Master Thesis
at Columbia University, Department of Computer Science

Secure Authentication of Remote IoT Devices Using Visible Light Communication: Receiver Design and Implementation

Submitted by:	Alexander Linssen 1141274
Supervising Professors:	Professor Dr. Henning Schulzrinne Professor Dr. Gabi Dreo Rodosek
Advisors:	Dipl.-Ing. Jan Janak Marcus Knüpfer, M.Sc.
Date:	August 24, 2018

Universität der Bundeswehr München
Fakultät für Informatik
Institut für Technische Informatik

Abstract

Since the number of deployed Internet of Things (IoT) devices in households is increasing significantly every year and the trend is expected to continue, there is a need of secure ways to authenticate those devices. Due to a lack of trusted Certificate Authorities (CA) for IoT devices in local networks, authentication of remote IoT devices requires physical access or trusted third parties.

To eliminate the need to rely on a trusted third party or a proximity-based authentication solution, we designed and built a prototype as a secure authentication system for remote IoT devices based on Visible Light Communication (VLC). The visible light channel serves as an Out-of-band (OOB) channel resistant to active Man-In-The-Middle (MITM) attacks. Our system does not require special equipment, any Android based smartphone can run the receiver application and the transmitter only requires an ordinary tri-color LED. Experiments show that the transmission of a SHA-256 fingerprint takes up to 8 seconds over a 4 meter distance using 4-level Frequency Shift Keying (FSK) modulation.

The concept of this new approach is separated in two parts. This thesis describes the design and implementation of the Android-based receiver. Hagen Odenthal's thesis (UniBwM, August 2018) "Secure Authentication of Remote IoT Devices Using Visible Light Communication: Transmitter Design and Implementation" [24] describes the design and implementation of the transmitter.

Abstract (German)

Die Anzahl der eingesetzten Internet of Things (IoT) Geräte steigt von Jahr zu Jahr erheblich und ein Ende dieses Trends ist vorerst nicht absehbar. Daher besteht ein hoher Bedarf an sicheren Authentifizierungsverfahren. Aufgrund von fehlenden Certificate Authorities (CA) für IoT Geräte im lokalen Netzwerk sind physischer Zugriff oder vertrauenswürdige Drittparteien für die Authentifizierung erforderlich.

Um unabhängig von Drittparteien und direktem Zugriff auf das Gerät zu sein, wurde ein Prototyp zur sicheren Authentifizierung von entfernt gelegenen IoT Geräten mithilfe von Visible Light Communcation (VLC) konzipiert und entwickelt. Der Übertragungskanal Licht dient in diesem System als zusätzlicher Kanal, welcher resistent gegen aktive Min-In-The-Middle (MITM) Angriffe ist. Das System erfordert keine spezielle Ausrüstung, da die Applikation des Empfängers auf jedem Android basierten Smartphone ausgeführt werden kann und das IoT Gerät lediglich eine drei Farben LED benötigt. Experimente zeigen, dass die Übertragung eines SHA-256 Fingerprints mittels Frequenzmodulation 4 in bis zu 8 Sekunden aus vier Metern Entfernung möglich ist.

Das Konzept des Systems ist in zwei Teile gegliedert. Diese Arbeit befasst sich mit dem Design und der Implementierung des Empfängers. Die Arbeit von Hagen Odenthal (UniBwM, August 2018) "Secure Authentication of Remote IoT Devices Using Visible Light Communication: Transmitter Design and Implementation" [24] setzt sich mit dem Sender auseinander.

Acknowledgments

In this chapter, I as the author of this thesis and Hagen Odenthal as the author of [24], would like to thank our supervisors Professor Dr. Henning Schulzrinne and Professor Dr. Gabi Dreo Rodosek for giving us the opportunity to work on our master's project in the Internet Real Time Lab and at this truly great university.

To the same level we would like to thank our advisor Jan Janak, at Columbia University, for his extraordinary support during the project and also, for being second reader of our theses.

We would also like to thank Marcus Knüpfer, our advisor at Universität der Bundeswehr München, for all his guidance.

Last but not least, we would like to thank our employer, the Bundeswehr (German Armed Forces), to enable us to write our master's theses in the United States as international students as well as improving our inter-cultural awareness and language skills.

Contents

1 Introduction **10**

2 Problem Description **13**

3 Concept Requirements **15**

4 Background **17**

4.1 Visual Light Communication 17

4.1.1 Color Spaces 17

4.1.2 Camera Model 19

4.2 Data Transmission 21

4.2.1 Modulation 21

4.2.2 Forward Error Correction 22

5 Related Work **24**

5.1 Authentication of IoT Devices 24

5.2 Prior Research on Visual Light Communication 25

6 System Overview **27**

6.1 Sketch of Operation 27

6.2 System Architecture 28

7 Receiver Design and Implementation **30**

7.1 Concurrency Model 30

7.2 Receiving Camera Frames 32

7.3 Processing Camera Frames 32

7.3.1 Color Detection 33

7.3.2 Demodulation 33

7.3.3 Sampling 34

7.4 Data Decoding Architecture 36

7.4.1 Line Decoder 36

7.4.2 Packet Decoder 37

7.4.3 RaptorQ Decoder 38

7.5 User Interface 39

8 Evaluation **40**

8.1 Hardware Setup 40

8.2 Laboratory Experiment I - Color Detection 40

8.2.1 Auto White Balancing (AWB) Experiment 41

8.2.2 Distance Experiment 43

8.3 Laboratory Experiment II - Performance Measurements 47

8.4 Laboratory Experiment III - Performance in Practice 50

9 Results and Analysis 51

10 Security Considerations 53

11 Conclusion 55

11.1 Limitations 56

11.2 Implications for Future Research 56

Bibliography 58

List of Figures

1.1 Authentication concepts using a channel resistant to active MITM attacks . . . 11

4.1 VLC . . . 17
4.2 Color spaces . . . 18
4.3 The Android Camera2 capturing frames . . . 20
4.4 Square Bayer filter on a pixel array . . . 21
4.5 Modulation techniques amplitude and frequency modulation . . . 22

6.1 Authentication scenario using VLC . . . 27
6.2 Components of transmitter and receiver . . . 29

7.1 Multi-threading setup . . . 31
7.2 Stage 1 image processing . . . 32
7.3 Stage 2 image processing - processing camera frames . . . 32
7.4 Euclidean distance in RGB color model . . . 34
7.5 Receiver capturing frames in predefined interval . . . 35
7.6 Receiver capturing frames in shifted interval . . . 35
7.7 Stage 2 image processing - data decoding . . . 36
7.8 Datagrams sent over the visible light channel . . . 36
7.9 Line decoder state machine . . . 37
7.10 Packet content . . . 38
7.11 FEC packet . . . 39
7.12 Receiver application UI during transmission . . . 39

8.1 White balancing Samsung Galaxy S7 . . . 42
8.2 Short distance color detection experiment using various smartphones as receivers . . . 44
8.3 Long distance color detection experiment using various smartphones and receiver . . . 46
8.4 Short distance performance measurement experiment using various smartphones and receiver . . . 47
8.5 Long distance performance measurement experiment using various smartphones and receiver . . . 49

List of Tables

7.1 Color Detection 34

8.1 Transmitter Setup 40
8.2 Receiver Setup 40

Acronyms

API	application programming interface
AWB	automatic white balance
CA	certificate authority
CMOS	complementary metal oxide semiconductor
CRC	cyclic redundancy check
EWMA	exponential weighted moving average
FEC	forward error correction
FPS	frames per second
FSK	frequency shift keying
HSV	hue saturation value
IoT	internet of things
LED	light-emitting diode
MITM	man-in-the-middle
OOB	out-of-band
P2P	peer-to-peer
PAKE	password authenticated key exchange
PBC	push button configuration
PKI	public key infrastructure
RGB	red green blue
SE	social engineering
TLS	transport layer security
UI	user interface
VLC	visible light communication
WPS	Wi-Fi protected setup

1 Introduction

The number of deployed Internet of Things (IoT) devices is increasing significantly every year. The Ericsson Mobility Report (June 2016) [11] forecast for 2021 predicts that there will be 16 billion IoT devices, with a compounded annual growth rate of 23 percent.

In a smarthome, the use of multiple IoT devices can simplify daily life and common tasks. IoT devices can have personal configurations as well as access private and sensitive data.

An apartment, which may have IoT devices installed, can have multiple successive tenants. If some of the IoT devices are part of fixed infrastructure, i.e., they are not taken when the tenant moves out, then such devices may need to be reset. This avoids that the next tenant can access sensitive information of the previous one.

In business environments, IoT devices, e.g., smart smoke detectors, offer great possibilities to reduce costs and simplify business processes on a larger scale. A secure integration of such remote devices into a new environment or a reintegration after a network restructure or maintenance is an important and time-consuming task.

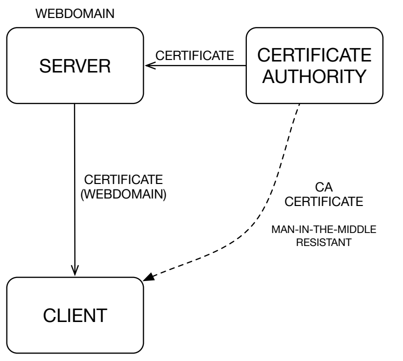
In order to securely integrate an IoT device into existing infrastructure, the device must first be authenticated.

In general, authentication is the process of proving if someone or something is, what it declares to be. For example, users can log into a system with their credentials to access a network.

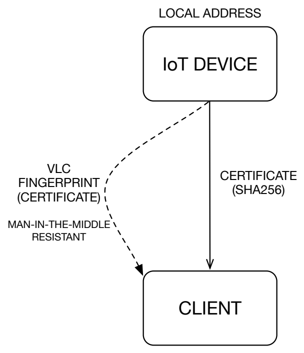
Figure 1.1a shows how authentication is done on the internet. A server provides its public key certificate to the client. If this certificate has been issued and signed by a trusted certificate authority (CA), the client's browser can establish the authenticity of the device.

The web browser has a built-in list of trusted CA certificates to check the validity of signatures and issuers of the received server certificates. The list of trusted CA certificates is installed into the web browser by its developers to have a channel resistant to MITM attacks. Also, a user can add trusted CA certificates to the list. Any web CA can serve as a trusted CA on a local network.

The problem is that most IoT devices have no name that can be verified by the client. For example, users do not refer to an IoT device by a domain name like "www.columbia.edu". A device could be named by their manufacturer's but other devices could still pretend to be the one by imitating this name. Thus, there is nothing for the certificate authority to verify. Common ways to authenticate such devices in a local network require physical access to the device.



(a) Secure internet authentication concept in which a CA proves the validity of the server's certificate to the client.



(b) Secure IoT authentication concept in which the visible light channel proves the validity of the IoT device's certificate.

Figure 1.1: Authentication concepts using a channel resistant to active MITM attacks

This thesis describes the design and implementation of a system to simplify the authentication of remote IoT devices by using visible lights communication as shown in Figure 1.1b. A companion report, Hagen Odenthal's Master thesis [24], describes the design and implementation of the transmitter. This thesis focuses on the receiver.

The following questions will guide this thesis:

1. How well does a smartphone serve as a VLC receiver?
2. What is the processing architecture of the receiver application?
3. How usable is a concept using VLC in terms of transmission speed and reliability?

As a first step this thesis defines the problem statement and based on that the requirements for this system. The background chapter summarizes concepts about necessary techniques and systems. Chapter 5 discusses the state of the art in authentication of IoT devices and prior research on visible light communication, pointing out why previous work does not achieve the goals this project requires.

The Chapter 6 gives general information about the entire system including transmitter and receiver. In Chapter 7, we describe the design and implementation of the receiver. Chapter 8 provides information about the accuracy and performance of the receiver application. In Chapter 9, we analyze the results of the experiments and the achieved goals. In Chapter 10, we consider the security aspects. Chapter 11 describes the limitations of this system and gives directions for future research.

2 Problem Description

The majority of IoT devices is designed for self-installation. It is often assumed by the manufacturer that the user who purchased the device will be setting up the device on their own. The increasing number of deployed IoT devices shows that there is a trend to equip more devices to the network. Since connected devices can bring many benefits, like simplifying daily tasks and being in control of the apartment remotely, we assume that the trend will continue.

But the concept of self installed IoT devices does not apply to future scenarios where a few of those devices are part of a fixed infrastructure. For example, such devices can be smart light-bulbs, smoke detectors, surveillance cameras and so on. Envision a scenario in which an apartment has successive tenants. The next tenant is confronted with three categories of devices. The first category consists of devices that the new tenant brings, including the access point, e.g., a router. The second category contains all devices that are already part of the fixed infrastructure of the apartment. The third category includes devices which do not belong to the apartment itself, but are commonly used by multiple tenants, like an intercom. The problem that the new tenant is facing, is to securely integrate all the devices into the new local network. To ensure that there is no MITM, authenticity of these is required.

