arduino has started to become one of the most popular choices when it comes to building and prototyping the nodes of a WSN [18]. This is mainly because they offer a platform which can be customized with a variety of microcontrollers that are

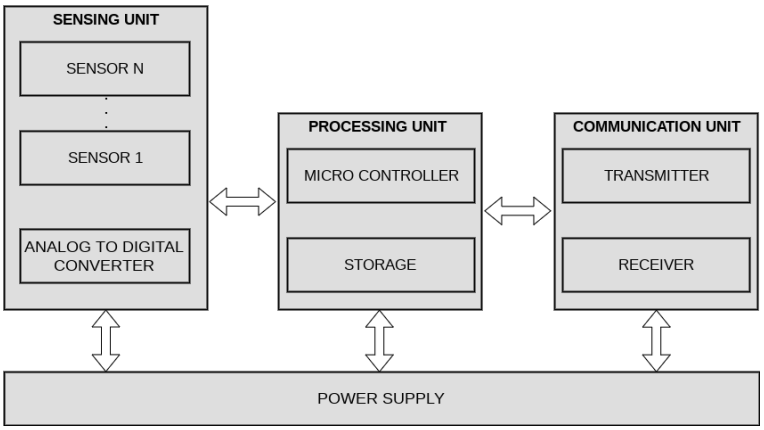

Figure 2.10: Basic building blocks of a sensor node.

different both in terms of physical size, computational power and circuit-specific features. They are developed to be easy to learn, build and maintain, which also leads to smooth and simple prototyping. The success of Arduino has resulted in different Arduino boards being used in much research regarding WSN:s, such as [9][20][47].

Arduino Uno R3

Out of all the possible boards, *Arduino Uno R3* is one of Arduino's most popular. It is based on a Atmega328p microcontroller which among other things has: a 10-bit ADC; 32kb flash memory; 2kb SRAM and 1kb EEPROM. Moreover, it has: 14 digital and 6 analog input/output (I/O) pins; a 16Mhz quartz crystal oscillator; a usb interface for easy programming and a power jack for easy powering of the device.

Consequently, looking at Figure 2.10, using this device you only need to equip it with a sensor and a transceiver to have a fully working wireless sensor node.

2.5 Android

Android is today one of the world's most popular mobile operating systems. It started off when Andy Rubin, Rich Miner, Nick Sears and Chris White founded a company named *Android Inc* in 2003. Their vision was to develop mobile devices which had the power to adapt based on user demand. In 2005, Google Inc. bought the company and propelled Android into what it is known for today.

As smartphones are starting to become ubiquitous, there is a good chance that when developing an Android application, the intended end user is already familiar with the system and its general application design.

2.5.1 System Architecture

The software stack of Android is built upon the Linux kernel and uses its own version of the Java Virtual Machine, ART, as its runtime. The stack also includes other essential libraries and key applications needed to work properly. Each application (written in Java) runs in its own process with its own instance of the virtual machine, this then relies on the Linux kernel for underlying functionality [45].

Figure 2.11 illustrates the software stack of Android and highlights some of the included components.

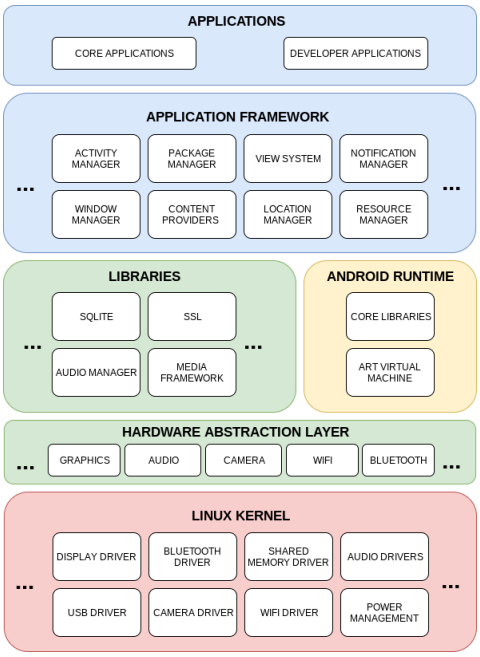

Figure 2.11: Androids software stack and the most common components

2.5.2 Application Structure

An Android application shares the smartphone's resources with all other running services and applications. Naturally, this means that not every application can be running simultaneously. Moreover, this also implies that whenever an application is not currently active, some of its resources could be released for the other application to use. To make resource management as effective as possible, Android has developed an application structure [26] each application must follow. This structure enforces the application to be within one of the states displayed in Figure A.2.

Briefly, once an application is launched, its main activity's `onCreate()` method will be invoked. This method initializes its threads, some class scope variables

and declares the structure of the view. For each new activity displayed, this function should be called once. The activity that goes into the background call its `onPause()`, followed by the `onStop()`, to release its resources. Furthermore, the state of the application is dependent on the user's input, yet, the flow always follows the flowchart in Figure A.2.

2.5.3 Android and Real Time Constraints

As seen in Figure 2.11, there is no support for real time constraints in the default system architecture. An application running on the Android system architecture is isolated in its own virtual machine and handled by the Linux kernel, as described previously. Linux itself is not a RTOS which makes the Android application unable to guarantee, for instance, specific timing requests.

Still, there are several plugins and frameworks trying to make Android real time compliant, for example RTDroid [46] which alters the software stack adding components required to make the system more of an RTOS. However, these often require some major alterations to the system architecture and core components which means that it could be considered a type of "specialized" system when implemented which is a contradiction to our scope.

2.6 Internet

The evolution of Internet is a chain-reaction which can be traced back to 1961, when Kleinrock published his research on packet switching theory [28]. His research showed that packet switching networks could prove to be superior to the then traditional circuit switched networks. This was a huge leap towards computer networks as we know them today. Kleinrock inspired another fellow researcher Lawrence G. Roberts to find out if computers could actually communicate, together with Thomas Merrill, he experimented on connecting a computer, via telephone lines, in Massachusetts to another computer in California [33]. This experiment made them realize the power of letting computers communicate, however, the circuit switched networks was not good enough to connect computers directly to each other.

Working for ARPA (later known as DARPA), Roberts published "Multiple computer networks and intercomputer communication" in 1967 [38], where he laid the foundations of the Internet. The term *ARPANet* also has its origins in this research. Due to his early interest in packet switching networks, Kleinrocks Network Measurement Center at UCLA was the first to be connected to ARPANET. Not long after Kleinrocks first connection, many computers started to get connected to ARPANET. However, it was not until 1985 that extensive work on commercialization had begun [30].

2.6.1 Internet of Things

As the name suggest, Internet of Things is dependent on the Internet. The concept was first coined back in 1999 by Kevin Ashton, when he was presenting his ideas of enabling computers to gather information about the physical world themselves,

replacing the human, with sensors, and using the Internet as a platform for communications [4]. This idea started to evolve into, “What if we connect everything to the internet?”, then we would be able to monitor and control any given thing we wanted.

Since the beginning, Internet of Things lacked a specific definition due to it mostly being a generic term rather than a specific type of system. This have made the evolution of Internet of Things quite hard to follow, industries use it freely with their own interpretation obscuring the development of IoT in popular science. Today, most agree upon that it is a way of developing a connected world. A world where more and more things are used in society to monitor and control different types of systems, with or without human interaction.

Chapter 3
Architectural Overview

This chapter is dedicated to explain some of the components used to build the prototypes. Also, why specifically they were chosen in comparison to other possible components that were considered, before the final decision was made.

To highlight that the communication in this system is unidirectional, only going from the smartphone to the sensor node, the transmitter is referred to as "(Key-)Transmitter" while the receiver is called "Sensor Node". Equally, these names are supposed to make it clear that the transmitter and receiver are two totally different types of devices. Whereas the sensor node is the device intended to actually be deployed as described in Section 2.3, and the key-transmitter is only used to configure the sensor node.

3.1 (Key-)Transmitter

To begin with, the first consideration was to build the key-transmitter as an NFC tag system. This was thought to be a good candidate as NFC systems provide great functionality to transfer small amounts of data while at the same time being incredibly cheap. Basically, the NFC tag would contain the pre-shared secret while the sensor node would be equipped with an NFC reader. Moreover, the pre-shared secret would be loaded into the tag utilizing a smartphone with NFC capabilities. However, as we tried to add as little extra hardware to the configuration procedure as possible, we thought that there had to be another way of eliminating the tag. Thus, making the key-transmitter and sensor node communicate in a more direct manner, consequently, using less extra hardware. Not only that, but as this NFC tag easily is lost or stolen, the network that used that particular NFC tag's secret key could be compromised. An adversary could then, before the lost tag was noticed, break and retrieve the secret key within the tag and thereby decode every encrypted packet. Alas, we concluded that an NFC tag based solution would not be the most optimal one.

Next, we considered developing our own customized key-transmitting device that could transmit data through the specific OOB channels. Ignoring the scope, this would be the ultimate candidate in the perspective of the system itself. However, it would require large amounts of time and money to develop. Obviously, such a solution could not be considered: "as little prerequisites as possible" and because of this, it was quickly discarded.

Lastly, in previous work such as [34][7][21], the solution always involved a "cameraphone" to achieve the transmissions. The fact that those research papers are all at least ten years old mean that any cameraphone they used is probably no match compared to today's smartphones. A modern smartphone outperforms older cameraphones not only in: computational power; memory; versatility of software; but much more. A decent smartphone by today's standards will by default contain a microphone, a loudspeaker and a camera. However, it is not uncommon that they contain extra components such as an accelerometer, a flashlight, an NFC reader and writer and so forth.

These qualities make a great foundation, giving us many possibilities of utilizing different OOB channels to configure a pre-shared secret into a wireless sensor node. Basing the key-transmitter on a smartphone would not only give the system great flexibility but once developed and tested it could easily be distributed via the official "Google Play Store".

Therefore, we decided to build our key-transmitter based on a smartphone. The smartphone should be of such sort that it by default came equipped with at least: a camera, a flashlight, a screen, a loudspeaker and a microphone. As smartphones are becoming ubiquitous we assume that they can be considered as little prerequisite hardware as possible, since there is a high probability that the end-user already owns one, or the intended company does. The smartphone should be running Android OS as 86% of the 2017 worldwide smartphone sales [2] are based on it.

3.2 Receiver (Sensor Node)

Since the Glass Break Sensor, (GBS) that Axis Communications wishes to incorporate into their current system is not defined yet, we had to resort to a simulation of it. After vigorous researching, a specific manufacturer stuck out from the rest of the crowd. As described in Section 2.4, Arduino has taken the versatile MCU called Atmega328P, made by Atmel and integrated it into a development platform that is easy to experiment with. An illustration of the MCU is seen in Figure 3.1. As a result, many research projects use an Arduino Uno R3 development platform to simulate their wireless sensor nodes. Because of this, we also felt that simulating the GBS with an Arduino Uno R3 would be a good enough.

Two other important factors that made us decide to proceed with this were: firstly, it supports easy integration of different wireless communication interfaces. Which clearly is an important factor when building a *wireless* sensor node, as this interface is one of the fundamental building blocks illustrated in Figure 2.10. However, as the scope of this thesis does not consider transmission over the wireless channel, we decided that we would not integrate a wireless interface in our prototype. Nevertheless, the option of extending the simulation further if ever needed is there. Secondly, the MCU offers several levels of *sleep modes*. These sleep modes, which are listed in Table 3.1, are described in more detail in [14]. Each and every mode has its own distinct characteristics concerning power consumption and wake-up triggers. Depending on how much power you want to save and what functionality you still want operational, there is a certain sleep profile that will suit

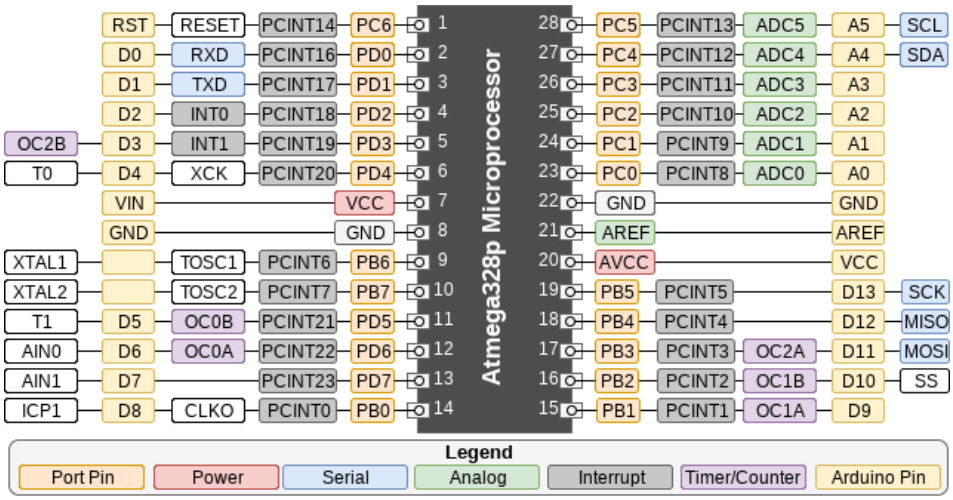

Figure 3.1: Mapping between the Atmega328p microcontrollers I/O pins and the Arduino Uno R3 I/O pins.

you. Turning to our case, we want the wireless sensor node to save as much power when asleep as possible while only waking up when the key-transmitter initiates a transmission. This translates into, the sensor node being asleep until an external interrupt occurs in the form of an initiated transmission.

Again, looking at Table 3.1 which pinpoints the operating functions under the corresponding sleep modes. It is seen that the modes *Standby* and *Power-down* have the fewest "X" marks. The rule of thumb is that the fewer the "X" marks there are on each line, the more power efficient that mode is [14]. Since the table shows that both *Standby* and *Power-down* have the same amount of marks it is easily thought that they have about the same power consumption when put in either mode. In fact, looking further into the datasheet [14] it is mentioned that when the MCU is put in *Standby* mode, the main clock source is enabled unlike in *Power-down* mode. Consequently, this means that the sleep mode *Power-down* actually is more energy efficient which is why we decided to use it.

In the future, when the prototypes need a ratified design, it is easy to decouple the Arduino development board and the MCU. As the development board is quite expensive, designed for experimenting rather than actual deployment, the decoupling reduces both cost and size significantly. As of today, the retailer `digikey.com`, sells the Arduino Uno R3 at a price of $23.38 while a standalone Atmega328 has a price of $1.25. This is great as it positively affects the choice of simulation.

3.3 Sensing Equipment

The choice of which light sensor that should be used to utilize the visual based OOB channel and which microphone that should be used to detect the audio

Table 3.1: Atmel Atmega328P's wake-up sources for each sleep mode [14].

	Wake-up Sources						Software BOD Disable
Sleep Mode	INT and PCINT	TWI Address Match	Timer2	SPM/EEPROM Ready	ADC	WDT	
Idle	X	X	X	X	X	X	
ADC Noice Reduction	X	X	X	X	X	X	
Power-down	X	X				X	X
Power-save	X	X	X			X	X
Standby	X	X				X	X
Extended Standby	X	X	X			X	X

transmitted signals had to meet one simple requirement. Both of which had to be very cheap to purchase, because the influence on the overall price of the sensor node should be affected as little as possible.