Without authentication, an attacker could perform a MITM attack by setting up a device that pretends to be the IoT device that the user wants to connect to. When the user unintentionally connects to the attacker's device, this device can secretly forward all the data between the user and the IoT device. So, the user and the IoT device think there is a direct link, but the attacker is able to intercept and manipulate the traffic. For example, a MITM could hijack the video stream of a webcam in the apartment. A possible scenario could include watching the videostream without manipulation, e.g., spying on the tenant. Another scenario could either manipulate or interrupt the video stream. In this scenario, an attacker could break in the apartment while streaming a pre-recorded video sequence.

Especially the authentication process of remote or hidden IoT devices can be challenging since special equipment is needed to physically reach them or they are not accessible, e.g., built into a wall. To authenticate those devices, the new tenant could use a trusted third party, e.g., the landlord or a service company that installed the devices.

The state of the art in authentication of IoT devices requires proximity to the device. Since an apartment in the future, as we assume, can contain dozens of connected devices, the secure integration procedure will take a lot of effort. The tenant has the options to trust the third party or to authenticate all the devices using available approaches that require physical access. Especially a device that deals with sensitivity data, like an already installed camera under the ceiling, should definitely be authenticated without using a third party.

To enable a secure authentication of remote IoT devices, we propose a VLC based authentication system. This authentication system allows a user to securely authenticate all IoT devices in the line of sight without the need of a trusted third party or special equipment.

3 Concept Requirements

This chapter discusses the concept's requirements of a secure authentication system for remote IoT devices. Since a tenant who moves into an apartment with an already existing IoT infrastructure needs to do the authentication process for all IoT devices once or only a few times, the concept should not require special equipment that has to be purchased. For example, devices that the tenant already owns, should be able to serve as the receiver. The procedure of authentication should not require professional knowledge or experience, so it can be done by everyone who may be the next tenant. Therefore, the procedure has to be self-explanatory and very easy to use. The receiver application should be handheld and not rely on physical access to the IoT device. Since the tenant may not know the landlord very well, the tenant cannot be sure if the landlord can be trusted in terms of providing correct authentication data for remote IoT devices. That is why the concept of the authentication process should avoid the need of third parties.

The concept should prioritize security and easily allow to detect if a MITM tries to hijack the authentication process. There is no need of a high-bandwitdh VLC channel since this concept includes only the transmission of authentication data. The time of the authentication process should be less or equal to the time other authentication systems need for a remote IoT device authentication.

A user should be able to initiate the authentication process through a request.

The authentication concept should be based on already existing techniques that are considered to be secure. To ensure that previous tenants cannot access the IoT devices anymore, the concept should allow to dynamically generate new unique authentication information and not rely on static ones. Also, the use of dynamic generated authentication data does not allow an attacker to replace static information, e.g., printed QR codes or labels located on the device itself.

Since IoT devices can have various purposes and different designs, the required hardware components for the VLC authentication system should be low cost and should need very little space on the IoT device's form factor.

To sum up, these are the requirements for the secure authentication concept:

- no special equipment

- no need of professional knowledge or experience
- self-explanatory and easy to use
- receiver is handheld and remote to transmitter
- avoid the need of trusted third parties
- no need of a high-bandwidth VLC channel
- based on secure techniques against MITM attacks
- dynamic authentication data
- portable

4 Background

In order to reliably transfer data between a transmitter and a receiver, background knowledge about smartphone camera capabilities and color representations for visual computing purposes, is necessary. Furthermore, relevant data transmission topics are described in this chapter.

4.1 Visual Light Communication

Visual light communication (VLC) is described as an optical wireless technology where a receiver gathers information using a photodiode or a camera sensor directed to a transmitters light source, e.g., a LED.

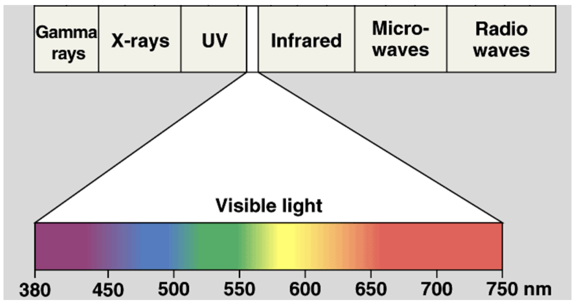


Figure 4.1: Visible Light Spectrum [10]

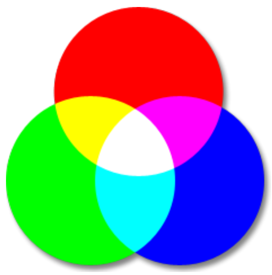
As shown in Figure 4.1, this is done on the visual channel (380 - 750 nm), which means that a human's eye can optically identify the transmitter's source.

4.1.1 Color Spaces

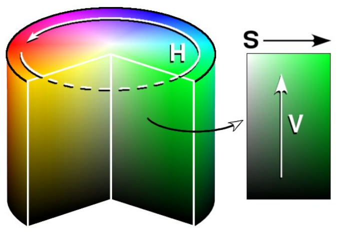
This section provides background on three of the most common color representations: Red Green Blue (RGB), Hue Saturation Value (HSV) and YUV.

The RGB color model is widely used in computer graphics. It is based on the physiology of the human eye which perceives color using three types of cone cells. Each cone type is sensitive to a different light wavelength: red, green, and blue. These are primary additive colors which means a mixture of these can be used to create a representation of a wide range of colors, also known as color space. Each primary channel is represented by one byte with a value from 0 to 255. The value represents intensity of the corresponding primary channel.

Image processing algorithms often need to separate luminance (brightness) information from chromaticity (color). The RGB model is not suitable for this purpose, because luminance is encoded in all three primary channels [26].



(a) RGB [28]



(b) HSV [15]

Figure 4.2: Color spaces

The HSV color model, which represents luminance and chromaticity separately, is more suitable for certain types of image processing tasks [26]. The RGB color model can be converted to a HSV model and vice versa. The hue component represents the closeness of the color to one of the primary colors. The relative lightness or darkness of the color does not influence the hue value.

Thus, for image processing, the hue value can be used to detect colors regardless of brightness or saturation. The hue range is designed like a circle with angular dimension representing smooth transitions between the primary colors, as shown in Figure 4.2b.

The saturation component represents the colorfulness, i.e., relative chromatic intensity. Together, hue and saturation are two attributes of chromaticity. The value component, also known as brightness, can shift the representation of a color between light and dark. If the brightness value is zero, the color is black regardless of hue and saturation components.

Another color model which is also based on a separation between the color and brightness components is YUV. Most Android smartphone cameras provide frame data encoded in the YUV420_888 format which uses 8 bits per channel. YUV420_888 is the default video format in the Android Camera2 Application Programming Interface (API) [5]. Using this model, a color is represented by a triple consisting of Y, U, and V. Y is the luminance component and serves as a container for the brightness of a pixel. U and V represent the chrominance components and include color information of the corresponding pixel, whereas U contains blue minus the brightness and V red minus the brightness. The YUV representation, originally designed for terrestrial TV broadcasting, was designed to provide backward compatibility with black-and-white TV sets. By transmitting the Y, U and V channels separately, a black-and-white TV set could only process the Y channel, while a color TV set could process all channels. This type of color representation enables a way to reduce memory storage. Because of the fact that human's eyes are more sensitive to luma than to chrominance, chroma subsampling can be used to sample color at a lower resolution than the brightness [17].

For example, YUV420 is used in the JPEG standard. The first digit gives information about how wide the reference block of the sampling pattern is in pixels. The second and third digits define the subsampling across row and columns, e.g., there are 8 pixels with each different luma data, but there are just two samples of color information which are taken from the top row. The third digit defines the color information gathered by the bottom row. Since the digit is zero, the pixels in the bottom share the same color information like the pixels in the top row [27].

4.1.2 Camera Model

In order to design and implement image processing features in a smartphone application, we have to understand how smartphone cameras take photos and videos, including how they operate and what camera characteristics are important.

The most common type of sensor in built-in smartphone cameras is the CMOS image sensor. CMOS sensors use the rolling shutter method to expose pixels to light. A sensor with a rolling shutter exposes each row of pixels in the frame sequentially [21]. An important side effect of exposing different rows at different times is distortion.

Android is designed to support a variety of types of smartphones and cameras. Since it needs to deal with hardware and feature variability, the Camera2 API is complex. The API provides means to discover the features supported by the camera [6]. Important characteristics are sampling rate (frames per second), exposure time and frame interval. By capturing a video, consisting of a sequence of images, the frame duration d is set to a fixed interval, determined by the camera's sampling rate setting.

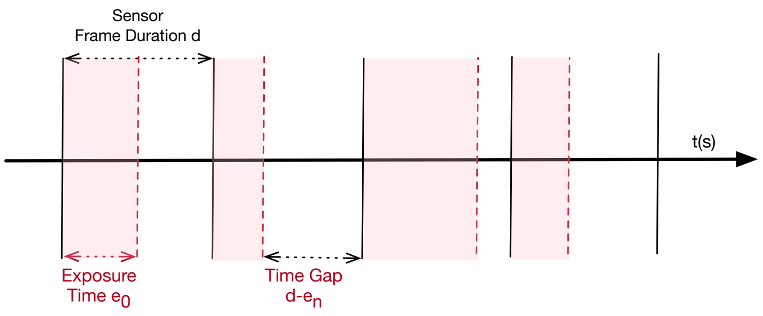


Figure 4.3: Android Camera2 capturing frames

As shown in Figure 4.3, the exposure time e_n can have different durations caused by the time the camera sensor needs be exposed to light to capture a required amount of light. This effect can be significantly enhanced by capturing frames in moving or changing environments when the camera is set to automatic exposure, which it is by default. Consequently there is a time gap $d - e_n$ which can have different durations. Depending on the frame rate a camera is set to and the resulting exposure time durations in various environments, the exposure time can overlap the next frame interval and lead to dropped frames. A way to control or adjust the exposure time is to set it manually. Using auto exposure compensation allows to adjust the automatically determined exposure time slightly, but does not solve the problem.

Automatic White Balance (AWB) is another important characteristic that is used to emulate the eye's perception of colors in different lightning conditions. Therefore, Camera2 provides a set of different color correction algorithms [4]. This can lead to different representations of the same color and has to be kept in mind during image processing.

Since camera sensors are sensitive to light intensity and not its frequency, a red, green or blue filter is placed in front of each pixel and arranged in a pattern. The most common pattern is the Bayer mosaic which contains twice as many green filters as red or blue ones, as shown in Figure 4.4.

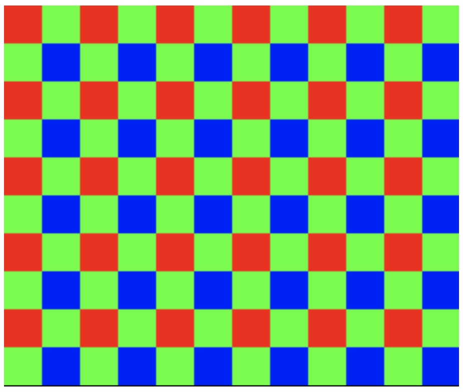


Figure 4.4: Square Bayer filter on a pixel array

Because there are more pixels with green filters, there is more information coming from the green portion of the spectrum when computing RGB. That is why chroma subsampling has to be done and the camera is using the YUV color space [9].

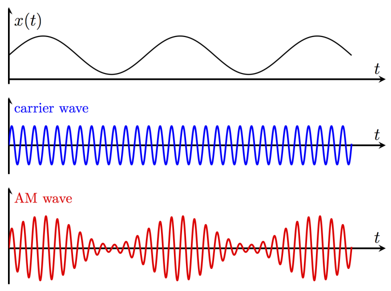
4.2 Data Transmission

This section provides background about transmission related topics like modulation schemes and forward error correction.

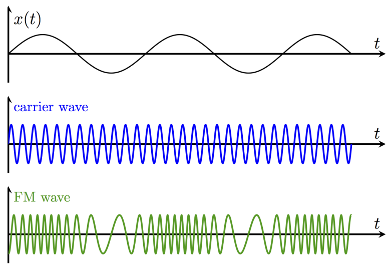
4.2.1 Modulation

Modulation is a technique that transmits a message signal to the receiver by varying the periodic waveform of a carrier signal. The resulting modulated signal can then be transmitted. For VLC we are using continuous wave modulation which consists of amplitude and angular modulation [23].

Amplitude modulation modifies the amplitude of the carrier signal according to message signal as shown in Figure 4.5a. Amplitude modulation can be used to vary the intensity of the LED. A simple form of amplitude modulation is On-Off keying which turns the LED on or OFF. For higher bandwidth, the amplitude range of the LED's brightness can be divided into multiple intensities. In contrast to On-Off keying in which the transmitted symbol either is the bit "0" or "1", multiple intensities enable the use of longer bit representations for each symbol.