As for the visual-based solution, we have chosen to use the photo-transistor *SFH213* manufactured by Osram, [40]. This photo-transistor has: a fast switching time of 5ns; filters out most of the light unless its coming into the photo-transistor at an angle of maximum 10 degrees; and it detects light with wavelengths between 400-1100nm. At the time of writing this report, the retailer `digikey.com` sells this photo-transistor at a price of $0.30268. Hence, we believe that this photo-transistor does not impact the overall manufacturing costs too much while providing great characteristics for the intended purpose.

Turning to the audio sensor used in our prototype, which is called *SKU:DFR0034* and manufactured by DFRobot [16]. Basically, it is a microphone connected in a breakout board to establish a connection to the Arduino platform with ease. It also provides some components for easier testing, such as a potentiometer controlling the sensitivity of the microphone and operational amplifiers to have greater detection ranges. Even if this "audio sensor" is relatively expensive, $6.90 each at `digikey.com`, the actual microphone should be bought without the breakout board when implementing the real product. Then, a microphone and the relevant components can be bought for no more than $0.34155 at the same retailer.

3.4 Line Encoding/Modulation Techniques

Achieving a reliable synchronization between devices that wish to communicate is generally a hard problem to solve. Many synchronization schemes used today are based on using either the Global Positioning System (GPS), Network Time Protocol (NTP) or a combination of both [37]. Yet, neither GPS nor NTP are viable in a wireless sensor node environment due to the sensor nodes resource constraints. Consequently, we decided to base our solutions on a self-clocking signal type. This solves the problem of providing a separate clock signal at the cost of bandwidth. However, since the application only is concerned with transmitting small amounts of data it is considered a positive trade-off.

To be able to compare and test, we chose to implement our visual-based solution on two different techniques. These techniques were Manchester Encoding and Pulse Width Modulation. To begin with, Manchester Encoding was chosen

because it is a robust technique, used in IEEE 802.3 when operating at 10Mbps [8]. This means that it is well tested as it has been already used in commercial applications. Although given the fact that the key-transmitter was based on an Android system, specific timed operations could not be taken for granted. It was foreseen that transitions from zero to one (`1` being sent) or one to zero (`0` being sent) would have varying times making it hard for the receiver to decode. Thus, Pulse Width Modulation was chosen as a complement to ME, because it does not switch from zero to one as much. This would potentially make the transmitted signal somewhat more reliable. Even if not being exact, the receiver could be made to decode the signal easier.

Regarding the audio-based solution we chose to implement the communication with FSK as signal modulation. This is due to the fact that FSK has a higher resilience towards noise compared to ASK and is widely used in systems with lower capabilities. The drawback of using FSK is that the decoder becomes more complex to implement. Furthermore, FSK requires large amounts of bandwidth, which is a concern if the system is intended to utilize the regulated frequency-band. However, since our solution is using sound waves this should not be a problem. Moreover, the reason to change the audio-based encoding technique to FSK was due to the difference in electromagnetic waves compared to sound waves. In the beginning, it was believed that these would behave similarly, which early testing disproved. For instance, if PWM was to be used in the audio-based channel, the time of a specific sound wave had to be measured. This was not as easy as we thought because sound waves propagate a lot slower than electromagnetic waves. This reduction in speed makes the sensor able to detect parts in the wave and not the complete wave. This phenomenon is illustrated in Figure 3.2.

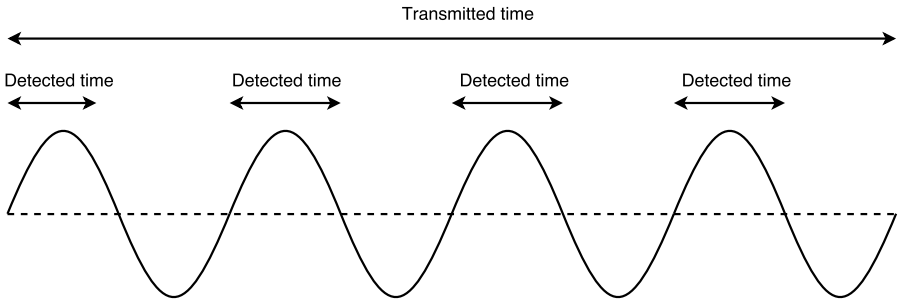

Figure 3.2: The problem of detecting a single sound wave transmitted a specific time as separate parts.

3.5 Data Encapsulation

Encapsulating the data to be sent grants many benefits. It makes it easier to detect, prevent and handle errors within the transmission. On top of that, the possibility of numbering each encapsulation becomes an option. Then, a sequence of numbered encapsulations can be received and interpreted as a combined larger

message, transmitted as minor packets. This is effective since it provides better scalability. The communication between the key-transmitter and the wireless sensor node is unidirectional and is always initiated by the key-transmitter. Therefore, the design of the data encapsulation is done in a way to be as fundamental as possible. It only contains the necessary elements that are needed, in order to transmit a small amount of data and detect if it fails. Because the messages are rather small, the overhead of automatic retransmission is too demanding. Hence, if a failed transmission is detected, the user is simply asked to retransmit manually.

Table 3.2: Overview of data encapsulation, each number is given in byte.

SYNC	PAYLOAD LENGTH	PAYLOAD	CRC
1	1	0 - 255	4

The design of the data encapsulation can be seen in Table 3.2, one frame is at most 261 bytes long, containing 4 fields: *sync*, *payload length*, *payload* and *crc*.

SYNC is primarily responsible of telling the receiver that there is data incoming. This is needed because as the receiver wakes up from its sleep mode on edge detection, it may take a few clock cycles to get all the functions started again. This way, sync provides a margin before the actual signal interpretation is needed.

The structure of the *sync* block will differ slightly if the transmitted messages are encoded using ME, PWM or FSK. Still, it will always be one byte long. Firstly, the *sync* of ME transmissions will be `0111 1111`. Secondly, the PWM version is `1111 1111` and lastly, the FSK uses `0010 1110` mapped to the frequency 1880Hz. The subtle change between the sync of ME and PWM is because with ME `0` will be encoded as `10`, waking the receiver at the right position in the signal. Put simply, to wake the sensor node there needs to be light, therefore, the transmission has to start with a `1`. Both PWM and FSK are relieved of this problem and could have the sync bits set to anything.

PAYLOAD LENGTH contains the length of the payload. This section is also only one byte, thus the largest amount of bytes the payload can contain is when this field is set to `1111 1111`, which is the same as $2^8 - 1 = 255$ bytes. With regard to the fact that the system is only supposed to send small amounts of data, 255 bytes of data should be plenty enough. As previously discussed, NIST states that using symmetric keys of length 128-192 bits (utilizing AES-128 or AES-192) will stay secure until the year of 2031 and beyond [6].

PAYLOAD is the actual data to be transmitted, in our case, the pre-shared secret. Its size will depend on the actual size of the key, but will most likely be either be 128 bits or 192 bits.

CRC is the last 4 bytes of the frame. It is present to be able to perform a light error detection of the transmitted frame. It must be energy efficient and

easy to compute since it will be calculated twice for every transmission, once by the key-transmitter and once by the sensor node. The key-transmitter calculates the CRC value based on the *payload* section and appends it to the frame. Once the frame is received, the sensor node will also calculate the CRC and compares its CRC value to the received CRC. Depending on if they match, it can decide whether or not it should accept the payload, or discard it.

To summarize a user scenario from a communication perspective: First, the key-transmitter appends the *sync* pattern depending on the chosen encoding style; Second, after the pre-shared secret is generated, calculate the length of this data and append that value to the *payload length* section; Thirdly, append the data itself to the *payload* section. Fourthly, calculate the CRC-value of the data using Javas built-in *CRC32*-library and append this to the *crc* section. Finally, the key-transmitter transmits the entire frame to the sensor node.

Chapter 4
Implementation

This chapter is dedicated to providing more details on how we implemented the key-transmitter as well as the sensor node. As the key-transmitter is built on top of the Android OS, it is purely software based. Concerning the sensor-node, schematics and explanation of the firmware will be stated.

4.1 Implementing the (Key-)Transmitter

To begin with, one of the great benefits of building the key-transmitter on top of Android, is that it is nothing more than another software application for the smartphone. The process of downloading and installing the software is something the end-user should already be familiar with and know pretty well. Conversely, if the user is not familiar with the operating system, Android is very intuitive and easy to learn.

Now, when the user starts the application, the operating system initiates the method `onCreate(){...}` of the class `LoginActivity`. This activity is responsible for *authenticating* the user trying to access the application. The authentication process is based on the user providing credentials which are then matched with an existing profile. If it is the first time running the application and the user has no profile yet, they are prompted to create a profile. Subsequently, if the entered credentials match the stored ones, clearly, the user is authorized. When authorized `onCreate(){...}` of `SelectorActivity` is called. The flow through the application is described in the UML diagram which is visualized in Figure A.3.

Next, once `SelectorActivity` is created the user is prompted to enter a message, choosing a OOB channel and selecting which encoding technique to use. The message is an ASCII-encoded string representing the *pre-shared secret* that is supposed to be loaded into the sensor node. The choice of representing the pre-shared secret this way is that we felt it to be a good way of generating different types of "random" bit patterns. As a real pre-shared secret could be any sequence of bits. The choices of OOB channels are obviously visual- or audio-based and the encoding techniques are ME or PWM for visual and FSK for audio. As described, our prototypes provide a dynamic way to choose which encoding technique that will be used. However, this decision is based on the fact that this renders easier testing and not ease of use. A finalized product should only contain, for instance, the most optimum encoding technique and not let the user choose themselves.

Lastly, when the user has entered the required fields in `SelectorActivity`, it starts a new activity correspondent to the choices made. The activity `LightTrans Activity` is started if the user intended to use the visual-based channel. Similarly, `AudioTransActivity` is started if the audio-based channel was selected. `LightTransActivity` contains a variable called `fullPacket`, which is a linked list containing all the bits to be transmitted. The list `fullPacket` is constructed following the data encapsulation scheme described in Section 3.5. The list is constructed using the `Utils` class. This class contains methods of generating the bits needed as well as methods utilizing Javas built-in `CRC32` library.

Likewise, if the audio-based OOB channel was selected `SoundTransActivity` is started. This activity contains a variable called `messageToSend`, which is a string containing all the characters to be transmitted. The reason for this is because the audio-based solution uses a larger symbol size than the visual-based channel. Each character in `messageToSend` is mapped to a specific frequency using a `FrequencyMapping` class. Once this is complete, the rest of the required fields are appended to the transmission queue. When a full transmission is generated, a "guard"-frequency is placed between every character to be sent. This is done because then the sensor node can differentiate between noise and signals. If the receiver detects a signal which is not followed by this guard frequency, it can assume that what it detected was noise rather than a signal.

Finally, in both cases described above, once the transmit button is pressed in either of the activities, a separate `Transmitter`-thread is started at the highest level of priority. This thread is responsible for the actual transmission of data. Blinking the flashlight in the visual-based case or playing different frequencies on the loudspeaker in the audio-based case.

4.1.1 Utilizing the Flashlight

Android provides high level access to the smartphones different hardware components, such as the flashlight, via the "hardware"-API. This API was used to gain access to the camera, which is required because the flashlight is considered a subset of the camera. Easy access to the camera was achieved by creating a `CameraManager` object, which is a system service that connects and manages the camera of the smartphone. With the help of this camera manager interface, the flashlight can be turned on or off with a call to the method `CameraManager.setTorchMode(String cameraId, boolean enabled)`, where `cameraId` is the id of the camera and `enabled` is true if the flashlight is to be turned on and vice versa.

4.1.2 Utilizing the Screen

The screen-mode has a large portion of the screen reserved as "the transmission area" where nothing but a background exists, illustrated in Figure 4.1. The background is a `TextView` object which can be controlled with the UI-thread of Android. The screen's brighness set to the maximum setting in order to emit as much light as possible. The UI-thread is used to change the TextView's background color to switch between black or white, depending on the bit to be transmitted.

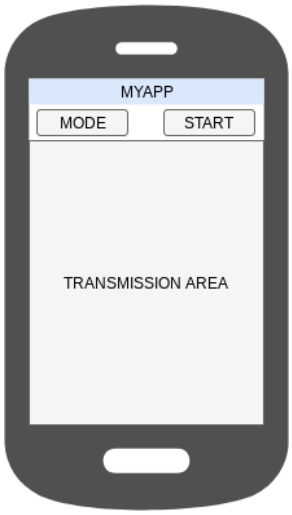

Figure 4.1: Application design when screen mode is active.

4.1.3 Utilizing the Loudspeaker

If the option to use audio as OOB channel was selected, a `ToneGenerator`-class is created. This class contains the means of creating sine-waves of different frequencies and then playing them on the loudspeaker of the smartphone. To be able to play a sound on the loudspeaker the class also creates an `AudioTrack` from the library `android.media.AudioTrack`. As the message to be sent is given to the class when constructing it, each character within the *pre-shared secret* is mapped to a specific frequency and loaded into an array of frequencies. Then, for each of the frequencies in that list the `AudioTrack` is filled with values. Those values are sine-waves created by sampling the `sin`-function of the `java.lang.Math`-library with a specific duration and sampling frequency 48kHz.

The mapping between frequencies and characters used by the application is listed in Table 4.1. The frequencies are roughly chosen with regards to the probability of the letter being used in the English language. As our prototypes take in human entered text, having low frequencies on common letters creates a less "annoying" signal. This is because humans' hearing range decrease with age, making the outer ranges harder to hear. Moreover, the character "~" corresponds to the guard frequency and "." corresponds to the sync pattern, as described in Sections 3.5 and 4.1.

4.2 Implementing the Receiver (Sensor Node)

We used PWM and ME encoded signals when transmitting through the visual-based OOB channel and FSK encoded signals in the case of audio-based communication. As described in Section 4.1.3 this is due to the fact that it is easier to make electromagnetic waves to act as "ordinary" line-encoding. Put simply,

Table 4.1: Mapping of character to be transmitted to its specific frequency.

Character:	**a**	**b**	**c**	**d**	**e**	**f**	**g**	**h**
Frequency[Hz]:	440	3200	3320	920	320	3440	3560	800
	i	**j**	**k**	**l**	**m**	**n**	**o**	**p**
	560	3680	1160	1040	3800	1520	3920	4040
	q	**r**	**s**	**t**	**u**	**v**	**x**	**y**
	4160	4280	680	1640	4400	4520	1280	1400
	z	**␣**	**~**	**.**	**0**	**1**	**2**	**4**
	4760	1760	200	1880	2000	2120	2240	2360
	5	**6**	**7**	**8**	**9**			
	2600	2720	2840	2960	3080			

mapping detection of light to *positive level* and no detection of light to *negative level* as in a more traditional system. We turned to FSK for the audio-based case because the sensor node had problems of mapping detection of sound to *positive level*. This issue was caused due to the fact that received signals' amplitude level were not consistent making it hard to differentiate between a signal and noise.