(a) Amplitude modulation modifies the amplitude of the carrier wave (middle) according to the message signal (top). [2]



(b) Frequency modulation modifies the frequency of the carrier wave (middle) according to the message signal (top). [12]

Figure 4.5: Modulation techniques amplitude and frequency modulation

A relevant form of angular modulation in our project is frequency modulation. As shown in Figure 4.5b, this modulation technique modifies the frequency of the carrier wave according to the message signal [23]. Using a constant amplitude (intensity) of the LED, the frequency modulation scheme called frequency shift keying (FSK) can be used to transmit different colors as shown in Figure 4.1. Each color represents a different symbol. The number of separations in the color spectrum determines the number of symbols. For example, FSK8 divides the color spectrum into eight different colors and each color represents a symbol consisting of 3 bits ($2^3 = 8$). Frequency modulation requires a tri-color LED.

The more colors are used, the faster is the transmission. The same goes for the number of different intensities in the amplitude modulation. A mix of both modulation techniques can improve the transmission speed.

4.2.2 Forward Error Correction

In noisy or unreliable channels, the probability of errors like packet loss or corruptions can be significantly higher than in other channels. Forward Error Correction (FEC) codes can be used to deal with such errors in a data transmission. The concept is to encode data in a redundant way. So, the receiver can obtain the data including a limited number of errors without the need of a reverse channel to request a re-transmission. Packets that are passed to FEC should always be checked before, e.g., using Cyclic Redundancy Check (CRC).

A widely used FEC technology is RaptorQ. RaptorQ is a fountain code that is able to efficiently generate a potentially unlimited number of packets from a fixed size data set [18]. This allows the receiver to reconstruct the origin data when packets are lost or corrupted. Encoding and decoding is done in linear time. The encoder block can generate these packets on the fly and the decoder stores all incoming packets as long as a specific number of packets have been received. There are two types of packets. The first type is a source packet which consists of a portion of the initial data set. The second type is a repair packet which includes parity bits of source packets. To decode packets correctly with a high probability, the decoder needs to receive at least the same number of packets that the encoder generated in form of source packets. For this, it does not matter if the received packets are source or repair packets. In case the receiver received all source packets without any error, the decoder is always able to decode the data. The number of received repair packets defines the probability to successfully decode the origin data.

According to Qualcomm's RaptorQ Technical Overview [18], the decoding of data that is encoded to k source packets, can be done with a probability of 99 % for k received packets, 99,99 % for $k + 1$ received packets, and 99,9999 % for $k + 2$ received packets. These probabilities apply for transmission data with various numbers and sizes of source packets. Also, the number of packets losses does not affect the probability.

Each RaptorQ packet consists of a source block number, a packet number and the payload. So, the first two parts of the packet inform the decoder how to construct the origin data.

5 Related Work

The first section of this chapter provides information about different approaches of using Out-of-band (OOB) channels to authenticate IoT devices via a channel resistant to active MITM attacks.

The second section describes previous research on VLC. Furthermore, it demonstrates why previous approaches do not achieve the objectives of this project.

5.1 Authentication of IoT Devices

The literature describes many types of secure OOB channels with varying levels of security [20]. Different types of channels have different properties, e.g., maximum bandwidth. These channels must be resistant to active MITM attacks and are used to securely transmit a shared secret, which either contains a low entropy password for Password Authenticated Key Exchange (PAKE) [8], or a fingerprint of a Diffie-Hellman public key. In general, this additional channel is used to authenticate a key exchange which has been performed over the insecure primary communication channel.

A simple way to authenticate devices is the approach described by Soriente [29] which relies on buttons which have to be physically pushed in a predefined interval. Visual authentication methods typically rely on printed bar codes or QR codes which can be located on the device itself [22]. An audio channel approach was introduced by Soriente, Tsudik and Uzun [30], exchanging data and verification information through microphones and speakers. Another authentication method relies on physical contact of the devices. By the same movement through shaking both of the devices, a shared secret can be generated by the built-in motion sensors. The last approach to mention is a proximity based one. The IoT device is continuously sending data to the smartphone using Wi-Fi. Then, the user is instructed to perform a specific gesture with the smartphone. By matching the received signal strengths in the Wi-Fi channel and the smartphone's sensor data during this gesture, the IoT device can be authenticated [33]. Since the motion has to be performed with the smartphone instructed by the IoT device, this may have to be done very close to the IoT device itself.

There is also a security standard for IEEE 802.11 networks to simplify the network setup procedure, e.g., the initial configuration of Wi-Fi clients, called Wi-Fi Protected Setup (WPS) [1]. Using WPS, a user does not need to manually add the credentials of the Wi-Fi network to every joining device. WPS uses a cryptographic protocol to establish a secure authenticated channel between the joining device and the access point. Therefore, WPS requires a shared secret. For example, the shared secret can be a dynamically generated PIN shown on the device's user interface (UI) or a static PIN printed on a label on the back of the device [31]. WPS also supports other MITM-resistant channels such as NFC tags or USB flash drives.

Since many devices do not have a UI to show a dynamically generated PIN, they often rely on a static PIN. The use of static PINs makes WPS vulnerable to brute force attacks [31] even though it relies on reasonably secure cryptographic protocols.

Moreover, the static PIN method is required for all WPS-compatible devices which makes the architecture of WPS insecure by default. Furthermore, authentication through WPS requires physical access to the device which does not meet the requirements of our project.

According to the concept's requirements in 3, most of the existing authentication methods require either close proximity or a trusted third party to mutually authenticate the two devices.

5.2 Prior Research on Visual Light Communication

This section provides information about research regarding visual light communication. A big challenge in visual light communication between a LED and a smartphone generally is the receiver diversity. The built-in camera sensors in smartphones can vary significantly in their characteristics which can affect color detection algorithms. Also, the frame rates can be different and even fluctuate which can make the synchronization more complex [21, 16].

Previous research mainly focuses on high bandwidth applications by taking advantage of the rolling shutter effect. The authors of [21] experienced a data rate of 12 bytes per second and Hu et al. (2015) increased the transmission rate up to 5200 bytes per second using Android operated smartphones [16]. Like described in the concept requirements in 3, the aim of our project is to establish a (slow speed) reliable communication channel to transmit a fixed amount of data for authentication purposes. Rather than using the rolling shutter effect, our project focuses on supporting a wider range of devices, with different camera characteristics, such as the maximum frame rate.

As well as the ColorBar project [16], the project DisCo [19], which introduced a way to transmit data from screens to rolling shutter sensors by temporally modulating the display brightness at high frequencies, transmits data in such a way that it is imperceptible for humans. The requirement that the communication is not visible to the human eye is another feature our project does not need. In fact, the transmission has to be visible so that one can see where the transmitter is located and whether or not it is transmitting. Another interesting difference in the ColorBar project [16] is that it requires at least 10 pixels per band which could mean that the camera needs to be focused on light which covers a large fraction of the image area. Since in our project the device is handheld and does not require proximity, the light may only cover a small part of the image.

In a different way than the authors of [32] and [16] experienced low transmission distances between centimeters and one meter, our project needs a greater distance up to a few meters to enable data transmission between remote devices.

Referring to the concept requirements in 3, previous related work does not achieve the needed objectives.

6 System Overview

This chapter provides information about the overall system architecture of the project. Firsts, it outlines the operation and then the necessary components for both the transmitter and receiver.

6.1 Sketch of Operation

For example, a smart smoke detector is equipped with a tri-color LED. A smartphone serves as a receiver and has a built-in camera.

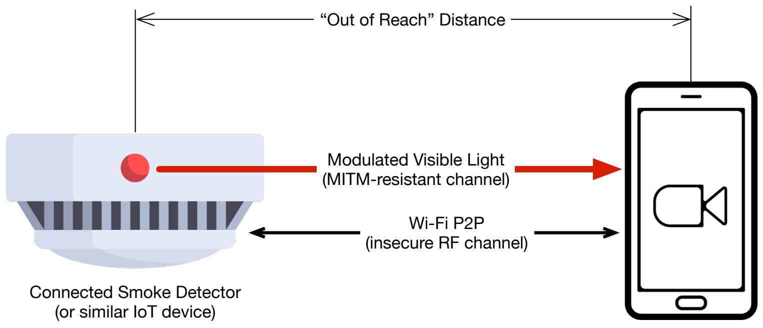


Figure 6.1: A possible authentication scenario consists of a smartphone that authenticates a remote remote smoke detector using VLC.

In Figure 6.1 the smoke detector is not easily reachable, however the device is within the line of sight. Based on this, the device cannot be authenticated without knowing a shared secret or doing push button authentication. Often, printed labels or QR codes located on device's back serve as a shared secret.

Using a Wi-Fi P2P connection from the smartphone to the smoke detector, which is accepted automatically by any device which is not part of a Wi-Fi infrastructure, the smoke detector sends its certificate to the smartphone in order to do a transport layer security (TLS) handshake. After the handshake has completed, the connection is encrypted but not authenticated. If the certificate has not been signed by a CA that the smartphone trusts, e.g., if the certificate is self-signed, the smartphone must use a MITM-resistant channel to authenticate the certificate.

The smartphone requests the fingerprint of the certificate to be sent on the visible light channel. By pointing the camera at the smoke detector, we ensure there is a direct link for transmission. After receiving the fingerprint matching the TLS certificate's fingerprint, the TLS connection is authenticated.

6.2 System Architecture

This section describes the general overview of both the transmitting and receiving parts of the project.

As shown in Figure 6.2, the transmitter consists of several components. Using a FEC scheme, incoming data is encoded to FEC packets. The use of FEC allows this project not to rely on a reverse channel to request retransmission of lost or corrupted packets. After attaching the checksum, which is calculated over the sequence number and the payload, a line coder encodes it, so that the receiver can detect the beginning and the end of a packet. In addition to that the data of the packet is modulated and the LED driver is in charge of translating the modulated signals into visible light by using pulse width modulation to control the intensities of the individual channels of the connected tri-color LED.

Hagen Odenthal's master thesis [24] describes the design and implementation of the transmitter components in more detail.

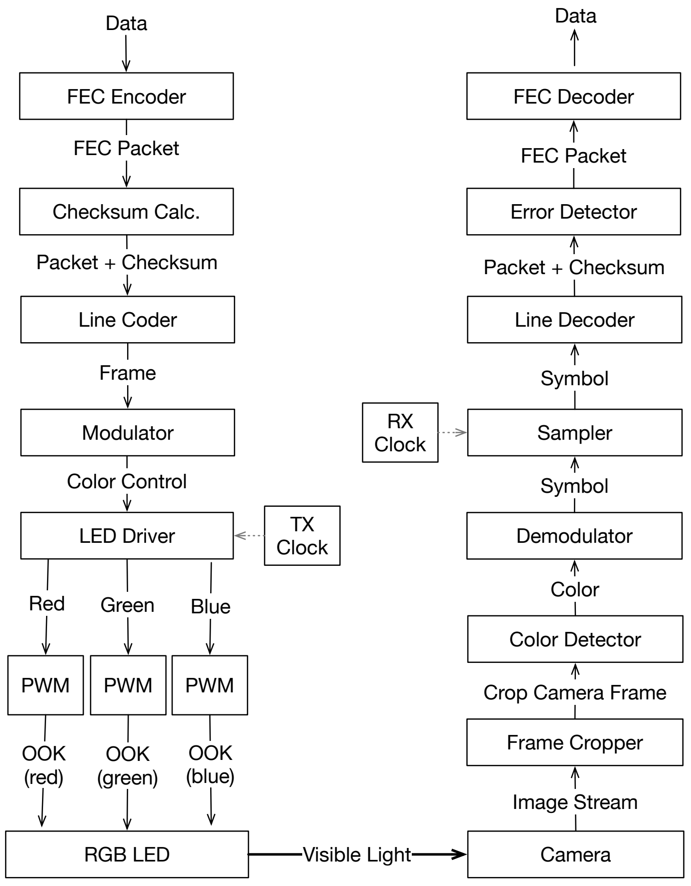


Figure 6.2: The overall system architecture contains components of transmitter (left) and receiver (right) to transmit data on the visible light channel.

The receiving part captures the incoming visible light through a camera sensor. The resulting image stream then is cropped to a smaller size camera frame. Using computer vision algorithms, the color detector component detects the color. Then, the demodulator converts the color to a symbol and the sampler forwards only the ones which are matching the predefined frame rate. The line coder component then collects the data for each packet which the error detector checks. Finally, the FEC decoder component decodes the packets into data.

In order to establish a reliable visible light transmission, the transmitter and receiver have to set common characteristics. Therefore, they use an API to define the modulation scheme, transmission speed and timeout.

7 Receiver Design and Implementation

The transmitter and receiver rely on a shared library. The shared library provides building blocks used by both the transmitter and the receiver.