Once the received signal is decoded all prototypes use roughly the same firmware to interpret the data. Basically, it works by first, detecting the specific synchronization pattern; second, reading the length of the payload; third, reading the correct amount of bytes as data; fourth, calculating a crc with the help of the library `CRC32` [5] and comparing it to the received crc; last, giving the user feedback by blinking the diodes based on if the two crc values match or not.

Moving forward, this section follows the following outline: the proccess of decoding the different type of signals is explaned; followed by details on the visual-based sensor node; and lastly, details on the audio-based sensor node.

4.2.1 Decoding PWM Signals

To decode PWM encoded signals a fairly straightforward firmware was implemented, a flowchart of the receiver software can be seen in Figure A.1. The port of which the photo-transistor was connected to, was set to trigger a Pin Change Interrupt (PCINT). This means that as soon as the photo-transistor detects light, the MCU was powered up again running the corresponding interrupt service routine (ISR). This interrupt uses both rising edge- and falling edge-detection, which means it triggers its ISR on any change sensed. Rising edge detection means a trigger will be sent when going from not sensing to sensing (0 to 1) and conversely falling edge detection means going from sensing to not sensing anymore (1 to 0), as illustrated in Figure 4.2.

Therefore, the ISR was designed to measure the time between two subsequent calls. This time would then represent the time that the photo-transistor detected light. As it would get called twice when the key-transmitter sent a pulse of light.

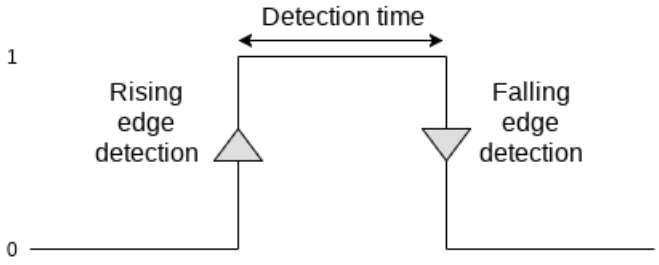

Figure 4.2: Rising and falling edge detection on a signal pulse.

Both on detecting light (0 to 1) followed by losing light again (1 to 0). This time could then be used to interpret what bit the key-transmitter sent by giving the receiver intervals of times corresponding bit `0` and bit `1`.

4.2.2 Decoding ME Signals

The method of decoding a signal of type ME is fairly easy, if the devices involved have real-time capabilities. This is due to the fact that there are many open source libraries which deals with Manchester decoding, however, these libraries require real-time capabilities. As this is not the case in our environment we had to implement our own, *non* real-time firmware. As we cannot change the time in which the key-transmitter actually transmits, compared to what it is set to transmit. It is, therefore, up to the sensor node to do the proper adjustments of detecting drifts in time.

In the first place, the firmware decoding ME signals is set to trigger an ISR on rising edges of a PCINT on the port that the photo-transistor is connected to. The ISR responds to a rising edge trigger and is responsible for adjusting the sensor node's clock depending on the drift in time, introduced by the unstable key-transmitter. The first time the ISR is triggered, it is already known that the bit being received is a `0`, according to the protocol. Therefore, the first rising edge detection is in the beginning of a transition. Now the sensor node resets the clock, by setting TCNT1 to zero, to have a point of reference in time at the beginning of the received signal.

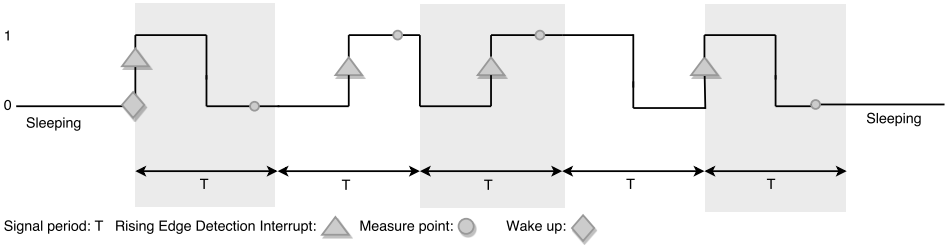

Figure 4.3: Key points of measurement in the firmware used by the sensor node to decode a ME signal.

Next, the rising edge detection is used to continuously synchronize the sensor node's clock to the signal. It does this by comparing the current time of each trigger to two predefined intervals. These two intervals are:

$$\frac{6 * T}{15} \leq 0.5 * T \leq \frac{11 * T}{15} \tag{4.1}$$

$$\frac{14 * T}{15} \leq T \leq \frac{3 * T}{15} \tag{4.2}$$

If the current time of the trigger is within the interval in Equation 4.1 the receiver resets its clock to exactly $0.5 * T$. Similarly, if the trigger is within the interval in Equation 4.2 the clock is reset to exactly T. Thus, the receiver adapts to the drifting time of the key-transmitter.

On top of that, two Output Compare Interrupts (OCR1A & OCR1B) are enabled. OCR1A is set to trigger its ISR at the time of $0.8 * T$ while OCR1B is set to trigger its ISR at the time T. On the one hand, the ISR corresponding to OCR1A does the actual signal measurement by reading the current value of the photo-transistor at the time of the trigger. The value of the measurement made by OCR1A is then used when making the symbol decision, seen in Figure 4.3. If that measurement is `1` then the symbol decision is also one. Similarly, if it is `0` the symbol decision is clearly zero as well. On the other hand, the ISR corresponding to OCR1B is used to reset the node at time T for each iteration. It does so by setting TCNT1 back to zero when triggered. Hence, the entire procedure is repeated.

4.2.3 Decoding FSK Signals

As for the firmware implemented to decode the FSK encoded signals, the library `Arduino_FFT` [13] was used. This library contains methods of calculating the DFT based on the FFT algorithm. We used a sampling rate of $F_s = 9600$ taking $N = 128$ samples. The sampling rate was based on the Nyqvist-Shannon sampling theorem, which states that the sampling frequency should be at least twice the size of the highest frequency in the signal that is being sampled. As the key-transmitter has the possibility of sending a symbol with a frequency as high as 4760Hz. The sample size is based on the largest one possible before the memory of the MCU runs out. From the frequency-domain data the FFT calculates were the most dominant frequency is located, corresponding to the peak of the data. Once the position and value of the peak is located quadratic interpolation is used to increase the accuracy of the frequency approximation. This approximation is then used as a base for a symbol decision.

Figure 4.4 depicts a visual representation of the resulting frequency-domain data calculated using FFT. The received signal, which was sampled, was a sinusoidal with the frequency 440Hz. This shows that the most dominant frequency (peak) is detected at about 459.4Hz. Consequently, it is important to have a margin for error when deciding which symbol corresponds to the received frequency. This allows the receiver to have a higher probability of successful symbol judgment even on less accurate approximations. We chose an interval size of 140 Hz to represent a single symbol, separating each frequency by 120Hz. The interval

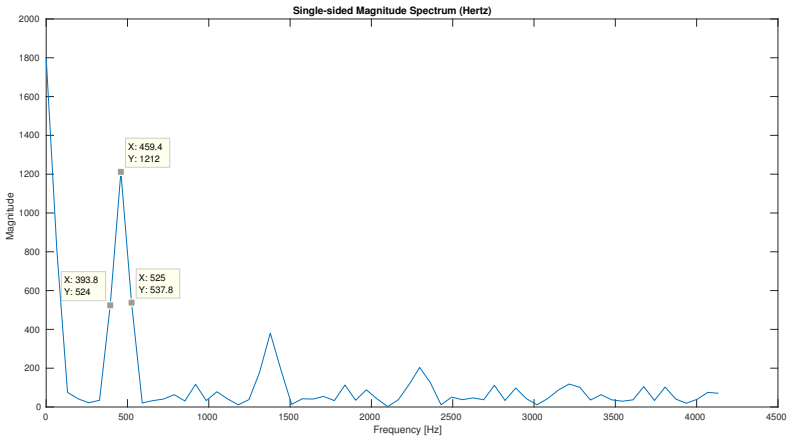

Figure 4.4: Resulting frequency bins of the FFT algorithm on the MCU (receiver). Sampling at $F_s = 9600[Hz]$ and taking $N = 128$ samples.

size is chosen to this size because it gives the receiver some space to breathe while in the meantime does not result in receiving frequencies higher than our limit of 5000Hz. Which is a limit set by the `Arduino_FFT` library.

4.2.4 Details on Visual Based Sensor Node

To have the functionality of receiving visual based signals, the sensor node was equipped with a photo-transistor. The photo-transistor is connected in series with a resistor. This way we are able to control the photo-transistor's sensitivity by changing the value of the resistor. The sensor node was also equipped with two extra diodes, these are used to be able to send some feedback to the user; one green diode for success and one red diode for failure. The schematics of the receiver is displayed in Figure 4.5. The only schematic difference when comparing detection of light emitted by flashlight or the screen, is the size of resistor *R3*. A larger value of R3 results in the photo-transistor being more sensitive to light, making it easier to detect the less intense light emitted by the screen.

To make the prototype as easy as possible to implement, R3 was chosen to be 390kΩ. This decision was backed up by the fact that this choise of resistor value would act as a hardware filter that would otherwise have to be implemented by software. Hence, we save some overhead from the sensor node's resources. In addition, the point of measure was connected to digital I/O port, D9. The reason behind this is derived from the fact that Atmega328p utilizes a 10-bit *ADC*, with a reference of 5 volts by default. Which in turn, yields a resolution of:

$$\frac{5V}{2^{10} - 1} = 0.0048875 \approx 4.9mV \tag{4.3}$$

with this resolution, any voltages below $4.9mV * 512 \approx 2.5V$ will be interpreted

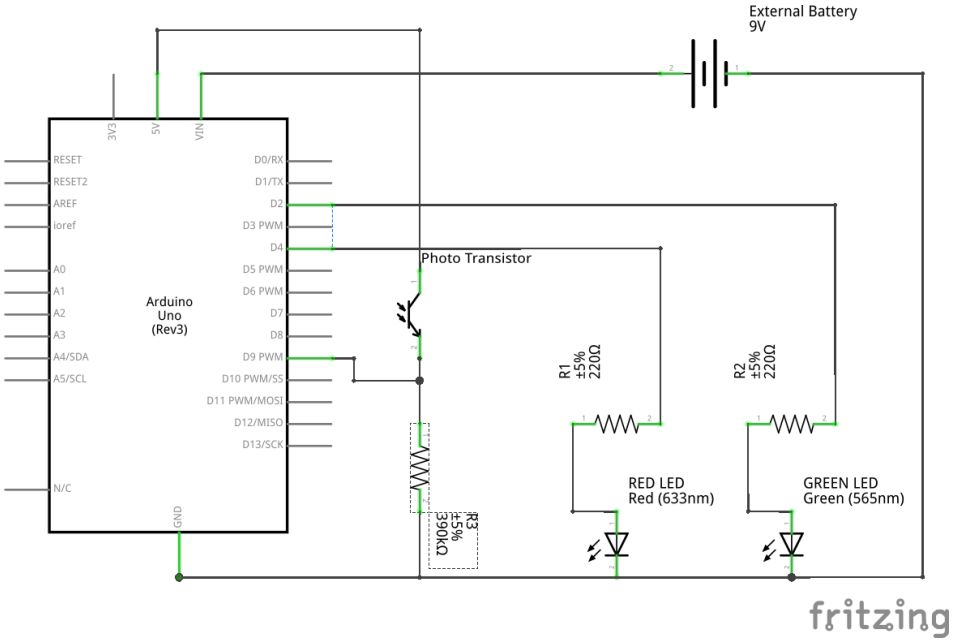

Figure 4.5: Schematic of the receiver with the capability of detecting light emitted by phones flashlight.

as a `0` on a digital I/O port while voltages above 2.5V will be interpreted as a `1`. As a consequence, a sort of hardware based filter against noise is created. Since noise would typically not create voltage levels high enough to push the digital I/O port into sending a `1`.

Using the result of Equation 4.3 the following function is created:

$$F(x) = \frac{4.9}{1000} * x[V] \tag{4.4}$$

This function takes the analog value measured by the photo-transistor and outputs the corresponding voltage level. The photo-transistor we used, detected approximately $F(55) = 55 * 4.9mV \approx 0.27V$ in a normal, well lit office space (350 lux). Clearly, 0.27V is well below 2.5V and was therefore interpreted as `0` at the connected digital I/O port.

Table 4.2 lists how the sensor node reacted to an increasing value of R3. Making the photo-transistor increasingly more sensitive, in an environment with 350 lux of illumination. The table also highlights the fact that a value on the resistor as high as 4MΩ is not viable, at least not if the sensor node is supposed to be deployed anywhere else than a very dark environment. Due to the noise levels being high enough to switch the digital I/O port to `1`, making it impossible to decode.

As for detecting signals transmitted via the screen, some adjustments had to be done. Firstly, we must determine a new value of R3 in order for that solution to work. There is a huge difference in the intensity of the light emitted

Table 4.2: Digital, analog and voltage values of the photo-transistor with different values on R3, both when the key-transmitter is emitting light through the flashlight and environmental impact.

	Environmental impact (350 lux)			**Flashlight-based signal**		
R3[Ω]	**D9**	**A1**	**F(A1)**	**D9**	**A1**	**F(A1)**
490	0	0	0	0	15	0.0735
10K	0	0	0	0	183	0.8967
30K	0	5	0.0245	1	550	2.695
390K	0	55	0.2695	1	1023	5.0127
4M	1	650	3.185	1	1023	5.0127

by the flashlight compared to the screen. Obviously, the screen does not emit one "beam of light" as the flashlight does. Even with the screen's brightness set to the maximum setting the level of intensity is not even close. Therefore, we must increase the sensitivity of the photo-transistor by increasing the value of resistor R3 even more.

As seen in Table 4.3, only the settings of last two rows are able to make the photo-transistor so sensitive that light emitted by the screen will result in a value of `1` on the digital I/O port, D9. Looking more closely, even the last row in the table seem impossible to use. As now the photo-transistor is so sensitive that even without transmitting, the digital I/O port is outputting `1`. However, the fact that the impact of the environment is slightly decreased when a transmission is in progress, the screen of the smartphone blocks the photo-transistor. Therefore, the value of R3 was chosen to 10MΩ in the sensor node receiving signals transmitted via the screen.

Table 4.3: Digital, analog and voltage levels of the photo-transistor with different values on R3, both when the key-transmitter is emitting light through the screen and environmental impact.

	Environmental impact (350 lux)			**Screen-based signal**		
R3[Ω]	**D9**	**A1**	**F(A1)**	**D9**	**A1**	**F(A1)**
212	0	0	0	0	4	0.0196
10K	0	0	0	0	15	0.0735
220K	0	15	0.0735	0	30	0.147
390K	0	25	0.1225	0	56	0.2744
820K	0	70	0.343	0	120	0.588
4M	0	400	1.96	1	650	3.185
10M	1	700	3.43	1	1023	5.0127

4.2.5 Details on Audio Based Sensor Node

As mentioned in Section 3.3 we used an experimental breakout board as the "audio sensor" which was easy to connect to the Arduino. This audio sensor is powered by 5V provided by the Arduino board and has one line connected to one of the analog I/O ports. The firmware continuously reads from the connected port saving the value which the audio sensor outputs. Once the number of samples needed has been reached the FFT algorithm is applied and a decision is taken.