The Android application uses Android Camera2 API for image receiving. This API offers various image receiving modes for different purposes. There are two modes which focus on frame rates rather than image quality. The first mode TEMPLATE_PREVIEW prioritizes high frame rates and the other mode TEMPLATE_RECORD should deliver images at a stable frame rate [3]. We considered and experimented with both modes and there was no noticeable benefit of using the mode TEMPLATE_RECORD with respect to frame rate. This is why we use the mode TEMPLATE_PREVIEW for image receiving.

For image processing we use OpenCV [25], a free computer vision library.

The following sections describe the design of the receiver application in detail.

7.1 Concurrency Model

The receiver uses a multi-threaded application architecture to move the workload from the main (UI) thread to background worker threads. That way, the UI thread will not be overloaded and it is possible to split the processing workload across multiple CPU cores. This design makes it possible to perform frame processing and camera sampling at a relatively high frame rate.

The application uses a high priority thread to receive frame data from Android's Camera2 API. We refer to the tasks performed on the image receiving thread as "stage 1". Stage 1 can only include tasks that run in constant time with respect to the size of the frame. This makes sure that the image receiving thread is not overloaded. Otherwise, if this thread was too busy, Camera2 API would drop incoming frames.

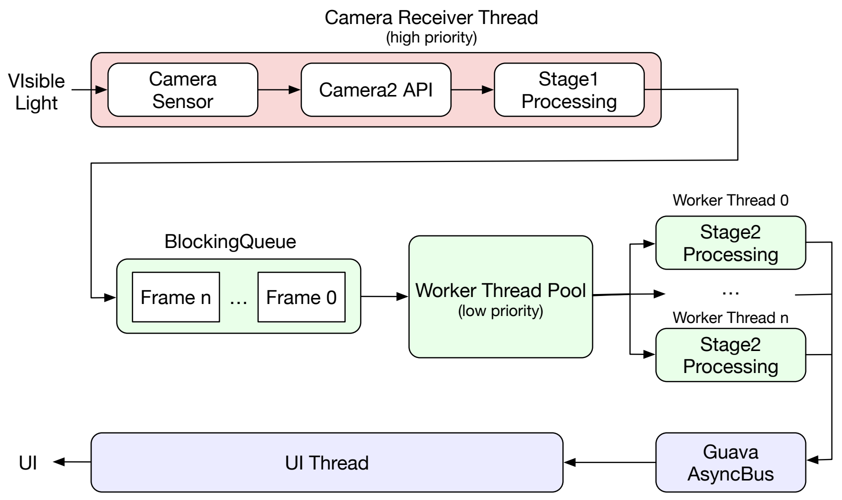


Figure 7.1: The multi-threaded setup consists of time-sensitive tasks running on the high priority camera receiver thread, time consuming tasks running on low priority background worker threads, and UI drawing tasks performed on the UI thread.

After a frame passes stage 1 image processing, it is stored in a blocking queue and waits to be delivered to "stage 2" processing. Stage 2 processing includes more time consuming tasks and is performed on a background worker thread. Since the frames are already stored in the blocking queue, these threads are not that time sensitive and can be configured with low priority.

Figure 7.1 shows a worker thread pool which can contain multiple worker threads depending on the number of available CPU cores. Whenever a worker thread becomes idle, it checks the blocking queue in order to pick up a frame to perform stage 2 processing. If the blocking queue is empty, it goes to sleep until it gets notified.

In case the processing on worker threads takes too much time, the blocking queue grows bigger and frames are processed with an increasing delay. Eventually, when the delay grows too large, there is a noticeable lag between the time the frame is displayed in the UI and the time it is processed. This is only a problem if, e.g., the area of interest is too big.

To display feedback and visualizations from image processing to the user, data has to be passed to the UI thread which drives the UI event loop. To prevent the Android activity manager and window manager system services from shutting down the application, one has to ensure the application's responsiveness at all times. Consequently, any time or resource consuming task is performed in a background worker thread. Worker threads communicate the results to the UI via an event bus implemented using the Guava library [14].

7.2 Receiving Camera Frames

Incoming visible light is captured by the smartphone's camera sensor. As mentioned in Chapter 4, Android uses the YUV420_888 format to encode sensor data by default. The luminance and chrominance channel data is stored together with the timestamp of the frame. Figure 7.2 shows a Rate Monitor block which estimates an average camera sampling rate over a short period of time using the Exponential Weighted Moving Average (EWMA) method.

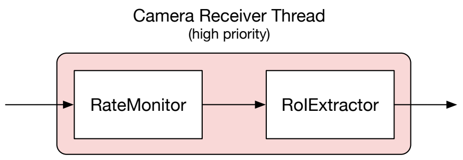


Figure 7.2: Stage 1 image processing is running on the high priority camera receiver thread and includes low cost operations like FPS calculation and image cropping.

The next receiving block, named RoIExtractor, crops the image to the region of interest, i.e., a small rectangular area in the center of the image which a user can specify in the application's UI.

In addition to that, the data is being passed to the blocking queue of the worker pool for further image processing purposes in stage 2.

7.3 Processing Camera Frames

The worker thread in Figure 7.3 applies four consecutive stage 2 processing blocks. This section describes each processing block individually.

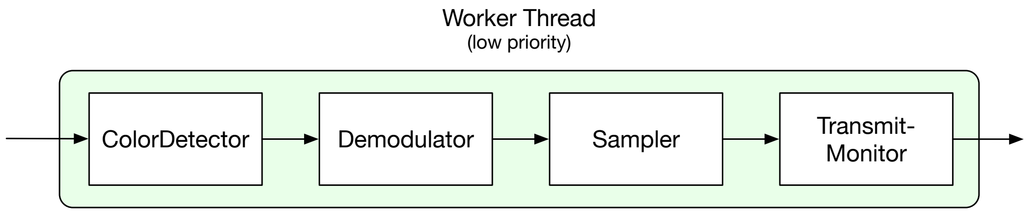


Figure 7.3: The stage 2 image processing tasks color detection, demodulation, sampling, and monitoring run on a low priority background thread.

7.3.1 Color Detection

The first processing block implements color detection. A HSV color model is very suitable for color detection purposes since it represents luminance and chromaticity separately. The relative lightness or darkness of a color does not affect the hue value which represents the color itself. Besides that, very bright or dark colors can be eliminated.

In Android incoming frames are encoded in YUV420_888 format. In order to represent the colors of a frame in HSV color model, a color model conversion to HSV is performed. Using OpenCV this can be done very efficiently. The color detection block iterates over all pixels in the buffer and eliminates very bright or dark pixels. This can be determined by predefined thresholds for saturation and value (brightness) components. Thus, the average hue calculation includes only the remaining pixels. Since hue is a color representation with angular dimension from 0 to 360, the calculation of the average is more complex, i.e., the average of two reddish colors with hue components 355 and 5 is not 180. Therefore, we use the following formula [7]:

$$\overline{\alpha} = atan2\left(\frac{1}{n}\sum\nolimits_{j=1}^{n} \sin\alpha_j, \frac{1}{n}\sum\nolimits_{j=1}^{n} \cos\alpha_j\right)$$

, where n describes the size of the corresponding pixels and α_j describes the angle at position j.

7.3.2 Demodulation

The demodulation processing block converts the average hue computed in the previous step into a symbol value. The modulation scheme FSK4 uses four different color values to encode four different symbols.[1] We pick FSK4 as the modulation scheme because that allows us to map three symbol values to primary colors and one symbol value to black. Also, this mapping is reliable, easy to detect, and simple to implement.

[1]In this project, we transmit one symbol as "black" (the LED is off) to simplify the implementation.

Table 7.1: Modulation Scheme FSK4

Symbol	State	Hue	Saturation	Brightness
00	off	0	0	0
01	red	0	100	100
10	green	120	100	100
11	blue	240	100	100

We use the nearest neighbor algorithm to match the calculated average hue with hues expected by the FSK4 demodulator. In detail, the algorithm calculates the distance of the detected color from the four colors used by the FSK4 demodulator. It then picks the FSK4 modulator's color which is nearest (in Euclidian space) to the detected color.

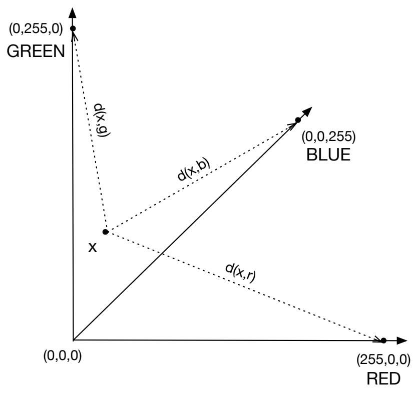


Figure 7.4: The Euclidean distance is calculated in RGB color space between primary colors and a detected color x.

In Figure 7.4, the distances between the received color x and the colors r, g, b are calculated using euclidean distance. Doing this in the RGB color space ensures that other modulation schemes, which may contain colors which are defined by brightness, e.g., white, can still rely on the implemented functions.

7.3.3 Sampling

Both the transmitter and receiver are set to a common interval $t = r$. The frame sampler block selects frames in the common interval for further processing. Other frames are dropped.

In Figure 7.5, the camera delivers frames in interval c. The camera interval c must be smaller than r to ensure that there is at least one frame sample per each interval. Since the camera sample rate can fluctuate, this interval can have slightly different durations during the transmission. Moreover, in Camera2 API is no option to set request including a timestamp for the camera to take a frame sample. By default, a receiver in this project runs at half the sample rate the camera is capable of to avoid fluctuation related issues.

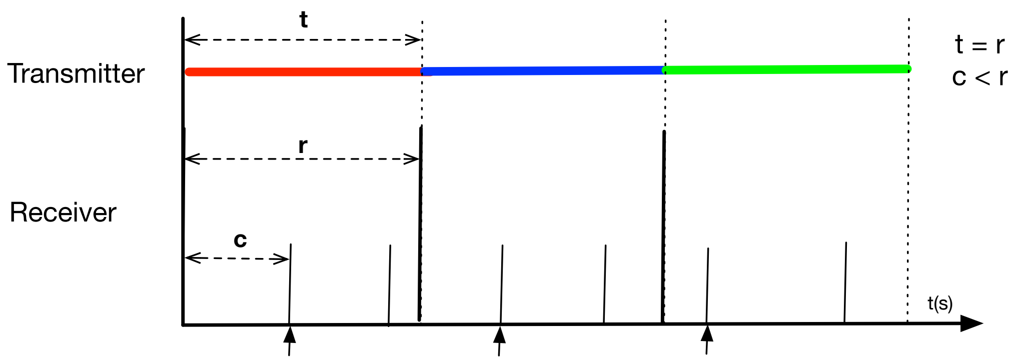


Figure 7.5: The receiver captures frames in an aligned interval, where the transmitter and receiver started exactly at the same point of time.

Figure 7.5 illustrates an ideal case in which the receiver and transmitter started at the exact same point of time. The frame sampler forwards the very first camera sample in each receiver interval to the next processing block. Therefore, it checks timestamps of each incoming frame and compares it to the one determined by the receiving interval r.

In practice, the transmitter and receiver often do not start at the exact same point of time. Consequently, there is a shift between the transmitter and receiver interval. As long as the shift is not too great, the frame sampler provides frames that represent the transmitters symbols correctly. Figure 7.6 shows a scenario that leads to missing frames.

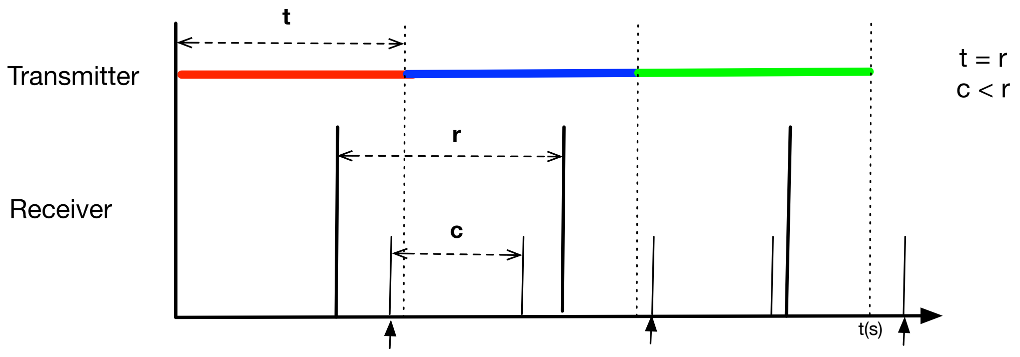


Figure 7.6: The receiver captures frames in a shifted interval which leads to missing frames.

The start of the receiver r interval is shifted close to the end of the transmitter interval t. In addition, the camera interval c fluctuates. Consequently, a camera sample for transmitter symbol can be missed, as shown for the blue one in Figure 7.6.

To avoid this situation, the frame sampler can take advantage of a synchronization approach which is described in the section future work.