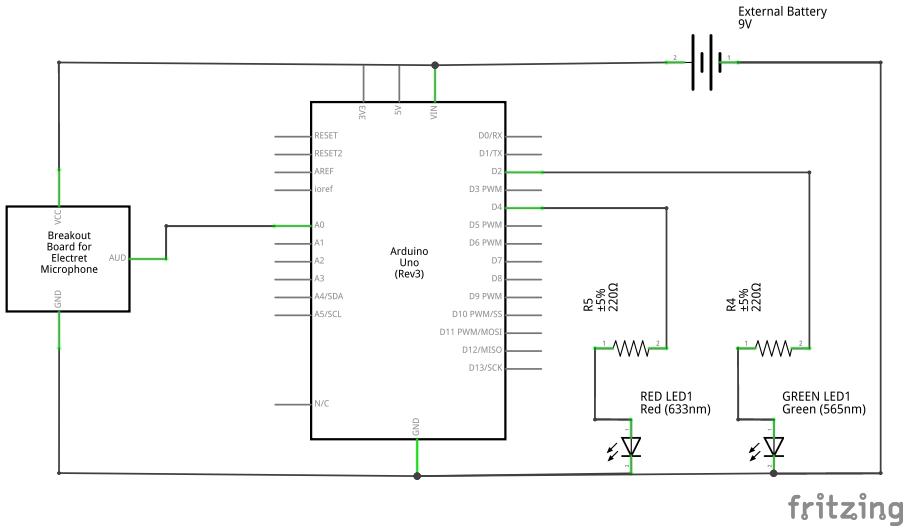

Figure 4.6: Schematic of receiver capable of detecting and sampling sound.

Figure 4.6 shows how the breakout board was connected to the Arduino while Figure 4.7 is a clipboard from the sensors datasheet [16] showing the details of how the audio sensor itself is built.

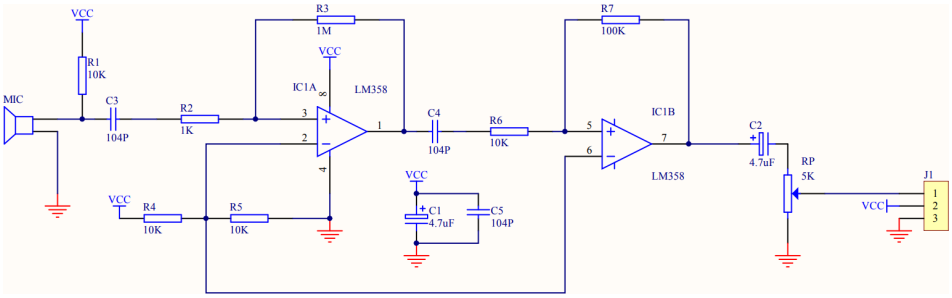

Figure 4.7: Detailed schematic of the audio sensor [15].

Chapter 5
Results

This chapter is dedicated to present the results derived from testing the developed prototypes. The results are separated into two categories. First, results of the two visual based prototypes are presented, then, the results of the audio based one. This separation is done to make the differences between the two clear.

Interpretation of the results will be further discussed in Chapter 6.

5.1 Visual Based Prototypes

To start, Figure 5.1 shows the measured duration the sensor node is detecting light, when the key-transmitter is set to emit a flash of light for both 5ms (the two boxes to the left) and 20ms (the two boxes to the right). In addition, the test is performed both with and without airplane-mode (flightmode) active on the key-transmitter. Each box in the figure has a sample size of 1200.

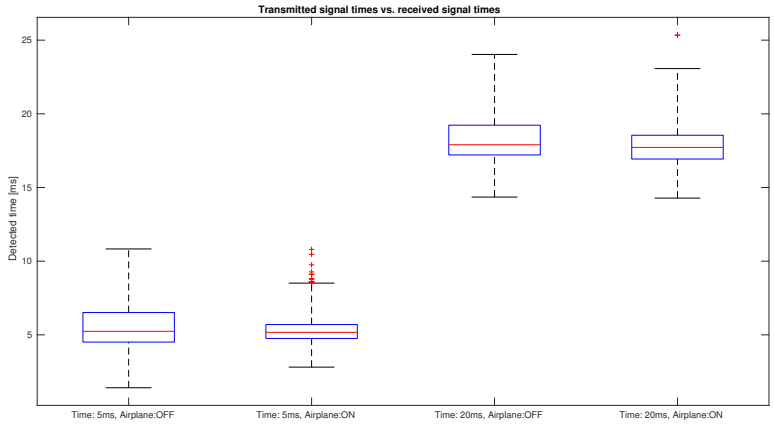

Figure 5.1: Detected times vs. transmitted times utilizing the **flashlight**.

Furthermore, Figure 5.2 displays the results of the same test using the screen of the key-transmitter instead of the flashlight.

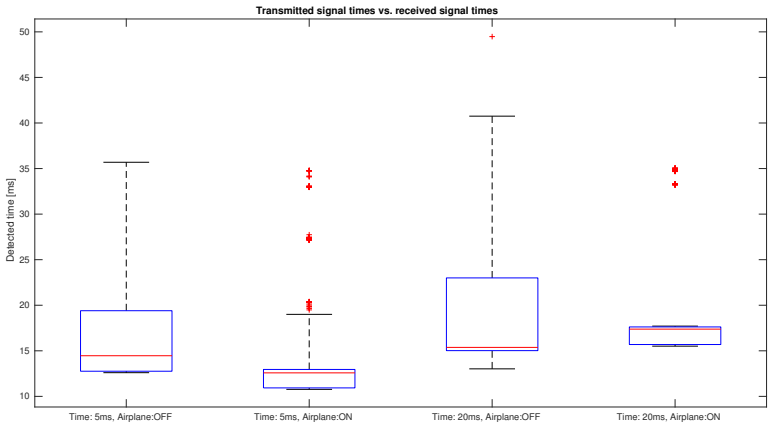

Figure 5.2: Detected times vs. transmitted times utilizing the **screen**.

5.1.1 Throughput and Success Rates

Next, Figures 5.3, 5.4, 5.5 and 5.6 illustrates the throughput versus the success rate of a full transmission. A full transmission is considered a frame of length 16 bytes. Moreover, the transmission is considered a success if the sensor node successfully compares the received CRC value with the calculated one.

These tests were performed using two different smartphones as key-transmitters. Both of which were running on Android version 7.0 (Nougat). Again, all tests were performed both with and without flightmode activated on the key-transmitter. Furthermore, the test environment had an illuminance level of 100 lux. Table 5.1 lists more specific details of the test environment.

Table 5.1: Details of the testing environment during the tests described in Section 5.1.1.

Encoding technique:	PWM (Figures 5.3 & 5.4) & ME (Figures 5.5 & 5.6)
Airplane mode:	ON & OFF
Transmission via:	Flashlight (Figures 5.3 & 5.5) & screen (Figures 5.4 & 5.6)
Transmitted data (pre-shared secret):	"Axis Com and LTH"
Key length:	16 bytes (128 bits)
Total amount of transmitted data (protocol overhead):	22 bytes (168 bits)
Distance between key-transmitter and sensor node	5 cm
Smartphone models:	Samsung Galaxy S6 & LG Nexus 5
Samples taken at each data-point:	25
Noise level:	100 lux

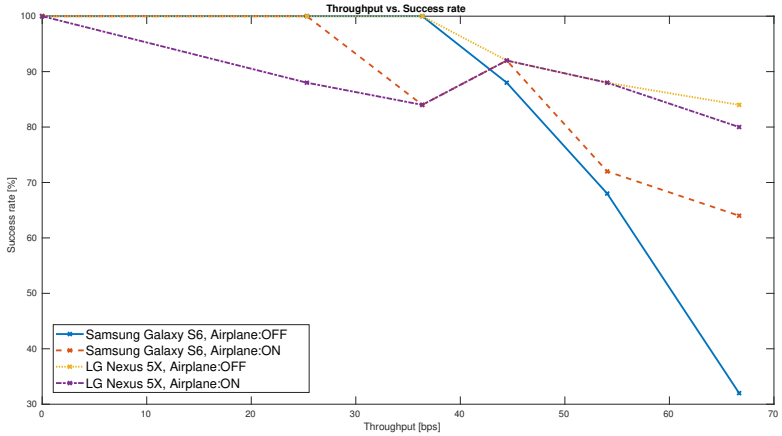

Figure 5.3: Throughput versus success rate when utilizing the **flashlight** and **PWM** as encoding technique.

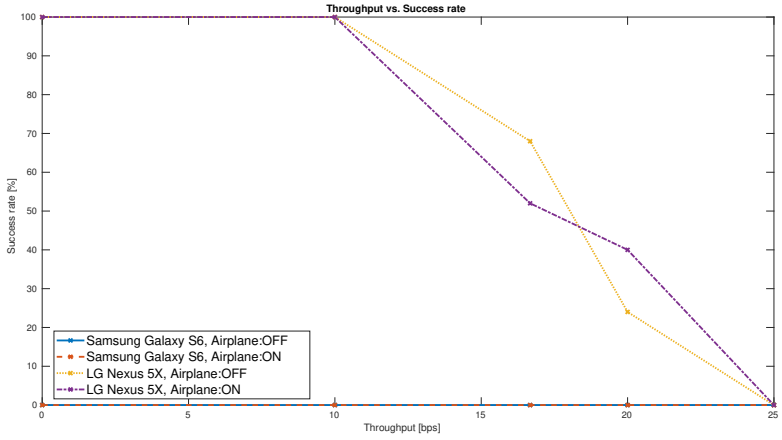

Figure 5.4: Throughput versus success rate when utilizing the **screen** and **PWM** as encoding technique.

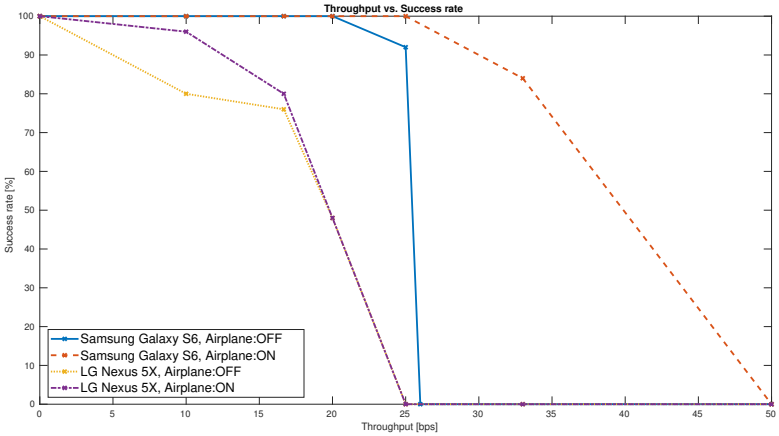

Figure 5.5: Throughput versus success rate when utilizing the **flashlight** and **ME** as encoding technique.

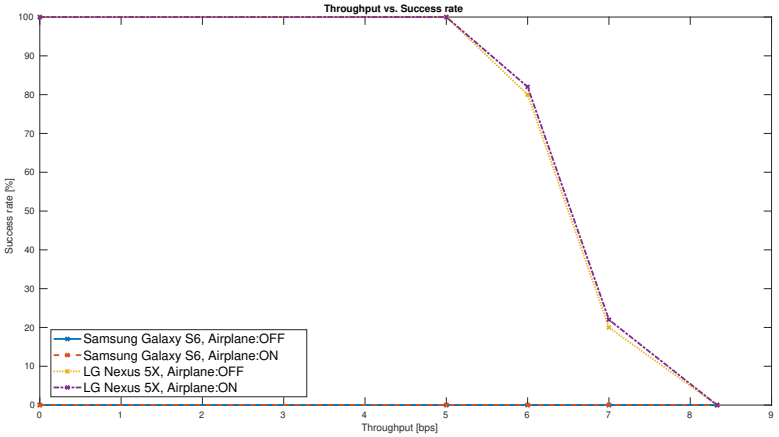

Figure 5.6: Throughput versus success rate when utilizing the **screen** and **ME** as encoding technique.

5.1.2 Screen Processing Times

Now, Figure 5.7 & 5.8 illustrates the time it takes for each subtask to complete the rendering of one frame on the key-transmitters screen. These tests were performed using, again, two different smartphones as key-transmitters. The data is extracted from the smartphones by dumping the "gfxinfo"-file located within the smartphone. The dumping is done using a tool called "dumpsys" provided by Android via the Android Debugging Bridge (adb). The figures are then generated by a Python-script called "slickr" [31], which reads this data and outputs the seen plots.

The color coding represents the following:

- Green: Point of reference set at 60fps, which is 16.16ms per frame.
- Blue: The time it takes to create and update a frame's content.
- Purple: The time it takes to upload the bitmap information to the GPU.
- Red: Time spent by Android's 2D rendering issuing commands to OpenGL.
- Yellow: The time the CPU is waiting for the GPU to finish its work.

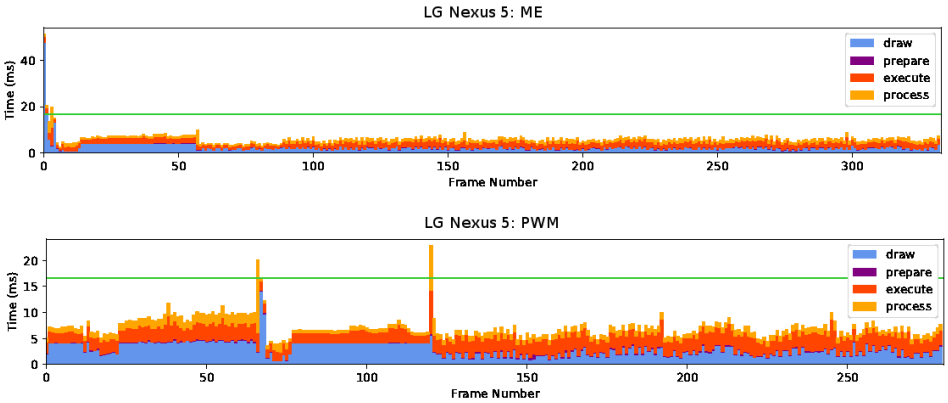

Figure 5.7: Duration of each subtask in the image rendering process of the LG Nexus 5 using both ME & PWM.

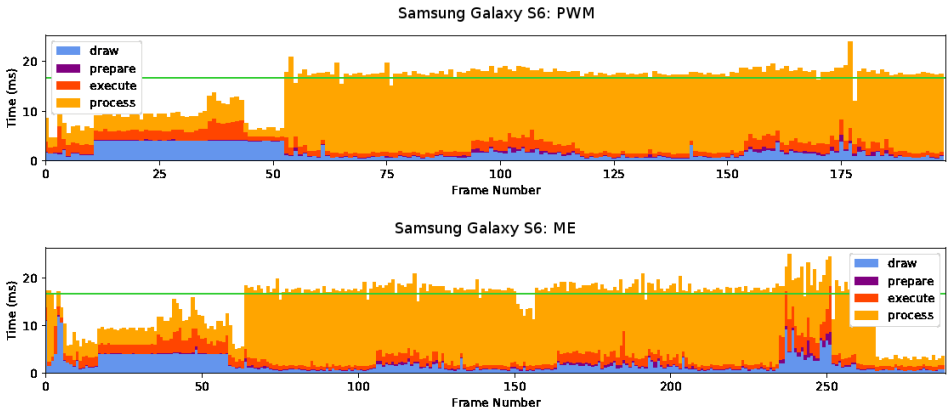

Figure 5.8: Duration of each subtask in the screen image rendering process of the Samsung Galaxy S6 using both ME & PWM.

5.1.3 Average Transmission Times

How the average transmission time is affected when increasing the length of the data to be transmitted is depicted in Figures 5.9 & 5.10. Also, the theoretical time it *should* take to transmit that amount of data is included. The theoretical time is calculated by looking at the bit-pattern of the frame to be transmitted and the specific duration of each bit. The practical time is calculated programmatically, by measuring the time the code takes to complete a full transmission.

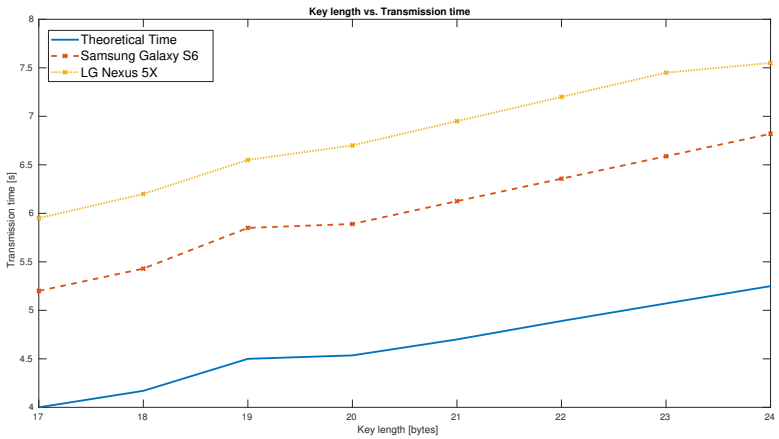

Figure 5.9: Practical transmission times compared to the theoretical time using PWM at 44bps.

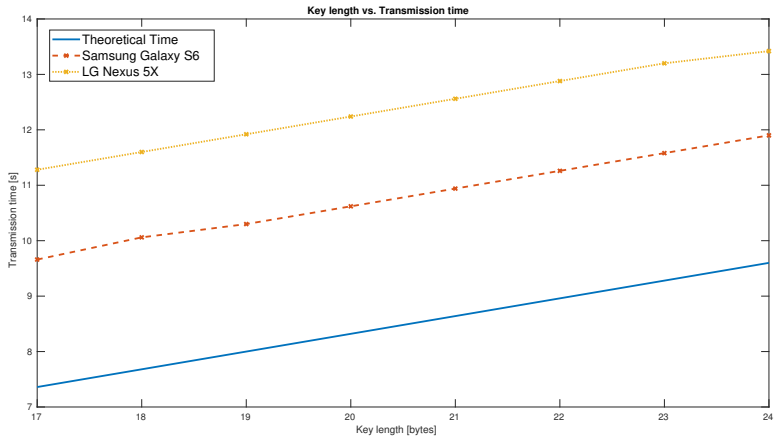

Figure 5.10: Practical transmission times compared to the theoretical time using ME at 25bps.

5.1.4 Increasing Distance and Noise

Figure 5.11 displays how an increasing distance between the key-transmitter and the sensor node affects the success rate of a transmission. The line encoding and transmission speed were chosen according to the most stable settings found in the results described in Section 5.1.1. Consequently, PWM as line encoding technique and a bit rate of 44bps. The noise was kept at an illuminance level of 100 lux during the entire test.

Similarly, Figure 5.12 illustrate how the success rate is affected by an increasing noise level instead. Still using PWM as encoding technique at a transmission speed of 44 bps. The amount of noise was measured with the ambient light sensor on the Samsung Galaxy S6 using the application “Lux Light Meter” developed by “Doggo Apps”. Moreover, the increasing noise was generated using ordinary office lightning as well as the flashlight of a smartphone not being used in the test.

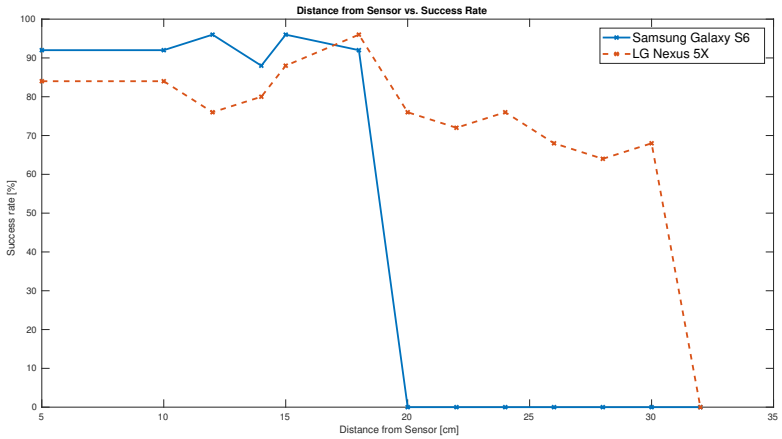

Figure 5.11: Distance versus success rate.

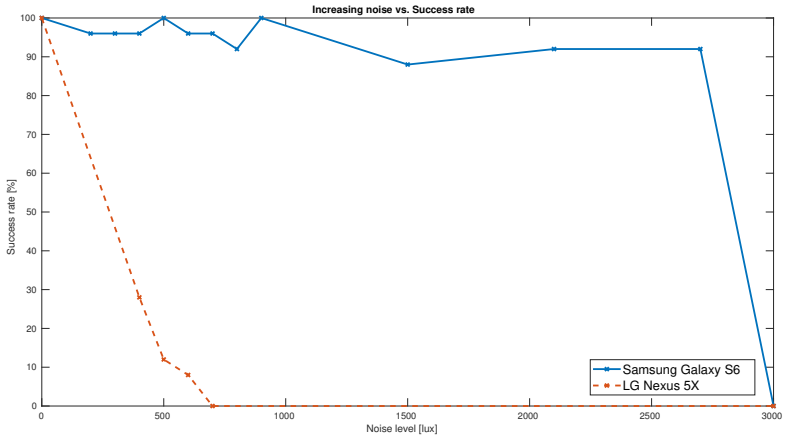

Figure 5.12: Noise level versus success rate.

5.1.5 Resource Consumption

Processing Loads

By measuring how many round-trips the MCU could do in an empty main program loop (`while(1){}`) under two seconds, a baseline for benchmarking was constructed. Then, to calculate the CPU-utilization, the constructed baseline was compared to how many round-trips the MCU was able to complete running the firmware of each prototype. Obviously, the firmware was running the same amount of time. Figures 5.13 & 5.14 illustrates how much load the visual-based prototypes put on the MCU.

Memory Consumption

Using the Arduino IDE, the memory consumed by each prototype's firmware was extracted. This is illustrated in Figure 5.15.

Power Consumption

Turning to power consumption, the Arduino development board (PCB) contains many components not necessary for the sensor node. These extra components consume a lot of power. So, to be able to get a more accurate measurement of power consumption, the MCU was extracted from the PCB. Next, a stable level of 5V was supplied to the MCU with a *EL302* power supply. While the voltage level across a "shunt-resistor" was extracted using an *DSO1024A* oscilloscope. The result of these measurements can be seen in Figures B.1-B.3, where every section of the voltage plot is labeled according to its then current subtask. A schematic of the setup is seen in Figure B.4.

Finally, using these plots we could extract the voltages, V. At the same time, knowing the value of the shunt-resistor, R, the current used by the prototype was calculated with $1[\mathrm{I}] = 1[\mathrm{V}]/1[\mathrm{R}]$. The current is the same throughout the entire schematic due to the serial coupling. Additionally, due to the coupling, the actual voltage used by our circuit is the difference between the 5[V] supplied and the measured voltage at the "shunt-resistor". Finally, the power consumption could be calculated by $1[\mathrm{W}] = 1[\mathrm{V}] * 1[\mathrm{I}]$. The results of each prototype is found in Table 5.2.

Table 5.2: Power consumption of each prototype during decoding and while providing feedback (blinking LED).

		Encoding technique		
		PWM	**ME**	**FSK**
Consumption during sleep: [μW]		19.5	19	-
Consumption when awake and: [mW]	**Decoding**	42.7	46	45.8
	Feedback	58.2	54.9	53.8
Time taken: [s]	**Decoding**	7.08	12.56	10.12
	Feedback	0.72	0.72	0.72
Energy consumed: [mWs]		344.22	617.29	502.23

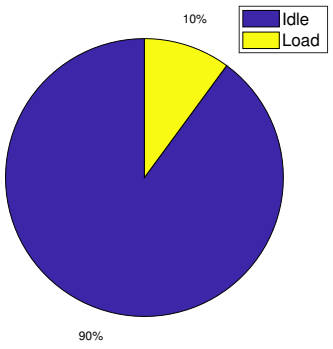

Figure 5.13: CPU-utilization of the **PWM** prototype.

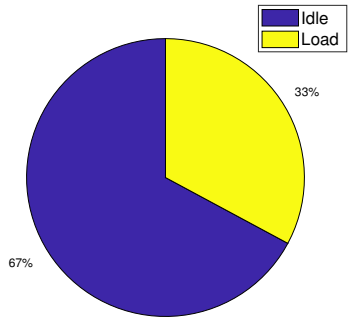

Figure 5.14: CPU-utilization of the **ME** prototype.

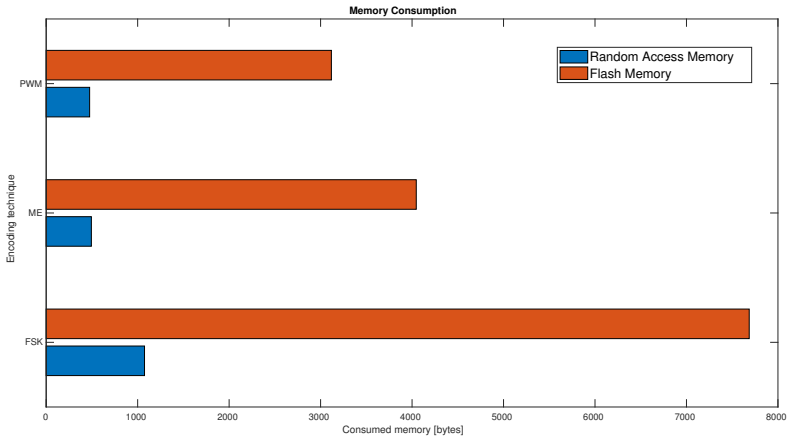

Figure 5.15: Memory consumption of both visual- and audio-based prototypes.

5.2 Audio Based Prototype

This section is dedicated to present the results regarding the audio based solution.

5.2.1 Throughput and Success Rate

In Figure 5.16 it is shown how the success rate of audio-based transmissions is affected by an increasing throughput. Similarly to Section 5.1.1, this test was performed without any *added* noise. In other words, only the noise within the office was present. This noise level was measured to 40dB using the Android application "SoundMeter" developed by the company "Abc Apps".

5.2.2 Increasing Noise and Distance

Now, Table 5.3 lists two tests. Firstly, it displays successful transmissions when different levels of white noise were added on top of the already present 40dB. The number embraced in parentheses represents the total level of noise, which is the added white noise along the noise within the office. The white noise was generated with an additional smartphone (Samsung Galaxy J3) running the application "Noise Generator" made by the company "TMSOFT". Secondly, the table displays the verdict of the transmissions when the distance between the key-transmitter and sensor node is increased. The reason of not plotting these results in a graph is because the verdict can only land on two outcomes, success or failure. Hence, this potentially renders a graph that only goes from one state to the other.

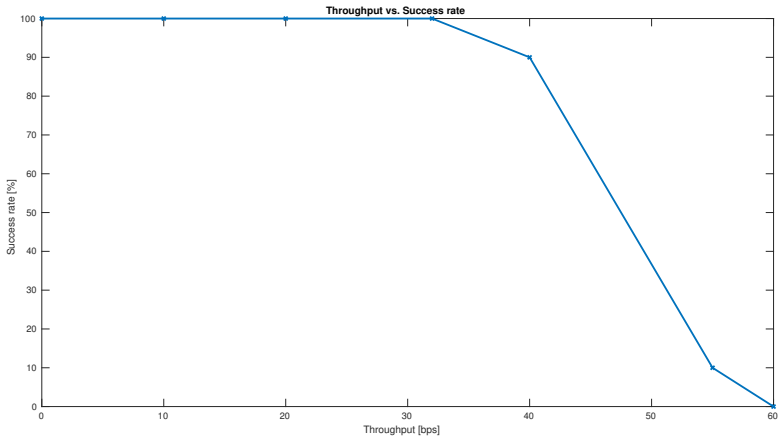

Figure 5.16: Throughput versus success rate when utilizing the key-transmitters loudspeaker and FSK as encoding technique.

Table 5.3: Verdict of transmission with an increasing noise level as well as increasing distance between the devices.

Added white noise [dB]:	20 (50)	25 (55)	30 (60)	35 (65)	40 (70)
Transmission status:	Success	Success	Success	Success	Failed
Distance between key-transmitter and sensor node [mm]:	2	10	15	20	
Transmission status:	Success	Success	Success	Failed	

5.2.3 Resource Consumption

Processing Load

The CPU-utilization of the audio-based protoype is illustrated in Figure 5.17. The CPU-utilization is measured and calculated using the same method desribed in Section 5.1.5.

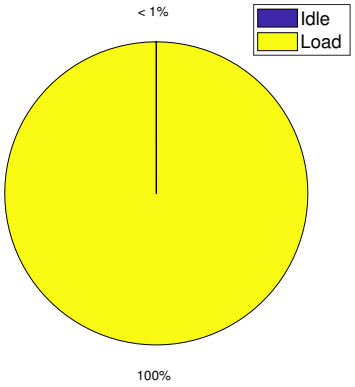

Figure 5.17: CPU-utilization of the audio based prototype.

Memory Consumption

By looking at the bars of *FSK* in Figure 5.15, the amount of memory consumed by the audio based protoype can be seen.

Power Consumption

The amount or power consumed by the firmware used by the audio-based prototype can be seen in Table 5.2.

Chapter 6
Discussion

To begin with, basing the key-transmitter on a smartphone gives the development many possible OOB channels to utilize for communication. However, as great as having many possibilities are, they come at a cost of losing any real-time capabilities. This presents a major problem as most methods of communication has high requirements on timing. Which obviously, is a thing that the Android OS cannot provide.