7.4 Data Decoding Architecture

This section describes necessary decoding blocks which have to be applied in order to retrieve the data. These blocks are executed in the background worker threads.

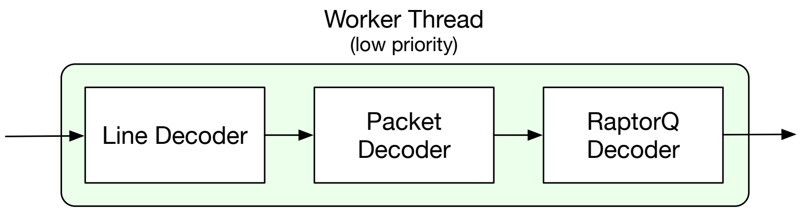


Figure 7.7: The stage 2 image processing tasks line decoding, packet decoding, and RaptorQ decoding are performed on a low priority background thread.

7.4.1 Line Decoder

In order to ensure that the receiver can detect the start and the end of a packet, we use a line coder. The transmitter prepends a starting sequence to the packet which then is called datagram. Without this, the receiver would not know where a packet starts and ends, e.g., if the receiver starts receiving in the middle of a transmission. A datagram is the data being transmitted and each datagram consists of a starting sequence and a packet as shown in Figure 7.8.

The line decoder block receives symbols from the frame sampler in a defined interval and arranges those symbols into datagrams.

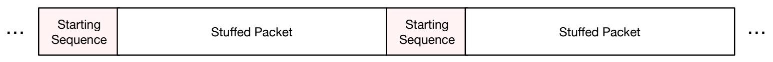


Figure 7.8: Multiple datagrams are constantly sent over the visible light channel during the transmission.

We use the symbol stuffing technique to make sure that the starting technique does not appear in the middle of a datagram. Otherwise, the datagram would end early, be marked as corrupted, and dropped in the next processing block. The state machine is designed to detect and remove starting sequences. It also removes the symbols inserted by the transmitter during symbol stuffing if necessary, as shown in Figure 7.9. For better understanding, symbol stuffing and the corresponding state machine is described in detail in the setup and implementation of the transmitter part [24].

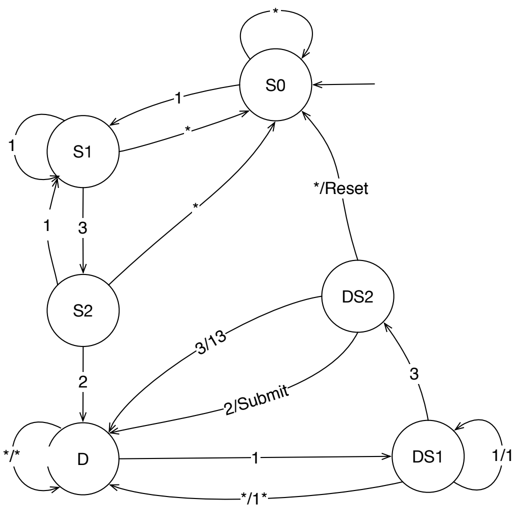


Figure 7.9: The line decoder state machine detects and removes the starting sequence symbols "1,3,2". Also, it erases stuffed symbols.

The line decoder state machine consists of six states, whereas state S0 is the starting state. Together with the states S1 and S2, these states detect the very first occurrence of the starting sequence "1,3,2". As soon as the first starting sequence is detected the state machine is in state D. As long as there is no symbol "1" coming in, the state machine adds any incoming symbol to the buffer. An incoming symbol "1" fires a transition to state DS1. In this state, it stores another incoming symbol "1" to the buffer and does not imply a change of state. An incoming symbol "3" changes the state to DS2, whereas all other incoming symbols change the state back to D and store symbol "1" and also the latest symbol. Since in state DS2 the two last incoming symbols "1,3" are close to the starting sequence, another incoming symbol "3" indicates a stuffed sequence and is removed by storing the symbols "1,3" and a change to state D. An incoming symbol "2" declares a starting sequence and submits the datagram buffer to the next processing block including a state change to D. Any other incoming symbol in state DS2 signals error during receiving and resets the state machine.

7.4.2 Packet Decoder

The packet decoding block separates the data of the packet into three parts, as shown in Figure 7.10. The first part contains the CRC8 checksum and is calculated over the payload and the sequence number. The sequence number is located in the tail, the payload is placed between the sequence number and the checksum. The sequence number field is at the end of the packet to enable variable sequence number encoding. Since the payload of the packet and the checksum have fixed sizes, the transmission of the sequence number is designed more efficiently. The sequence number originally has 8 bits in size, but in most cases the sequence number is low because the data is split into a small number of packets. By erasing the leading zeros, fewer bits have to be transmitted, making the transmission faster.

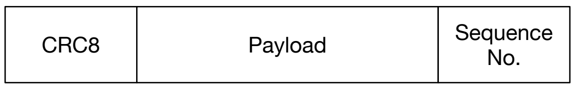


Figure 7.10: Packet content

To ensure the packet is received correctly, the CRC8 checksum is recalculated and matched to the one stored in the first part of the packet. If the checksum matches, the remaining data which is the payload and the sequence number is a FEC packet and is forwarded to the next decoder block. Otherwise, the packet is dropped.

7.4.3 RaptorQ Decoder

The last decoding block is named RaptorQ Decoder. Since the visible light channel can be very challenging, RaptorQ as a form of FEC is used to retransmit lost or corrupted data due to noise and the camera not being pointed at the transmitter correctly. RaptorQ is a fountain code which means it can generate a very large number of FEC packets, consisting of source and correction packets, for a fixed set of data. RaptorQ can recover from data loss by only accumulating n or more FEC packets, where n is the number of source symbols the source data was split into. Moreover, it does not matter whether the accumulated FEC packets are source or correction data.

The RaptorQ decoder can only decode the data correctly if all FEC packets it receives are correct. To do so, corrupted packets must never be forwarded to the decoder. Consequently, the CRC8 error detection in the packet decoder block drops corrupted packets. In other words, the error detection layer turns corrupted packets into lost packets.

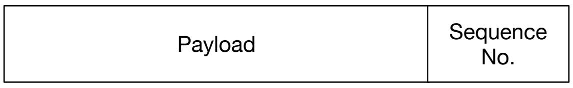


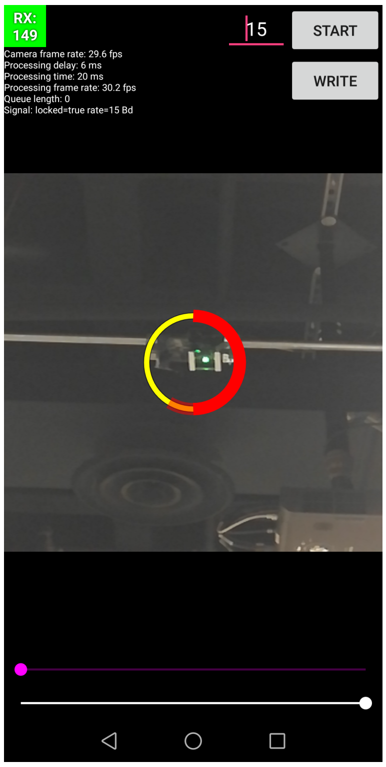
Figure 7.11: FEC packet

The source data is converted into a limitless sequence of fixed-sized FEC packets, each of which carries a portion of the source data, or parity (correction) data.

7.5 User Interface

In order to provide feedback to the user, the receiving application uses the Guava AsyncBus [14] to forward information about the current transmission progress to the UI thread and then draws it on the UI.

Figure 7.12 shows the UI elements which control the receiving application. In the center of the screen, a circle represents the area of interest within the picture. A user can resize the circle by a pinching gesture on the screen. Furthermore, this circle serves as an indicator if the desired frame rate is reached. When it is, the circle turns yellow, otherwise it is white. So a user can be sure the camera is pointed correctly. When the transmission is in progress, the circle is partially covered with a red overlay. As soon as the entire circle is covered, the transmission has completed.
There are two sliders in the bottom of the screen. The top one can be used to applies auto exposure compensation; the other one changes the digital zoom. In the top right corner, a button starts the visual light transmission.



```
Camera frame rate: 29.6 fps
Processing delay: 6 ms
Processing time: 20 ms
Processing frame rate: 30.2 fps
Queue length: 0
Signal: locked=true rate=15 Bd
```

Figure 7.12: The red overlay that covers the circle in the center of the receiver UI indicates the progress of the transmission. The sliders in the bottom adjust the camera.

8 Evaluation

8.1 Hardware Setup

This section describes the setup, including hardware and operating system specifications.

This project uses two different transmitters, listed in table 8.1. Both of them include a tri-color RGB LED which is connected to three GPIO pins and consists of three independent monochromatic LEDs (red, green, blue) in a diffused package. The maximum current flowing through each LED is limited to about 20 mA.

Table 8.1: Transmitter being used in the experiments

ID	Device	OS	Kernel	LED
1	Raspberry Pi 3 Model B, revision 1.2	Raspbian (Debian 9.4)	Linux 4.14.52-v7+	Cathode, RGB, 5mm
2	Raspberry Pi 3 Model B, revision 1.2	Raspbian (Debian 9.4)	Linux 4.14.50-v7+	Anode, RGB, 10mm

On the receiver side, the experiments use different Android based smartphones:

Table 8.2: Receiver being used in the experiments

ID	Device	Android version	Max fps
1	Huawei Honor 7X	7.0	30
2	Samsung Galaxy S7	8.0	30
3	OnePlus 3T	8.0	60

8.2 Laboratory Experiment I - Color Detection

The experiments are performed on different transmitters and different receivers to generate comparable results.

We modified the implementation to test the accuracy of the visible light channel. The transmitter iterates over a sequence of hue values. Each hue is transmitted for one second. The smartphone application records received hue values together with timestamps and camera-related characteristics in a local file for later analysis.

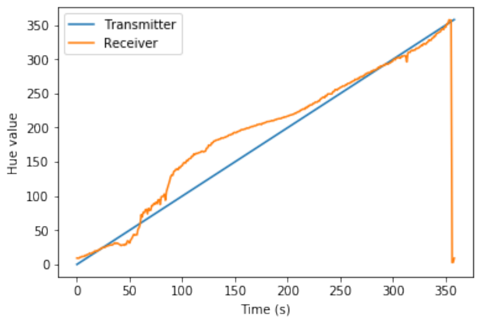
We run the color detection experiment over two distances: 2 meters (short) and 4 meters (long). In all cases, the transmitter was fixed under the ceiling and the receiver (smartphone) was mounted on a tripod.

8.2.1 Auto White Balancing (AWB) Experiment

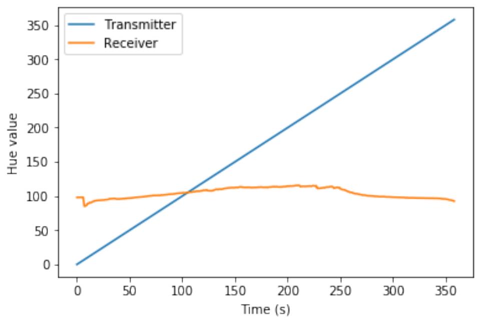
The AWB experiment uses Samsung Galaxy S7 as a receiver and the transmitter 1 at short distance (2 meters).

Generally, the noticeable change in between the hue values 359 and 0 in the 0-10 time range and in the 350-369 time range does not indicate inaccurate color detection. The problem here is that as the hue value wraps from 0 to 359 or from 359 to 0, the small change has a disproportionally large effect on the graph.

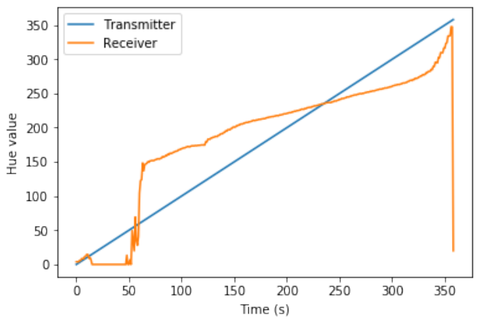
Since AWB manipulates the colors to emulate the eye's perceptions under different lightning conditions, there is a set of different modes available. If white balancing is set to automatic, these modes are switched automatically regarding to changing conditions. This experiment describes the incandescent mode and the cloudy daylight mode because we believe that these modes are on the opposite ends of a hypothetical warm-cold AWB spectrum. Besides that, this experiment evaluates the color detection when the AWB mode is on and off.



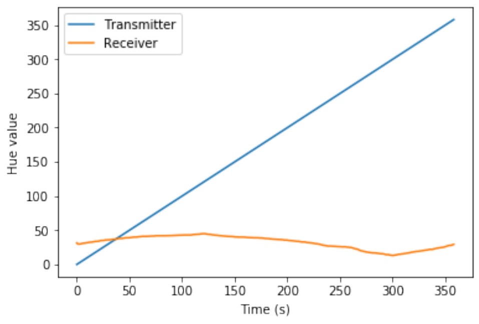
(a) AWB on



(b) AWB off



(c) incandescent mode



(d) cloudy daylight mode

Figure 8.1: The white balancing experiments shows that the Samsung Galaxy S7 detects colors more reliable when AWB setting is on than using AWB mode off, incandescent mode or cloudy daylight mode.

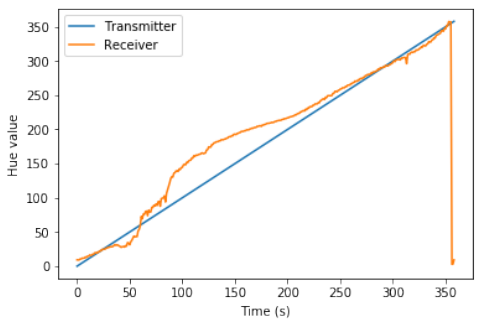
Figure 8.1 shows Samsung Galaxy S7 detecting colors with different white balance settings. Using AWB in Figure 8.1a, the color detection exhibits deviations across the entire range. With the AWB setting disabled, the phone detects only greenish colors across the entire hue range. According to Camera2 API, color correction matrices have to be defined to control white balancing manually when the AWB setting is off [4]. When the AWB mode is set to incandescent mode in Figure 8.1c, the smartphone does not detect yellow colors. So, it is not suitable. Figure 8.1d shows the cloudy daylight mode, but the Samsung Galaxy S7 only detects reddish colors across the entire hue range.

Based on this experiment, the smartphone detects the colors most reliably when AWB setting is on. This project relies on detecting hue values as accurately as possible and does not care about what the temperature of the color is like. In some environments, specific white balancing modes may lead to a better color perception for humans, but this is not the purpose of this project. This project uses AWB.

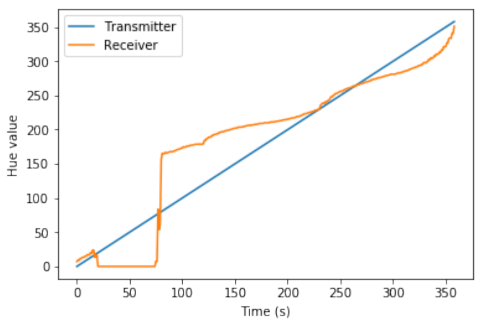
8.2.2 Distance Experiment

This section describes short distance (2 meters) and long distance (4 meters) experiments. We use the smartphones Samsung Galaxy S7, Honor 7X, and OnePlus 3T as receiver and both transmitters.

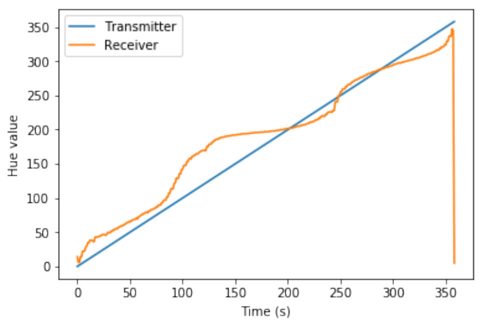
In the graphs using transmitter 1 in figures 8.2a, 8.2c, and 8.2e there is a significant offset that all smartphones have in common in the hue range from roughly 80 to 180. This hue range is known for having a green portion in the color. The graphs of the experiments using transmitter 2 in figures 8.2b, 8.2d, and 8.2f show a similar distribution in this range, whereas the Galaxy S7 has a greater offset and the Honor 7X's offset interval is shorter, caused by a greater lack of color detection in the yellowish spectrum. In 4 meters distance, there is still a similar offset using both transmitters as shown in all graphs in Figure 8.5. Since the perceived hues in this range in all smartphones have a similar offset, there could be a common factor. This effect can be caused either by the smartphones which could use a filter that interprets the greenish colors in the same way or be produced by the transmitter's LEDs which can transmit greenish colors inaccurately. One possible reason for the use of a filter in smartphones that enhances the green spectrum is that humans are more perceptive to green color spectrum as described in the background. Another possible reason is the use of a filter that adjusts this color range for the human skin to look natural.



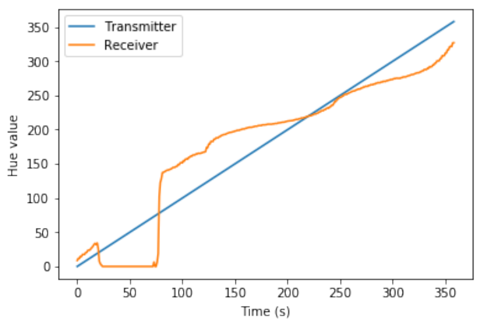
(a) Samsung Galaxy S7 short distance color detection including transmitter 1



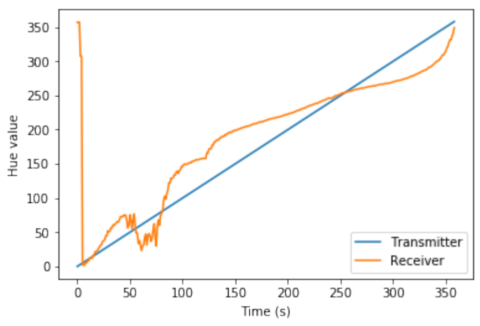
(b) Samsung Galaxy S7 short distance color detection including transmitter 2



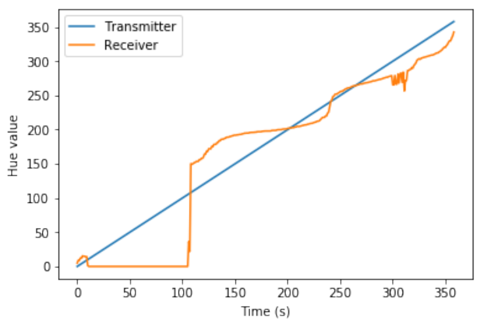
(c) OnePlus 3T short distance color detection including transmitter 1



(d) OnePlus 3T short distance color detection including transmitter 2



(e) Honor 7X short distance color detection including transmitter 1



(f) Honor 7X short distance color detection including transmitter 2

Figure 8.2: The short distance (2 meters) color detection experiment shows that the receiver Samsung Galaxy S7, Honor 7X, and OnePlus 3T detect colors more accurately with transmitter 1 than transmitter 2.

Except for the Samsung Galaxy S7 in short distance using transmitter 1, all other graphs show a similar offset in the range from roughly 280 to 360. Like the offset in the green spectrum, this can either be caused by a camera filter in the smartphones or an inaccurate transmission in both transmitters.

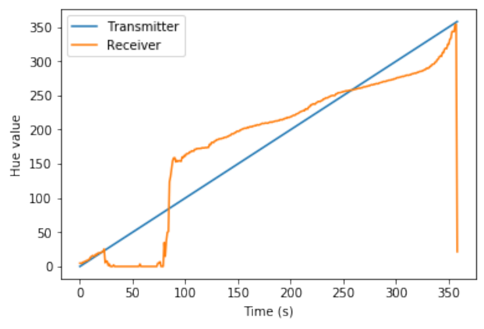
A very significant characteristic in all experiments using transmitter 2 is that the obtained values in the yellowish spectrum (roughly in the first quarter) are way smaller than they are expected. Hue values do not represent amplitude, which means if no color is detected the hue value is 0 by default. Except for the Honor 7X which has some minor peaks in the long distance as shown in Figure 8.3f, the hue value remains constantly 0. This is very likely to be caused by noise because this smartphone does not detect colors in the same range in short distance using the same transmitter in Figure 8.2f. Also, this smartphone is capable of little digital zoom which increases the perceived area of interest in long distance. In other words, the Honor 7X cropped camera frame includes more of the surroundings of the receiver than the other smartphones do.

Thus, all smartphones do not detect the colors in the yellowish spectrum using transmitter 2. Since the color detection is based on saturation and brightness thresholds to eliminate grayish colors, the LED in transmitter 2 may be too bright for an accurate transmission of a faded color like yellow. Besides that, the graph of the Samsung Galaxy S7 in Figure 8.3a indicates that the transmission of the yellowish color using transmitter 1 at long distances is inaccurate.

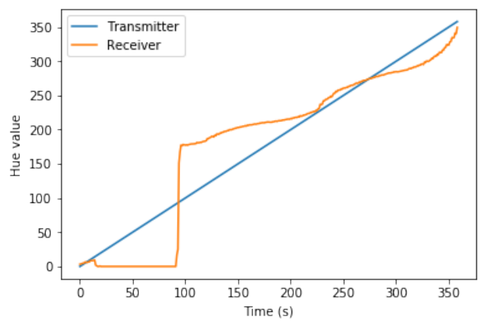
Also, the hue values in the graphs of the OnePlus 3T and Honor 7X show significant drops in the same range which means that some colors are detected but the perception of this color still is unreliable. The fewer colors a camera detects, the more the hue values in the graph drop. According to a noticeable drop distribution in this color range in Figure 8.2e shows that the Honor 7X camera does not detect yellowish colors accurately at short distances. Since the detection of the other smartphones is stable in this range in short distance but not in long distance, indicates that the some colors appear more grayish with increasing distance, likewise yellow.

The graphs of the Honor 7X shows more dramatic drops in short as well as long distance than the other smartphones which indicates it is less reliable for color detection than the others.

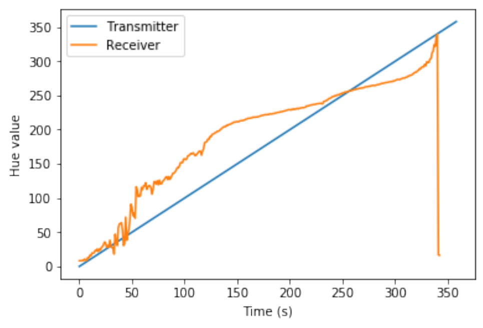
When a transmitter sends one of the three primary colors, all of the receiver graphs are closer to this primary color than to the other two remaining colors which is important for the nearest neighbor algorithm and allows to use FSK4.



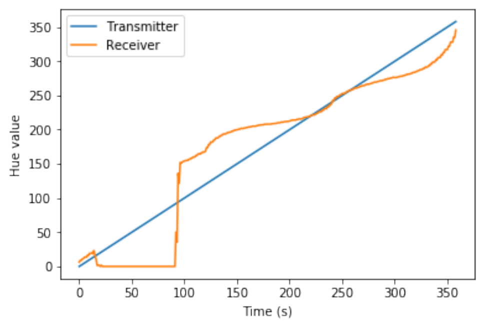
(a) Samsung Galaxy S7 long distance color detection including transmitter 1



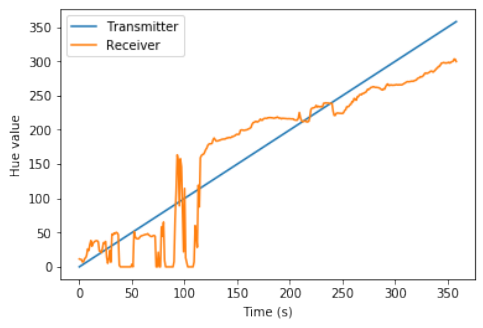
(b) Samsung Galaxy S7 long distance color detection including transmitter 2



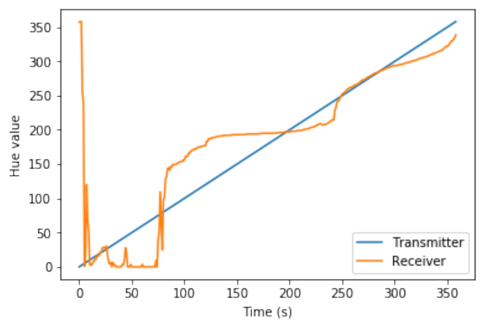
(c) OnePlus 3T long distance color detection including transmitter 1



(d) OnePlus 3T long distance color detection including transmitter 2



(e) Honor 7X long distance color detection including transmitter 1

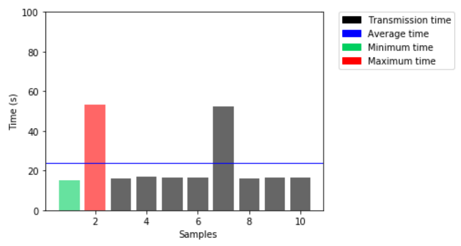


(f) Honor 7X long distance color detection including transmitter 2

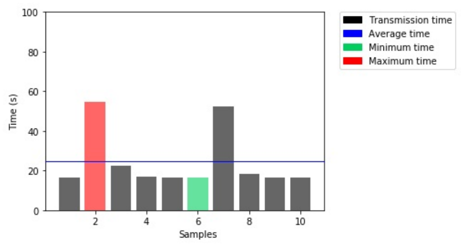
Figure 8.3: The long distance (4 meters) color detection experiment shows that the receiver Samsung Galaxy S7 and OnePlus 3T detect colors more accurately than the Honor 7X.