On top of that, another issue is that there is no such thing as *any* Android-powered smartphone. In other words, the company manufacturing the smartphone has huge implications on how the system will perform. The choice of things like hardware or specific driver implementations affect the viability greatly. This is seen throughout Chapter 5, where the Samsung Galaxy S6 outperforms the LG Nexus 5 in almost every test. Even though they are using the same version of OS.

Because of this, when distributing a system like this it is not sufficient to say "an Android-powered smartphone is required". As one particular smartphone might be able to perform as requested while another may not. So, to be able to provide the user with an accurate recommendation on which smartphone to use, a huge variety of smartphones consisting different makes and models would have to be tested and approved specifically.

6.1 The Problem of Timing

The problem of timing has been a major problem throughout this thesis. Achieving a reliable channel of communication based on a system where one of the devices have the trouble of meeting *any* deadline given to it has been difficult. In this section, we discuss some of the main contributors to this problem and how we have tried to mitigate them.

6.1.1 Android and Deadlines

Figures 5.1 & 5.2 clearly display the inability of the tested smartphones to transmit a signal for an *exact* amount of time. This is true regardless if the smartphones are pressured to send a signal for 5ms or if a time of 20ms is set. Which is highlighted by the fact that the same irregular pattern appears in both cases. Now, even if the "transmission"-thread runs at the highest priority, the OS will still schedule other

services ahead of it from time to time. Thus, the thread misses the deadline more often than not.

When the key-transmitter is set to send a signal for 5ms using the flashlight, a detection interval of ±5ms should be used. Then, the sensor node would have an accuracy of approximately 100% regarding symbol decisions. However, if the screen were to be used under the same circumstances, then the sensor node would have a symbol decision accuracy of approximately 0%. An attempt to mitigate services being scheduled ahead of the transmission-thread was to put the smartphone in *flight mode*.

Activating Flight-Mode

In flight mode, the smartphone turns off all the communication devices available. The hope of doing this was to minimize the number of services that could possibly interrupt the transmission. However, the problem was only mitigated slightly. The only difference when the transmission is done with flight-mode active is that there are more outliers in the upper quartile (making those times less likely). In other words, the receiver could shorten its decision interval to about ±3 milliseconds with approximately 95% decision accuracy. Therefore, we concluded that even if flight mode did not solve our timing problems, it could still be used to slightly increase reliability.

The Audio-based Special Case

Looking at only the timings, our audio-based solution shows a higher success rate in meeting deadlines compared to its counterpart. This is because it is based on FSK, which has a larger symbol size. When using the audio-based prototype, each signal is transmitted for 200ms. This is, for instance, approximately 6 times longer than the longest time used in PWM. With this, seemingly, long signal time the pressure of the key-transmitter is reduced. Still, almost the same amount of throughput is achieved, as the symbol size is a full byte instead of a bit. Then, each bit could theoretically be considered to be $\frac{200}{8} = 25$ms long.

Conclusively, this way we were able to achieve high bit rates while at the same time reducing the probability of the smartphone missing its deadline. Moreover, looking only at this result, FSK is considered to be the best choice to further investigate.

6.1.2 Screens and Refresh Rates

Looking at Figure 5.2, the screen has a much harder time of making the deadline than the flashlight. However, one of the problems when using the screen is that you are restricted not only to unreliable times set by Android but the refresh rate of the screen as well. For instance, both tested smartphones have a screen refresh rate of 60Hz, giving it a theoretical minimum image switching time of 16.6ms. Obviously, it is therefore impossible to send signals as short as 5ms with the screen. Not only does this explain the radical offset when trying to send a signal for 5ms in Figure 5.2, the result can be deemed inconclusive. However, the two rightmost boxes in the same figure illustrate that there still are outliers

showing signals being detected as 50ms long, when it was supposed to be 20ms. Which is a time that the screen with a 60Hz refresh rate should be able to handle.

What we conclude from this is that the key-transmitter using the screen actually has two problems compared to the flashlight based. It does not only have the problem of Android missing deadlines, it also has a problem of different screens having different refresh rates. Where the refresh rate of the smartphone increases the timing problem even further, complicating the solution. Therefore, we believe that the result of this indicates that the way to proceed would be to use the flashlight instead of the screen.

6.1.3 Impact of the Manufacturer

More importantly, something that has the possibility of affecting the deadlines to a point of breaking the entire system, is the manufacturer of the smartphone. Depending on how they choose to implement the Hardware Abstraction Layer, (See Figure 2.11), they control the viability of this system. To put it into perspective, even if Android provides an API to utilize the flashlight, it is the hardware drivers that ultimately execute the operations necessary. Consequently, if the manufacturer has implemented the drivers non-optimally, the times to switch on the flashlight can be extremely slow. Coincidentally we tried this system using an older smartphone model. However, it was found that this, HTC One M9, had not optimized the HAL. This resulted in a request to turn on the flashlight taking about 600ms on that particular smartphone.

The same type of phenomenon also appears when looking at the GPU performance in Figures 5.7 and 5.8. We see that the GPU of the LG Nexus 5 is, for the most part, able to render the requested frames within 60 frames per second. Although looking at the GPU performance of the Samsung, it is barely able to complete the same task. This is also a consequence of the decisions made by the manufacturer. Now, if the GPU misses the deadline of 60fps the result is that the frames start to drop. Frames being dropped translates into losing bits of the transmission. Moreover, the result also correlates to the results being displayed in Figures 5.6 and 5.4. The reason for Samsung not having any success is because the GPU never successfully renders the frames within 60fps. Resulting in dropped frames which translates into the packet missing bits and ultimately transmission failure.

These results emphasize how the specific hardware within the Android impacts the timing of signals for better or worse. It also highlights the importance of choosing a specific model with the right properties. It does not suffice to choose any Android smartphone at will. To mitigate this a specific list of tested and approved smartphones would have to be published.

6.2 Achieved Throughput

By working around the problem of timing, we were able to achieve the bit rates presented in Section 5.1.1 and 5.2.1. Out of these the highest possible throughput we were able to achieve with approximately 100% success rate were:

1. 35bps using the flashlight and PWM as encoding technique.
2. 33bps using the loudspeaker and FSK as encoding technique.
3. 20bps using the flashlight and ME as encoding technique.
4. 10bps using the screen and PWM as encoding technique.
5. 5bps using the screen and ME as encoding technique.

Clearly, no matter which setting we choose the bit rates are not astoundingly high compared to other modern communication systems. Although, given the scope of the communication, these bit rates are considered sufficient. As seen in Figures 5.9 and 5.10, the total time of a transmission when the data is corresponding to an AES-128 based key is between 4 and 8 seconds. While an AES-192 based key would take between 7 and 12 seconds.

Comparing our highest throughput with the highest throughput presented in [21], we see that our smartphone-based key-transmitter only is 12.5% slower than their specialized device. Although, their PDA-based solution is 65% slower compared to our screen-based solution. With this throughput, at least 12 sensors using AES-128 keys, or 6 sensors using AES-192 keys could be configured in under a minute.

Therefore, we conclude this throughput to be viable for the intended purpose. Furthermore, the results show that the technology has evolved further, making a specialized device not able to outperform a similar system based on a smartphone.

6.3 Noise Resiliance and Increasing Distances

We believe that the result of the noise test performed on the visual based prototypes, to be inconclusive. This is based on comparing the results between the increasing noise and increasing distance tests, seen in Figures 5.12 and 5.11. A contradiction could be derived from the fact that if the LG Nexus 5 can have successful transmissions at larger distances, it would imply that its flashlight has a higher intensity. However, if this is the case, it should also be able to handle more noise. Now, looking at Figure 5.12, the LG Nexus 5 is actually more sensitive to noise compared to the Samsung.

6.3.1 Reason for Inconclusive Results

As for why these results are inconclusive, we traced it to the difficulty in introducing stable light-based noise. The fact that the photo-transistor [40] which we used, mostly detects light that comes from an angle within the interval $\pm 10°$, making the angle of the generated noise very important. A slight adjustment in angle impacts the detected value immensely. On top of that, when the key-transmitting device is held above the sensor node, much of the noise is blocked by both the device and the user's arm. These aspects have probably affected the test environment, introducing inconsistent noise values.

Looking at what caused this, the fault is believed to be in the method of testing. Specifically, the way we introduced higher levels of noise. To begin with, we

measured the present noise and found a location which had the desired level. Then, we continued by placing the sensor node at that location and finally conducting the test. This procedure works when the noise levels are kept low. However, getting further into the test, we started to generate noise with a secondary smartphone. This was to achieve levels of noise which otherwise was not present in the environment. By using the secondary smartphone, we also started to introduce the inconsistent values. This is traced to the secondary smartphone being placed in slightly different places each time the test was conducted.

To eliminate the inconsistent values, a more specific way to generate noise should be constructed. On where, especially, the angle of the generated noise is measured when testing.

6.3.2 Increasing Distance

Unlike the inconclusive noise test, Figure 5.11 illustrates a conclusive result. Why this result is considered conclusive is because it shows that the LG Nexus 5 was able to transmit successfully at longer distances. It surpassed the Samsung Galaxy S6 by about 10cm. Considering that the LG Nexus 5 has dual LEDs as light source, it is only natural that with twice the intensity a further distance can be achieved. Again, emphasizing the fact that each smartphone is different, depending on the different hardware installed.

6.3.3 Thoughts on the Audio-Based Prototype

Turning to the fact that the results indicate that the audio-based solution is very sensitive to an increasing distance. Already at a distance of 2cm, the sensor node is unable to detect incoming signals. However, this is because the audio-sensor is set to the lowest possible sensitivity. By forcing the loudspeaker to be very close to the audio-sensor, some of the noise introduced by the environment is eliminated. Table 5.3 shows that decreasing the distance, the prototype was able to handle relatively high amounts of noise.

Therefore, purely based on this result we concluded this prototype to be a success.

6.4 Resource Demands

Resource demands have always been a concern when talking about sensor nodes. This section will discuss the results of the tests performed on the prototypes that were based on the categories: Power consumption, CPU-utilization and memory usage.

6.4.1 Consumption of the Audio-based Prototype

The problem of having low resources available constantly provides an extra problem to consider during development. Especially when trying to decode frequencies using FFT. The FFT algorithm requires large amounts of memory as well as processing power. Figures 5.17 and 5.15 show that our audio-based prototype puts

a 99.99% load on the MCU while utilizing 80% of the available RAM. Another factor to consider is that this prototype never goes into any sleep mode. Consequently, putting it into an unstable condition. As for memory, the firmware uses almost nine times more RAM and three times more flash memory than the solution presented in [21].

Turning to the power consumption, we envisioned that it would be quite high since the prototype is so resource demanding. Surprisingly, the results show that the power consumption during a transmission was of the same magnitude compared to the visual-based prototypes. Therefore, we derived that when the MCU is active (during transmission) it is always working at full capacity. This means it will consume the same amount of power regardless. However, in the long run, the audio-based prototype will consume a lot more power than the visual-based prototypes because it never goes into sleep mode.

Conclusively, even if the audio-based *prototype* is working for our intended purpose, looking at it from a larger perspective, there is neither enough memory nor processing power left to perform any other type of task. This renders the prototype not viable as it stands. If it were to be integrated into a larger system, the main purpose of that system would probably not be to only receive a secret key.

6.4.2 Consumption of the Visual-based Prototypes

With regard to the visual based prototypes, they were a lot less resource demanding. The sensor node decoding PWM signals required the least amount of resources both in terms of processing load and memory. Though it should be noted that the difference in memory consumption between the PWM decoder and the ME decoder was only 19 bytes of RAM and 928 bytes of flash memory. Again, comparing this to [21], our smallest solution is 38% larger in terms of RAM and 77% larger in terms of flash memory. Similarly, compared to [39] our solution is 88% larger in terms of RAM and 32% larger in terms of flash memory. Now, this may seem like much, but the total memory overhead placed on the sensor node is still not very large. The PWM decoder uses 10% of the total RAM available and 23% of the total flash memory available.

As far as processing load, looking at Figures 5.13 and 5.14, *during* transmission PWM put a 10% load on the MCU while ME used 33%. Once the transmission is complete, these prototypes only affect the MCU memory-wise, as they both go into sleep mode.

Concerning the power consumption, as expected, the consumed power when the MCU is in sleep mode is similar. Additionally, the PWM encoded firmware consumes less power for a transmission compared to the ME encoded firmware. The reason behind that is that PWM generally completes a transmission faster. Therefore, it goes into sleep mode faster as well, making it consume less power.

6.5 Security Analysis

This section will provide a theoretical security analysis of this system. Even though this thesis mainly focuses on the actual procedure of sending a key to the sensor

node, and not actually using it, many aspects within security can be considered to alter the viability of this system.

6.5.1 Why Not Placing the Key in EEPROM?

As in [21], we believe that these types of sensors will be sold pre-programmed for their intended purpose. In that case, it means that an end-user would have to trust the retailer to provide them with an uncompromised key. With a long line of production, the company could find this hard to achieve. However, with this solution the company does not have to ensure their trustworthiness since they let the user configure the key themselves. While at the same time, not affecting neither cost nor functionality considerably.

6.5.2 Why Not Use the Wired Interface?

Looking at two things: first, the manufacturing of these sensors are highly pressured in cost; second, if they are sold pre-programmed; then, there is no need to provide a wired interface at all. This solution provides the user with the possibility of configuring their own key while keeping the manufacturing costs low. Without a wired interface, the other option would be forcing the user to trust the already configured key. Not only this but utilizing OOB is great for security as well. It provides protection against attacks such as Man-in-the-Middle since OOB signals are hard to intercept. Basically, because the OOB signals do not propagate far. Even if they would, it would only be in the direction the operator was aiming for. Hence, to intercept the signals of the transmission the adversary would have to be within the operator's sight. In that case, we assume that the operator would notice and refrain from proceeding the transmission.

On top of that, this solution also increases the ease of use. We believe it is intuitively easier to only point the key-transmitter to the sensor node, press a button and start blinking upon it. Compared to connecting it to a computer via some type of interface and perform the configuration there. In other words, the knowledge of the person doing the configuration is reduced to only require that person of having basic knowledge in smartphone usage. Additionally, we also think that the speed of configuration is increased by using our solution as there are no physical cables that need to be connected over and over again.

6.5.3 Authentication of the Key-Transmitter

As this solution stands, there is currently no authentication of the transmitting device. This means that anyone with that application installed on their smartphone has the possibility of initiating a transmission. On the one hand, in terms of the specific use case described in Section 1.1, this would easily be detected. If this was to happen, the compromised sensor would be detected by the base station. Since the sensor would be using another key compared to that of the base station. Now, solving this could be achieved by making the base station stop communications with that sensor node. It would, also, notify the operator. Consequently, this would become a Denial-of-Service attack and not a way of reading the messages

being sent. Now, the attacker could potentially use this DOS attack to force the base station to stop communicating with the sensor node as a way to bypass it. The attack would be successful in stopping the detection of breaking the glass, however, a notification of the DOS would indicate an attack all the same.