8.3 Laboratory Experiment II - Performance Measurements

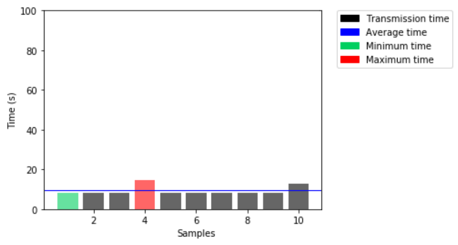
We performed Laboratory Experiment II with different transmitters and receivers. Similar to the color detection setup, this experiment setup includes short distance and long distance experiments. The receiver is mounted on a tripod and the transmitter under the ceiling.



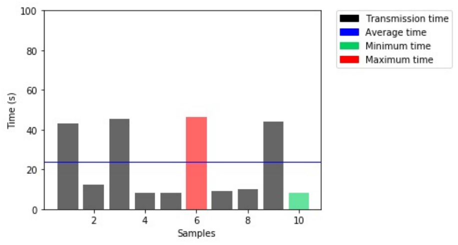
(a) Samsung Galaxy S7 short distance performance measurement including transmitter 1



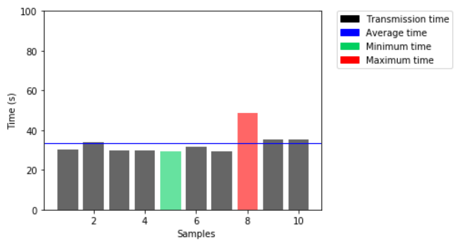
(b) Samsung Galaxy S7 short distance performance measurement including transmitter 2



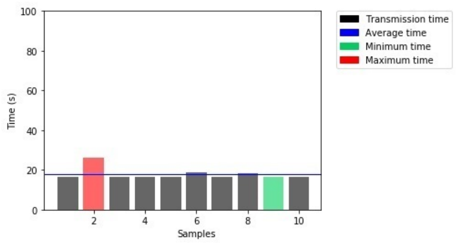
(c) OnePlus 3T short distance performance measurement including transmitter 1



(d) OnePlus 3T short distance performance measurement including transmitter 2



(e) Honor 7X short distance performance measurement including transmitter 1



(f) Honor 7X short distance performance measurement including transmitter 2

Figure 8.4: The short distance (2 meters) performance measurement shows that the OnePlus 3T performs better than the other smartphones using transmitter 1.

In this experiment, the transmitter operates unmodified, transmitting SHA-256 data, i.e., 256 bits (32 bytes) without counting the overhead produced by the starting sequence, symbol stuffing, and CRC8 checksum. The receiver application is modified in order to receive 10 complete transmissions in a row. The results are presented in a chart, where a red bar indicates the longest transmission duration and a green bar the shortest transmission duration. Also, there is an additional blue line that represents the average transmission time.

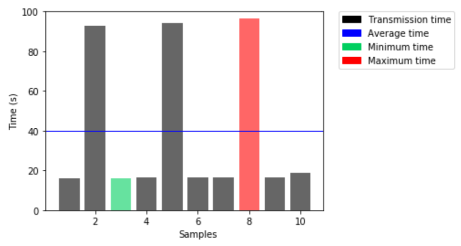
To minimize bit slips, the receivers run half the sample rate which the camera is capable of. For the smartphones Samsung Galaxy S7 and Honor 7X the transmission speed is set to 15 frames per second (FPS), whereas it is set to 30 FPS for the OnePlus 3T.

The results for the Samsung Galaxy S7 and OnePlus 3T both show two ranges of transmission duration. As an example, in the chart of the Samsung Galaxy S7 in Figure 8.4a, 8 out of 10 samples have a duration of 15 to 16 seconds. The shortest bars in a chart represent the short duration level. Two out of 10 bars are significantly higher which have a duration of 52 to 53 seconds and those longer bars can be defined as long duration level. Since there is no sample with a duration in between these levels, there has to be a factor which delays the transmission. As mentioned in chapter 7, the frame sampler block drops incoming frames which do not fit to the specified common transmission interval. Since synchronization is not implemented in the current version of the prototype, it is very likely that the receiver interval is shifted very close to the next transmitter interval and needs some time to recover from it. This would also explain why the differences in between the short and long duration levels are almost the same.

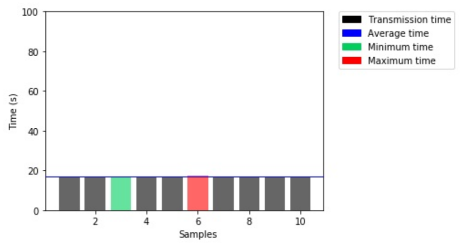
Another conspicuous attribute about the long duration level bars for the Samsung Galaxy S7 using transmitter 1 is that these bars increase for the longer distance, whereas the short duration level bars remain in the same time range. It takes longer for the transmitter and receiver to re-synchronize with this phone. The OnePlus 3T has a similar duration in the long duration levels as shown in Figure 8.4d for short distance using transmitter 2 and in Figure 8.5c for long distance using transmitter 1.

The number of samples which belong to the long duration level varies in the experiments. For example, the Samsung Galaxy S7 has two long duration level bars in short distance using both transmitters in Figures 8.4a and 8.4b. In contrast, the OnePlus 3T has 4 of them in short distance and none of them in long distance using transmitter 2 as shown in figures 8.4d and 8.5d. In all experiments the number of short duration bars is greater than the number of long duration bars.

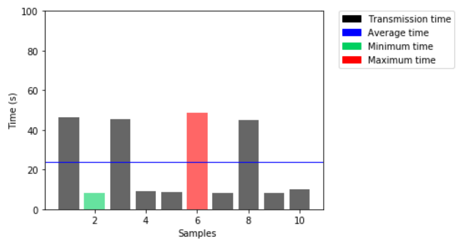
The experiments that do not show long duration levels like 8.5b and 8.5d indicate that the receiver starts at a aligned point of time referring the transmitter which does not mean that there would be no shift if the experiment included more samples.



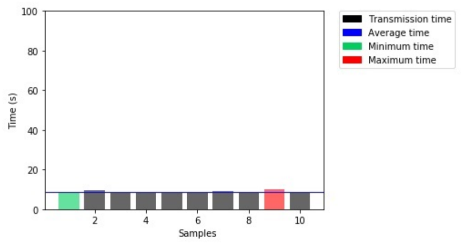
(a) Samsung Galaxy S7 long distance performance measurement including transmitter 1



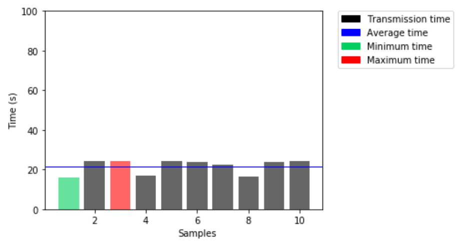
(b) Samsung Galaxy S7 long distance performance measurement including transmitter 2



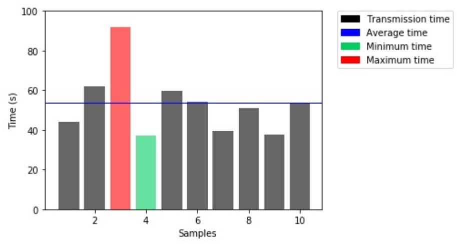
(c) OnePlus 3T long distance performance measurement including transmitter 1



(d) OnePlus 3T long distance performance measurement including transmitter 2



(e) Honor 7X long distance performance measurement including transmitter 1



(f) Honor 7X long distance performance measurement including transmitter 2

Figure 8.5: The long distance (4 meters) performance measurement shows that the smartphones Samsung Galaxy S7 and OnePlus 3T perform better than the Honor 7X using transmitter 2.

In this experiment, the Honor 7X performs differently in various setups, whereas the Samsung Galaxy S7 and OnePlus 3T have a lot of similar distributions in terms of long and short duration levels. For example, in a setup including transmitter 2 in long distance, the Honor 7X does not show short and long duration levels since the time varies in each sample. Also the times increase significantly in comparison to the ones in short distance shown in Figure 8.4f. Another example is that the times in the short duration level bars, as shown in Figure 8.4e, are greater than the ones for the Samsung Galaxy S7 in Figure 8.4a running at the same frame rate.

8.4 Laboratory Experiment III - Performance in Practice

In order to know how this authentication approach performs in practice, we introduce an experiment with two people who are not familiar with the project. The goal is to gain information about the time that the transmission needs and which factors effect the duration. Two test persons performed the test twice either on the smartphone Samsung Galaxy S7 or OnePlus 3T. The distance between the transmitter and the receiver was between two and three meters. The smartphones were handheld during the entire transmission and the time measurement started as soon as the transmission was requested through the receiver application.

The first test person used the Samsung Galaxy S7 which runs at 15 fps. In the first try the transmission duration took 55.7 seconds. During the transmission, test user one aligned auto exposure compensation to fit the lightning conditions. Also, test user one tried different levels of zoom to get closer to the transmitter. Consequently, the camera was not pointed to the transmitter correctly and some frames were dropped. In the second try, test user one aligned auto exposure compensation and level of zoom immediately and the transmission took only 17.9 seconds.

The second test person used the OnePlus 3T which runs at 30 FPS. Test Person two did not change any settings and the transmission took 10.5 seconds. Also, the second try did not include any change and the transmission took 9.6 seconds.

This experiment shows that test persons who are not familiar with the application can adjust camera settings, like auto exposure compensation or level of zoom, to improve the perception of the transmitter's LED. Also, devices like OnePlus 3T do not require adjustments in this setup. Moreover, the UI, including progress indication and adjustment components, seemed to be user friendly for both test persons.

9 Results and Analysis

This chapter discusses the results with respect to the concept requirements in 3 and the guiding questions in 1.

One of our concept requirements was to implement an authentication system that does not require special equipment.

Referring to the requirements, in our protoype the transmitter is equipped with a LED, which is a low-cost component. The receiver prototype uses Android based smartphones. The smartphone can be handheld and the range between the transmitter and the receiver can be described as "across the room". As the performance in practical tests show, the concept is easy to use and self-explanatory. The authentication concept uses a public key infrastructure certificate to prove the transmitter's identity. Using the VLC channel to transmit the fixed size certificate's SHA-256 fingerprint, there is no need of a high-bandwidth VLC channel. This concept enables the dynamic generation of authentication data, e.g., certificate rotation.

With reference to the first guiding question, we use the Android Camera2 API for image receiving and the computer vision library OpenCV for image processing in the Android receiver application. This application allows a user to receive visible light transmission and also provides feedback about the transmission progress to the user at the same time. This is achieved by a multi-threaded application architecture which also enables the application to work on various Android smartphones with different hardware capabilities.

As described in the color detection experiment, all smartphones we use are capable of receiving the primary colors over short (2 meters) and long distances (4 meters). This allows us to map three symbol values to the primary colors and one symbol value to black in the FSK4 modulation scheme. Also, the graphs in this experiment show that different smartphone models perceive some colors differently. For example, the Honor 7X detects colors in some channels less accurately than the Samsung Galaxy S7 and the OnePlus 3T. Consequently, the built-in camera determines the accuracy of color detection. Also, the hardware components of the Android smartphone determine the processing speed.

Referring to the second guiding question, the processing architecture of the

Android application consists of different building blocks. A frame cropper block minimizes the workload by cropping incoming image frames. The color detection block calculates the average color in the image frame and the demodulator block demodulates the color to the corresponding symbol using a modulation scheme. The sampler block forwards symbols that match the common symbol rate between the transmitter and the receiver. The line decoder block collects the symbols for each packet and forwards them to the error detector which checks if the packets are received correctly. The FEC decoder block performs error correction.

The third guiding question is about the usability of this approach in terms of transmission speed and reliability. We define an authentication system to be usable when the time to complete the authentication process is comparable or shorter than other similar authentication methods. Since this project is about authentication of remote IoT devices, other authentication mechanisms require special equipment to physically reach the device, e.g., a ladder.

As discovered in the performance measurement experiment, the transmission time can vary across smartphones. The supported camera frame rates determine the transmission time. For example, the OnePlus 3T runs at 60 fps and the overall minimum transmission time is 8.1 seconds, whereas it is 15.3 seconds for the Samsung Galaxy S7 which runs at 30 fps. Also, the setup, for example, using different transmitters and distances, can affect transmission times.

In a majority of cases during our performance experiments, the lack of clock synchronization between the transmitter and the receiver did not seem to be a problem. In other cases, the transmission time is significantly higher. For example, the longest transmission time was 48,8 seconds using transmitter 1 over the short distance. The minimum time was 8.2 seconds, and the average time was 23.8 seconds.