On the other hand, looking at this from a general perspective, this imposes a bigger problem. It is impossible to predict what an attacker might be able to do with unauthenticated configuration mechanics. However, what is not impossible, is to predict that something affecting the system could happen. To deal with this, we think the solution lies in protecting the receiving hardware by means of physical design. The sensor (photo-transistor or microphone) could be physically covered when deployed. Now, adding a mechanic to detect any tampering of this physical cover would trigger a tampering alarm. This alarm would indicate a compromised sensor node and the system's owner could take action.

Conclusively, the reason we feel that physical security is better than an authentication process in the actual transmission is simple. Not only would an authentication procedure of the transmissions require bidirectional communication, but the process of configuring would not be as easy to complete. Moreover, as these solutions are trying to solve the "*pre* key-exchange problem", placing authentication on the transmission would only extend this problem to another platform. Instead of making improvements to the actual problem of getting the keys into the sensor nodes.

6.5.4 Storing the Encryption Keys in Android

It should be noted that there definitely are some dangers regarding security given the fact that the key is stored on the smartphone. These risks are largely put on the person using the smartphone, as it becomes their responsibility to handle the smartphone with care and not exposing it to potential viruses or hacks. To mitigate this risk we make sure that the sensitive data is never really stored on the smartphone for longer than necessary. As soon as the transmission is complete, on all parties involved, the variable holding the key should be wiped.

This makes the time window of such an attack as little as possible, however the risk does remain. Nevertheless, we think that given the background of the problem, the user should be aware of this and therefore act accordingly.

Chapter 7
Conclusion

To begin with, with this thesis we have shown that it is possible to use an Android-powered smartphone to perform the configuration of pre-shared secrets in a sensor node via two different OOB based side-channels. This has been achieved by building prototypes which are able to successfully transmit data to a simulation of a sensor node. Moreover, the tests performed on the prototypes have shown that a visual-based approach has the best possibilities for further development. Both in terms of effectiveness as well as the impact on manufacturing costs. The visual-based solution is less complex in terms of the sensor node's hardware schematics, consequently, being cheaper as well. Furthermore, the receiver software is less complex in the visual-based prototypes compared to that of the audio-based. Again, highlighting the visual-based prototypes as a better candidate. The results also strongly suggest that the flashlight of the smartphone should be further investigated, as it has shown better reliability and higher throughput, compared to the screen.

Secondly, we have managed to achieve reliable communication at bit rates fast enough to do the intended task of configuring a pre-shared secret that is no longer than 256 bits. Which is a good size for symmetric encryption. The bit rates achieved has been approximately the same magnitude as previous research published 2009 [21]. The difference is that their high bit rates were produced by a specialized device produced solely for the purpose of configuring a sensor node. Our thesis extends these findings by applying some of the same principles to an *non-specialized* device.

Lastly, the result of this proof-of-concept shows that it is possible to use the various hardware provided by a modern smartphone to achieve communication without the need of any sophisticated network protocols. Hopefully, these results will act as a baseline for further development, both within the areas of securing WSN and smartphone-based OOB communication generally.

Chapter 8
Future Work

This chapter introduces some ideas that we think should be further researched. The suggestions are based both on the conclusions as well as lessons learned throughout the thesis.

8.1 Optimizing the Screen

To begin with, improving the screen-based solution can be done via hardware or software.

First, an easy to test improvement would be to use a smartphone that has a screen with higher refresh rate. As our results concluded, the refresh rate of the smartphone put a limit of what throughput can be expected. There are today commercially available smartphones which have refresh rates as high as 120Hz. One of which are presented in [27]. Now, this is mainly suggested because it is easy to test while still providing insight if the screen has more potential than our conclusions might have displayed. Only using a higher refresh rate would potentially double the throughput but, otherwise, still have the problem of meeting deadlines.

Second, we believe that developing the application with the Native Development Kit instead of the Software Development Kit might further improve the performance. Using the NDK the application would be written in C/C++ instead of Java with a more direct access to lower levels of the hardware. For instance, using "OpenGL" which is a graphics API for 2D and 3D applications [24]. With this, one can expect a more optimized performance of the image rendering process. Similarly, the same result could be done by optimizing the hardware drivers themselves to achieve the same effect. However, the latter might be considered specializing the device.

8.2 Installing RTDroid

We believe that installing RTDroid would greatly improve the communication. The largest problem we faced was the one of adapting and working with an unreliable key-transmitter. However, utilizing this would potentially eliminate this problem entirely making huge improvements to both throughput and reliability.

Yet again, using this could potentially make the smartphone be considered as a specialized device. However, if that is of no concern then we believe this is the absolute way to proceed.

8.3 Increasing the Symbol Size

8.3.1 Different Levels of the Flashlight

Some of the most modern smartphone models are starting to introduce the functionality of setting the flashlight to different intensity levels. Today, this can be done on a Samsung Galaxy S8. This functionality has the potential to improve our best solution even further. With this, the flashlight-based communication can improve both in throughput and reliability. Having different levels, the times of each signal could be made longer, making the Android less likely to miss the deadline. While at the same time increase the throughput. The effect of this is already proven in the audio-based solution having higher reliability while maintaining the throughput.

8.3.2 Encoding with Colours

Similarly to Section 8.3.1, introducing colours to the encoding could potentially improve the throughput and reliability of the screen considerably. However, the sensor node would have to change the hardware to be able to detect colours.

8.4 Decreasing Power Consumption

Looking at the results of our prototypes regarding power consumption, we believe this could be improved upon even further. Our prototypes are mainly focused on utilizing the best possible sleep mode to reduce power consumption. This is because we felt it to be a good general guideline to what could be expected. Although there are numerous other additional ways to decrease the power consumption even further. However, this would depend on how the finalized sensor node is supposed to work. For instance, with a specific setting, more components could be disabled or even switch out the MCU to a lesser grade one.

References

[1] Gartner says 8.4 billion connected 'things' will be in use in 2017, up 31 percent from 2016, February 7, 2017. [Online]. Available: `http://www.gartner.com/newsroom/id/3598917`. [Accessed: 26- Oct- 2017].

[2] Gartner says worldwide sales of smartphones grew 9 percent in first quarter of 2017. *Electronics Bazaar*, May 24, 2017. [Online]. Available: `https://www.gartner.com/newsroom/id/3725117`. [Accessed: 24- Oct- 2017].

[3] T. Arampatzis, J. Lygeros, and S. Manesis. A survey of applications of wireless sensors and wireless sensor networks. In *Proceedings of the 2005 IEEE International Symposium on, Mediterrean Conference on Control and Automation Intelligent Control, 2005.*, pages 719–724, 2005.

[4] K. Ashton. That 'internet of things' thing. *RFiD Journal*, 22(7), 2011.

[5] C. Baker. 'crc32'. 2017. [Online]. Available: `https://github.com/bakercp/CRC32`. [Accessed: 28- Feb- 2018].

[6] E. Barker. Recommendation for key management, part 1: General. Technical report, U.S. Department of Commerce, Jan 2016.

[7] A. Bartoli, E. Medvet, and G. Davanzo. Rainbow crypt: Securing communication through a protected visual channel. In *2011 11th International Conference on Intelligent Systems Design and Applications*, pages 72–75, 2011.

[8] J. Benjumea, A. V. Medina, S. Martin, and G. Sanchez. Using laboratory to improve understanding of 802.3 physical characteristics. In *2009 39th IEEE Frontiers in Education Conference*, pages 1–6, 2009.

[9] S. A. Cassero. *Design and Analysis of Arduino, Raspberry Pi, and Xbee Based Wireless Sensor Networks.* PhD thesis, 2016.

[10] C. Castelluccia and P. Mutaf. Shake them up!: a movement-based pairing protocol for cpu-constrained devices. In *Proceedings of the 3rd international conference on Mobile systems, applications, and services*, pages 51–64. ACM, 2005.

[11] C. Castelluccia and A. Spognardi. Rok: A robust key pre-distribution protocol for multi-phase wireless sensor networks. In *2007 Third International Conference on Security and Privacy in Communications Networks and the Workshops - SecureComm 2007*, pages 351–360, 2007.

[12] H. Chan, A. Perrig, and D. Song. Random key predistribution schemes for sensor networks. In *2003 Symposium on Security and Privacy, 2003.*, pages 197–213, 2003.

[13] E. Condes. 'arduinofft'. 2018. [Online]. Available: `https://github.com/kosme/arduinoFFT`. [Accessed: 28- Feb- 2018].

[14] Atmel Corporation. Atmega328/p datasheet complete. Technical report, Atmel Corporation, Nov 2016. [Online]. Available: `http://ww1.microchip.com/downloads/en/DeviceDoc/Atmel-42735-8-bit-AVR-Microcontroller-ATmega328-328P_Datasheet.pdf`. [Accessed: 26- Feb- 2018].

[15] DFRobot. 'analog sound sensor'. 2013. [Online]. Available: `http://image.dfrobot.com/image/data/DFR0034/V2.0/Analog%20Sound%20Sensor%20SCH.pdf`. [Accessed: 20- Feb- 2018].

[16] DFRobot. 'analog sound sensor sku: Dfr0034'. 2017. [Online]. Available: `https://www.dfrobot.com/wiki/index.php/Analog_Sound_Sensor_SKU:_DFR0034`. [Accessed: 28- Feb- 2018].

[17] L. Eschenauer and V. D. Gligor. A key-management scheme for distributed sensor networks. In *Proceedings of the 9th ACM Conference on Computer and Communications Security*, CCS '02, pages 41–47, New York, NY, USA, 2002. ACM.

[18] R. Faludi. *Building wireless sensor networks.* O'Reilly, Beijing [u.a.], 1. ed. edition, 2011.

[19] D. Feldman. Photovoltaic (pv) pricing trends: Historical, recent, and near-term projections. Technical report, National Renewable Energy Laboratory, April 21 2014. [Online]. Available: `https://escholarship.org/uc/item/06b4h95q/`. [Accessed: 19- Mar- 2018].

[20] S. Ferdoush and X. Li. Wireless sensor network system design using raspberry pi and arduino for environmental monitoring applications, 2014.

[21] M. Gauger, O. Saukh, and P. J. Marron. Enlighten me! secure key assignment in wireless sensor networks. In *2009 IEEE 6th International Conference on Mobile Adhoc and Sensor Systems*, pages 246–255, 2009.

[22] D. Gollmann. *Computer Security.* John Wiley & Sons, Ltd., West Sussex, United Kingdom, 3 edition, 2011.

[23] M. T. Goodrich, M. Sirivianos, J. Solis, G. Tsudik, and E. Uzun. Loud and clear: Human-verifiable authentication based on audio. In *26th IEEE International Conference on Distributed Computing Systems (ICDCS'06)*, page 10, 2006.

[24] Khronos group. 'opengl overview'. 2018. [Online]. Available: `https://www.opengl.org/about/`. [Accessed: 28- Feb- 2018].

[25] J. Hwang and Y. Kim. Revisiting random key pre-distribution schemes for wireless sensor networks. In *Proceedings of the 2Nd ACM Workshop on Security of Ad Hoc and Sensor Networks*, SASN '04, pages 43–52, New York, NY, USA, 2004. ACM.

[26] Google Inc. 'android activity structure guide'. 2018. [Online]. Available: `https://developer.android.com/guide/components/activities/index.html`. [Accessed: 1- Mar- 2018].

[27] Razor Inc. 'razor phone - 120hz mobile master race'. 2018. [Online]. Available: `https://www.razerzone.com/campaigns/120hzmobilemasterrace`. [Accessed: 28- Feb- 2018].

[28] L. Kleinrock. Information flow in large communication nets. *RLE Quarterly Progress Report*, 1, 1961.

[29] C. Kuo, M. Luk, R. Negi, and A. Perrig. Message-in-a-bottle. SenSys '07, pages 233–246. ACM, Nov 6, 2007.

[30] B. M. Leiner, V. G. Cerf, D. D. Clark, R. E. Kahn, L. Kleinrock, D. C. Lynch, J. Postel, L. G. Roberts, and S. Wolff. A brief history of the internet. *SIGCOMM Comput. Commun. Rev.*, 39(October 2009):22–31, 2012.

[31] E. Leong. 'python and bash scripts for gpu frame rendering'. 2018. [Online]. Available: `https://github.com/ericleong/slickr`. [Accessed: 27- Feb- 2018].

[32] G. Lindell. Introduction to digital communications, August 2006.

[33] T. Marill and L. Roberts. Toward a cooperative network of time-shared computers. AFIPS '66 (Fall), pages 425–431. ACM, Nov 7, 1966.

[34] J. M. McCune, A. Perrig, and M. K. Reiter. Seeing-is-believing: using camera phones for human-verifiable authentication. In *2005 IEEE Symposium on Security and Privacy (S&P'05)*, pages 110–124, 2005.

[35] A. J. Menezes, P. C. Van Oorschot, and S. A. Vanstone. *Handbook of applied cryptography*. CRC Press, Boca Raton [u.a.], 1997.

[36] J. G. Proakis and D. G. Manolakis. *Digital Signal Processing*. Pearson, Upper Saddle River, NJ, 4. ed., pearson new. internat. ed. edition, 2014.

[37] I. K. Rhee, J. Lee, J. Kim, E. Serpedin, and Y. C. Wu. Clock synchronization in wireless sensor networks: an overview. *Sensors (Basel, Switzerland)*, 9(1):56–85, 2009.

[38] L. Roberts. Multiple computer networks and intercomputer communication. SOSP '67, page 3.6. ACM, Jan 1, 1967.

[39] R. Roman and J. Lopez. Keyled - transmitting sensitive data over out-of-band channels in wireless sensor networks. In *2008 5th IEEE International Conference on Mobile Ad Hoc and Sensor Systems*, pages 796–801, 2008.

[40] OSRAM Opto Semiconductors. Silicon pin photodiode version 1.3: Sfh213. Technical report, OSRAM Opto Semiconductors GmbH, Dec, 21 2015. [Online]. Available: `http://www.osram-os.com/Graphics/XPic4/00211451_0.pdf`. [Accessed: 3- Dec- 2017].

[41] S. Smith. *Digital Signal Processing: A Practical Guide for Engineers and Scientists*. Newnes, Oxford, 2013.

[42] J. A. Stankovic. Wireless sensor networks. *Computer*, 41(10):92–95, 2008.

[43] P. Tague and R. Poovendran. A canonical seed assignment model for key predistribution in wireless sensor networks. *ACM Trans.Sen.Netw.*, 3(4), oct 2007.

[44] A. Tyagi, J. Kushwah, and M. Bhalla. Threats to security of wireless sensor networks. In *2017 7th International Conference on Cloud Computing, Data Science & Engineering - Confluence*, pages 402–405, 2017.

[45] C. Wang, W. Duan, J. Ma, and C. Wang. The research of android system architecture and application programming. In *Computer Science and Network Technology (ICCSNT), 2011 International Conference on*, volume 2, pages 785–790. IEEE, 2011.

[46] Y. Yan, S. Cosgrove, V. Anand, A. Kulkarni, S. H. Konduri, S. Y. Ko, and L. Ziarek. Real-time android with rtdroid. In *Proceedings of the 12th annual international conference on Mobile systems, applications, and services*, pages 273–286. ACM, 2014.

[47] N. S. A. Zulkifli, F. K. Che Harun, and N. S. Azahar. Xbee wireless sensor networks for heart rate monitoring in sport training. In *2012 International Conference on Biomedical Engineering (ICoBE)*, pages 441–444, 2012.

Appendix A
Software Schematics

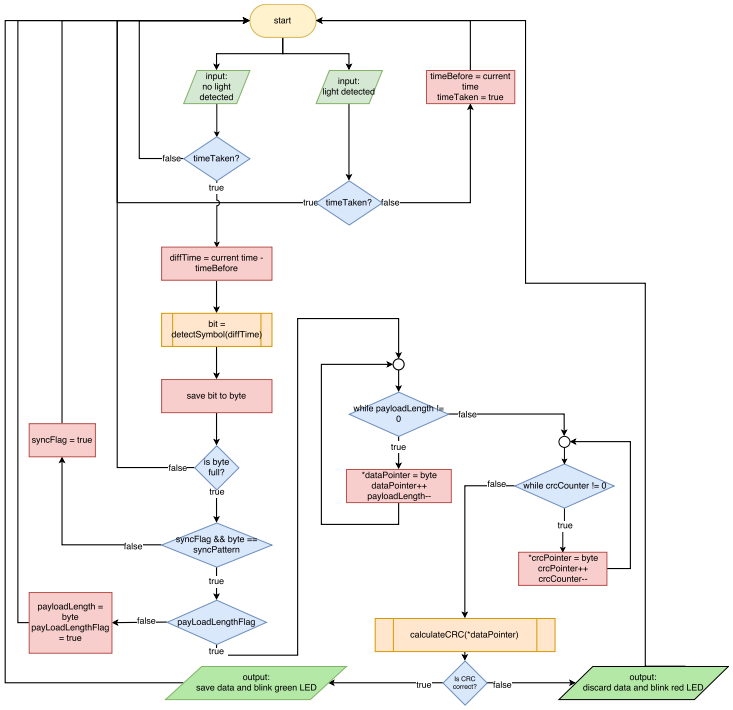

Figure A.1: General flowchart of the sensor node's firmware.

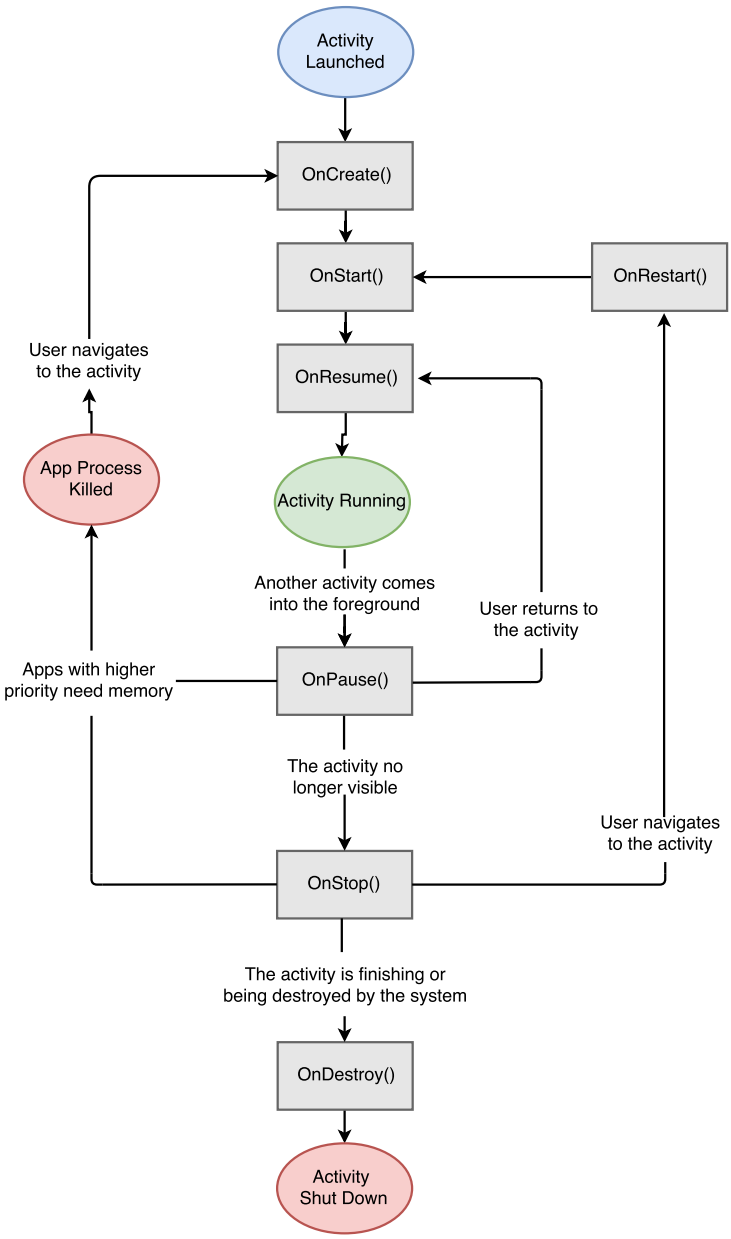

Figure A.2: Application states of Android

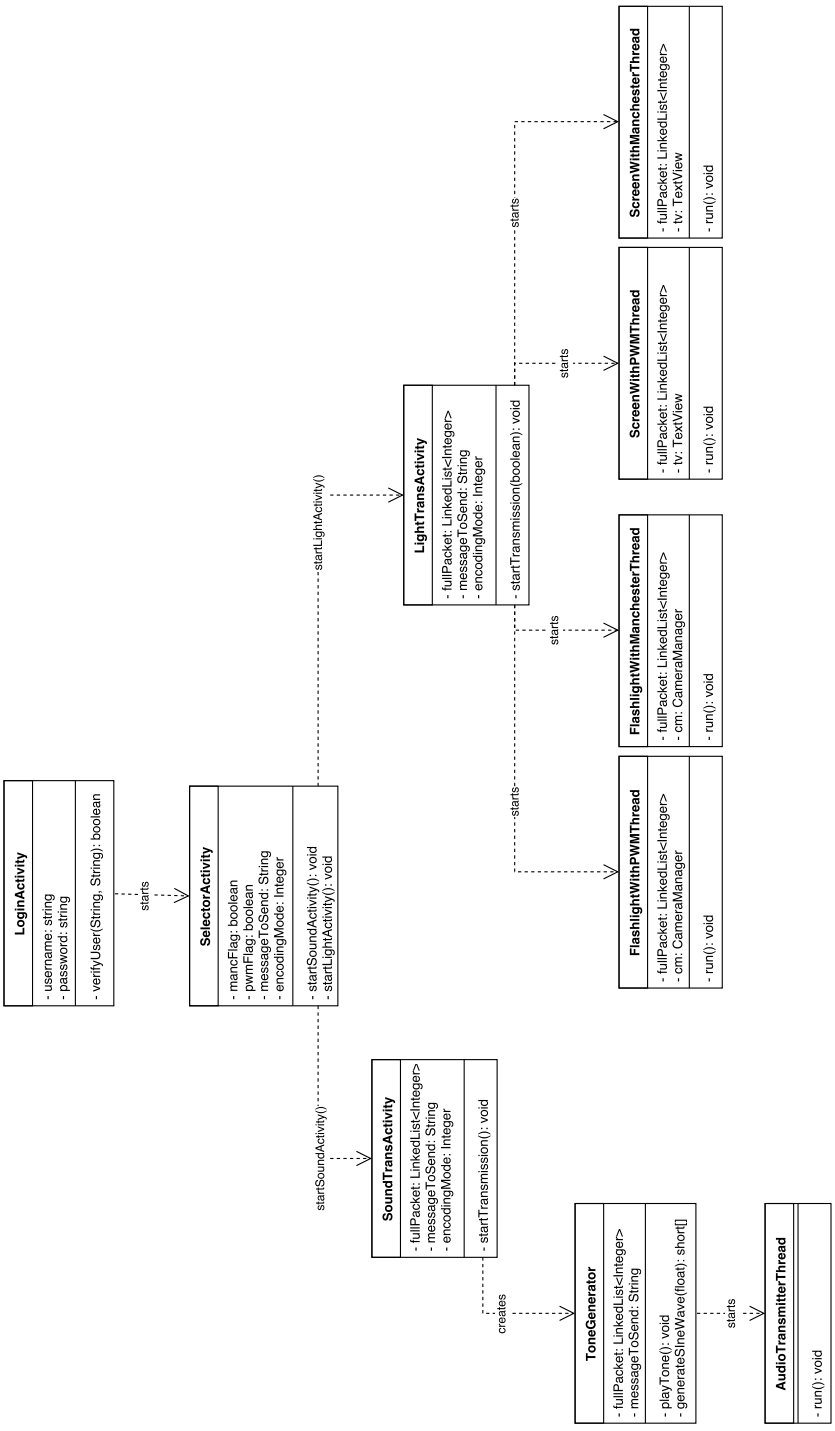

Figure A.3: Basic UML of the key-transmitters software.

Appendix B
Power Consumption Figures

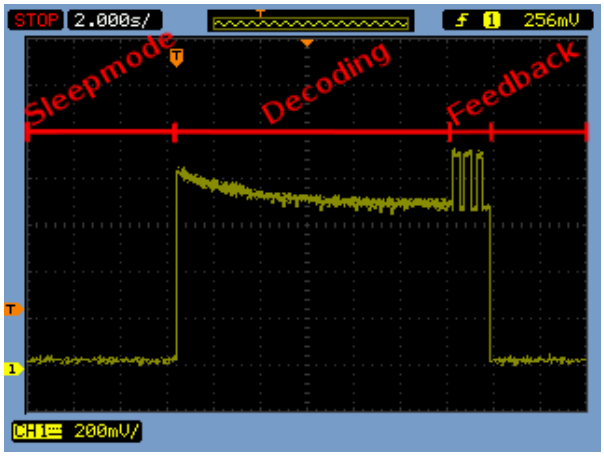

Figure B.1: Voltage plot of a **ME** transmission.

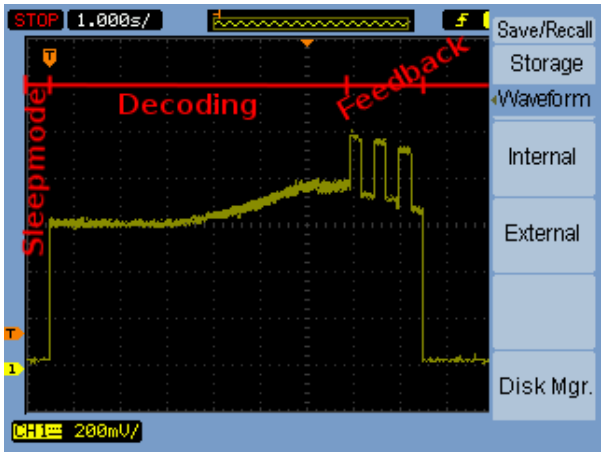

Figure B.2: Voltage plot of a **PWM** transmission.

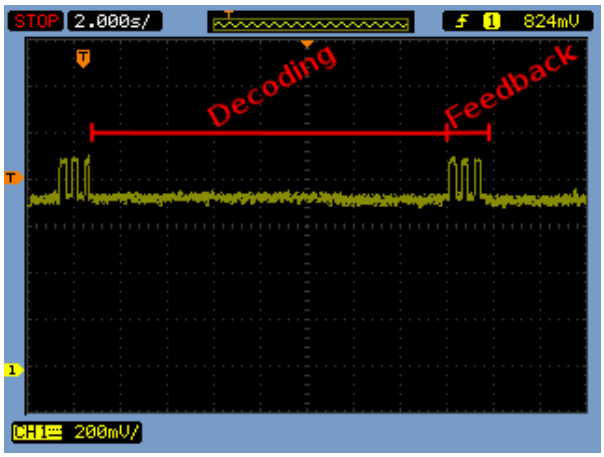

Figure B.3: Voltage plot of a **FSK** transmission.

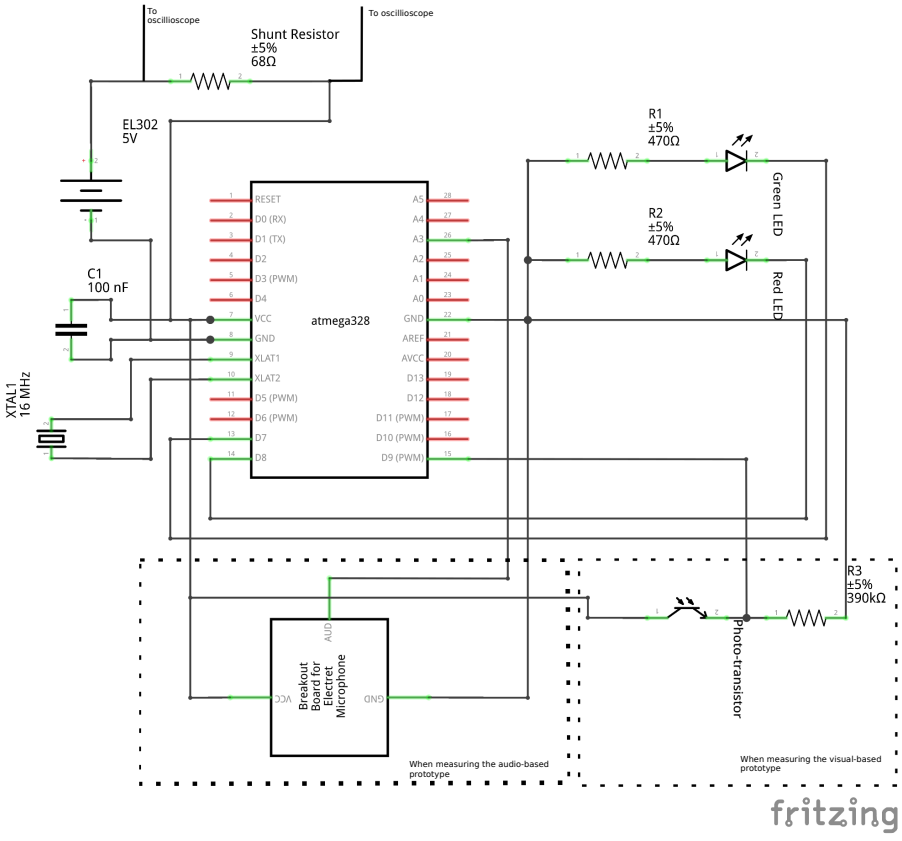

Figure B.4: Schematic of the setup to measure power consumption.

Printed by Tryckeriet i E-huset, Lund 2018

LUND
UNIVERSITY

Series of Master's theses
Department of Electrical and Information Technology
LU/LTH-EIT 2018-623

http://www.eit.lth.se