Nevertheless, our system can still be described as usable since the performance in practical experiments show shows that test users who were not familiar with the project could complete the transmission successfully in times that may be faster than fetching equipment for physically accessing the remote IoT device. Our experiments show that a user can adjust the camera settings to enable a more reliable and fast transmission. Moreover, when the user does not receive any feedback at the start of a transmission, he or she can immediately restart the transmission to avoid situations with a big time shift in between the transmitter and receiver interval.

10 Security Considerations

This chapter discusses possible attack vectors that may affect the VLC authentication process.

Compromised IoT devices and smartphones are out of scope. If an attacker can modify a device, it can also modify the VLC transmission itself. There is no way to prevent this kind of attack.

A possible social engineering (SE) attack could be a cloned receiver application which can be downloaded from the application store. The UI could look similar to the original one, and the application could accept any transmission, regardless whether the camera is pointed to a different device. Another SE approach could include ways to trick someone into bypassing the authentication process.

There are different ways to prevent such attack vectors. Minimizing the chance of SE attacks in business environments includes security awareness programs for employees. This also applies for cloned receiver applications in the application stores. For example, the user has to ensure that the receiver application has been installed from a trusted source.

An attacker could perform a denial-of-service attack by preventing VLC by destroying or covering the LED of the transmitter or the smartphone camera.

Another possible attack vector could be message tempering. The attacker could use a more powerful light source to transfer its own certificate's fingerprint to the receiver application. The attacker could also try to use the infrared light spectrum for a transmission that is not perceptible by human eyes.

Message tempering attacks can easily be spotted since the attacker would have to be physically in sight-line. The user can avoid other visible blinking light sources because he or she is in control of pointing the camera and setting the area of interest within the image. Although human's eyes cannot perceive infrared lights, the user can still detect the light source in the application's image and the application would not recognize infrared sources as colors.

Besides that, an attacker could use cameras to record entire the transmission and try perform an offline attack. Afterwards, the attacker could analyze the data that was sent.

There is no need to perform an offline attack since the transmitter only sends publicly available information which is unencrypted.

11 Conclusion

The increasing number of deployed IoT devices creates the need to securely authenticate these devices. Due to a lack of trusted CAs for IoT devices in local networks, there is no efficient way to securely authenticate remote ones without physical interaction, such as pressing buttons or reading QR codes. For devices that are hard to reach, a visible light channel can serve as an OOB channel resistant to active MITM attacks. Using VLC, the transmitter proves its authenticity to the user remotely. The transmitted data can be dynamically generated and enables certificate rotation. VLC-based authentication as implemented in this thesis does not require any special equipment. Any ordinary Android based smartphone runs the receiver application and a LED requires very little space in the IoT device's form factor. Also, LEDs are very low cost and many IoT devices are already equipped with LEDs. These are mostly monochromatic, but can easily be replaced with a tri-color LED for faster transmission.

Our experiments show that the perception of colors and the transmission performance vary across different smartphones.

In the experiments, the transmission of a SHA-256 fingerprint of a TLS certificate took approximately 8 seconds for a OnePlus 3T smartphone with capturing images at a sampling rate of 60 fps. On Samsung Galaxy S7 with a camera running at 30 frames per second the same transmission took roughly 15 seconds as well as for the Honor 7X with the same setting.

A small number of experiments yielded a significantly larger transmission time, up to 48 seconds for the OnePlus 3T. This was caused by a lack of receiver synchronization, as we did not implement such a capability in our prototype.

We tested our system in the lab, including with two lab members who were not involved in the design. The test shows that the current state of the prototype is practically usable. The testers were able to receive the SHA-256 data in less time than would likely have been required to gain physical access to the IoT device, e.g., by ladder.

The implementation of the receiver as well as the transmitter is open source and can be obtained from GitHub [13].

11.1 Limitations

This authentication system is limited to the line of sight and only works where the IoT device can be clearly seen by the camera. The transmission may also fail with strong ambient light. The area surrounding the transmitter LED needs to be white or gray. The smartphone camera characteristics, .e.g., the set of the supported camera frame rate, limits the transmission speed. Also, the smartphone's hardware capabilities determine the processing delay.

11.2 Implications for Future Research

Future research could focus on increasing the transmission speed. The system could use higher modulation schemes like FSK8. To enable FSK8 modulation, one has to introduce four more colors. Since colors can be perceived differently on various smartphones, we suggest using a color mapping table. The color mapping table can be initialized once in a calibration mode. The calibration mode can detect all colors that the smartphone camera is capable to differentiate. Thus, the number of colors can determine which modulation scheme is best for the transmission. Besides that, a modulation scheme that is based on amplitude and different colors (frequency) could increase the transmission speed.

The implementation of receiver synchronization could help to avoid situations in which the receiver needs time to recover from interval shifting. When the transmission rate is set to half of the maximum camera sample rate, there are two frames in each transmission interval which can help to determine the middle of each interval to set as a timestamp for the next forwarding of a frame.

To generate less overhead in line coding, symbol stuffing can be replaced with bit stuffing.

Another idea to increase the transmission speed is to use an IoT device with multiple LEDs. For example, the Amazon Echo has a strip that contains multiple LEDs. Since our system uses the RaptorQ fountain code, it does not matter in which order packets are received. The receiver application could be extended by the feature to specify multiple area of interests.

An idea to make the transmission more reliable is to periodically send calibration packets. The perceived colors can overwrite the ones stored in a color mapping table. This allows the receiver to automatically adjust to changing lighting conditions.

The transmitter could also be ported to CPU constrained devices, like an Arduino. The time-consuming task of RaptorQ packet generation can be done beforehand and stored locally. But this would disable the certificate rotation feature on this device.

Currently there are two major operating systems for smartphones. One is Android and the other one is iOS. To implement an iOS based receiver application we would need to use the iOS camera API. Nevertheless, the implementation has to rely on the features this API provides. The receiver implementation part of the shared library would need to be re-implemented because iOS cannot run Java applications. Also, one would need to find OpenCV and RaptorQ libraries for the iOS platform.

Bibliography

[1] Wi-Fi Alliance. "Wi-fi protected setup specification". In: *WiFi Alliance Document* (2007).

[2] *Amplitude Modulation.* https://github.com/PetarV-/TikZ/tree/master/. Accessed: August 2018.

[3] *Android Camera2 Documentation - Camera Device.* https://developer.android.com/reference/android/hardware/camera2/CameraDevice. Accessed: August 2018.

[4] *Android Camera2 Documentation - Capture Request.* https://developer.android.com/reference/android/hardware/camera2/CaptureRequest.html. Accessed: July 2018.

[5] *Android Camera2 Documentation - Image Format.* https://developer.android.com/reference/android/graphics/ImageFormat.html. Accessed: July 2018.

[6] *Android Camera2 Documentation.* https://developer.android.com/reference/android/hardware/camera2/package-summary. Accessed: July 2018.

[7] *Averages/Mean angle.* https://rosettacode.org/wiki/Averages/Mean_angle. Accessed: July 2018.

[8] Steven M. Bellovin and Michael Meritt. "Encrypted Key Exchange: Password-Based Protocols Secure Against Dictionary Attacks". In: *Proceedings 1992 IEEE Computer Society Symposium on Research in Security and Privacy.* IEEE. 1992. URL: https://www.cs.columbia.edu/~smb/papers/neke.pdf.

[9] Noy Cohen. *A Color Balancing Algorithm for Cameras.* Tech. rep. 2011. URL: https://stacks.stanford.edu/file/druid:my512gb2187/Cohen_A_New_Color_Balancing_Method.pdf.

[10] *Color Physics.* http://facweb.cs.depaul.edu/sgrais/images/ColorTheory/emspectrum.gif. Accessed: May 2018.

[11] Ericsson. *Ericsson Mobility Report*. Tech. rep. 2016. URL: `https://www.ericsson.com/assets/local/mobility-report/documents/2016/ericsson-mobility-report-june-2016.pdf`.

[12] *Frequency Modulation*. `https://github.com/PetarV-/TikZ/tree/master/`. Accessed: August 2018.

[13] *GitHub Repository Project Lighthouse*. `https://github.com/columbia-irt/lighthouse`.

[14] *Guava: Google Core Libraries for Java*. `https://hub.com/google/guava`. Accessed: July 2018.

[15] *HSV cylinder*. `https://commons.wikimedia.org/wiki/File:HSV_cylinder.jpg`. Accessed: July 2018.

[16] P. Hu, P. H. Pathak, X. Feng, and P. Fu H. Mohapatra. "ColorBars: Increasing Data Rate of LED-to-Camera Communication using Color Shift Keying". In: *Proceedings of the 11th ACM Conference on Emerging Networking Experiments and Technologies*. 2015. URL: `http://spirit.cs.ucdavis.edu/pubs/conf/pengfei-conext15.pdf`.

[17] N. Ibraheem, M. Hasan, R. Khan Khan, and P. Mishra. "Understanding Color Models: A Review". In: *ARPN Journal of Science and Technology* (2012).

[18] Qualcomm Incorporated. *RaptorQ Technical Overview*. Tech. rep. 2010. URL: `https://www.qualcomm.com/media/documents/files/raptorq-technical-overview.pdf`.

[19] Kensei Jo, Mohit Gupta, and Shree K. Naya. "DisCo: Display-Camera Communication Using Rolling Shutter Sensors". In: *ACM Transactions on Graphics (TOG)* (2016). URL: `http://www1.cs.columbia.edu/CAVE/publications/pdfs/Jo_TOG16.pdf`.

[20] Arun Kumar, Nitesh Saxena, Gene Tsudik, and Ersin Uzun. "Caveat eptor: A comparative study of secure device pairing methods". In: *Pervasive Computing and Communications, 2009. PerCom 2009. IEEE International Conference on*. IEEE. 2009, pp. 1–10.

[21] H. Lee, H. Lin, Y. Wei, H. Wu, H. Tsai, and K.C. Lin. "RollingLight: Enabling Line-of-Sight Light-to-Camera Communications". In: *Proceedings of the 13th Annual International Conference on Mobile Systems, Applications, and Services*. 2015. URL: `https://www.csie.ntu.edu.tw/~hsinmu/wiki/_media/paper/mobisys15.pdf`.

[22] Jonathan M McCune, Adrian Perrig, and Michael K Reiter. "Seeing-is-believing: Using camera phones for human-verifiable authentication". In: *IEEE symposium on security and privacy*. IEEE. 2005, pp. 110–124.

[23] *Modulation and Different Types of Modulation*. `https://www.electronicshub.org/modulation-and-different-types-of-modulation/`. Accessed: August 2018.

[24] Hagen Odenthal. *Secure Authentication of Remote IoT Devices Using Visible Light Communication; Transmitter Design and Implementation*. Tech. rep. 2018.

[25] *OpenCV (Open Source Computer Vision Library)*. `https://opencv.org/`. Accessed: April 2018.

[26] Nishad PM and Dr. R.Manicka Chezian. "VARIOUS COLOUR SPACES AND COLOUR SPACE CONVERSION ALGORITHMS". In: *Journal of Global Research in Computer Science* (2012).

[27] Charles Poynton. *Chroma subsampling notation*. Tech. rep. 2008. URL: `https://poynton.ca/PDFs/Chroma_subsampling_notation.pdf`.

[28] *RGB Circles*. `http://colorizer.org/img/rgb.png`. Accessed: July 2018.

[29] Claudio Soriente, Gene Tsudik, and Ersin Uzun. *BEDA: Button-Enabled Device Association*. 2007.

[30] Claudio Soriente, Gene Tsudik, and Ersin Uzun. "HAPADEP: human-assisted pure audio device pairing". In: *Information Security Journal*. Springer-Verlag, 2008, pp. 385–400.

[31] Stefan Viehboeck. *Brute forcing Wi-Fi Protected Setup*. Tech. rep. 2011. URL: `(https://sviehb.files.wordpress.com/2011/12/viehboeck_wps.pdf`.

[32] Yanbing Yang and Jun Luo. *Boosting the Throughput of LED-Camera VLC via Composite Light Emission*. Tech. rep. 2018. URL: `http://www3.ntu.edu.sg/home/junluo/documents/CASK.pdf`.

[33] Jiansong Zhang, Zeyu Wang, Zhice Yang, and Qian Zhang. "Proximity Based IoT Device Authentication". In: *IEEE INFOCOM 2017 - IEEE Conference on Computer Communications*. IEEE. 2017. URL: `(https://www.microsoft.com/en-us/research/wp-content/uploads/2017/02/IoTAuthenticationInfocom2017-2.pdf`.

Erklärung

Hiermit versichere ich, dass die vorliegende Arbeit selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt wurden.

Ferner habe ich vom Merkblatt über die Verwendung von Masterabschlussarbeiten Kenntnis genommen und räume das einfache Nutzungsrecht an meiner Masterarbeit der Universität der Bundeswehr München ein.

(Alexander Linßen)