A Practical Approach: Implementing Security for limited *packet size*

- Encryption for a *minute packet protocol*

LUNDS UNIVERSITET
Lunds Tekniska Högskola

LTH School of Engineering at Campus Helsingborg
Elektro- och informationsteknik

Bachelor thesis:
Alexander Sköld

LTH School of Engineering
Lund University
Box 882
SE-251 08 Helsingborg
Sweden

LTH Ingenjörshögskolan vid Campus Helsingborg
Lunds universitet
Box 882
251 08 Helsingborg

Printed in Sweden
E-husets tryckeri
Lunds universitet
Lund 2017

Abstract

This report describes the development of an encrypted tunnel for the Crazyflie drone made by Bitcraze. This project began with the need for protection while using a Crazyflie drone in a public space. The Crazyflie drone is using open source software and has no protection against hijacking. Any phone able to download the app to control the drone is able to hijack and control the drone.

To remedy this problem the communication between the Crazyradio and Crazyflie was encrypted. The encryption was implemented in such a way as to create a secure tunnel while only a minimum amount of code needed to be changed. The method of encryption chosen was AES 128 using GCM. The memory footprint of the encryption solution on the drone was important and therefore minimised.

The resulting solution does protect against the type of hijacking a person with a phone is able to do. Furthermore, the strength of the encryption is of good enough quality to dissuade attacks from opportunistic attacks.

Keywords: Security, Encryption, AES, GCM, Tunnel, Open source.

Sammanfattning

Denna rapport beskriver en krypterad tunnel utvecklad för Crazyflie-drönaren som är skapad av företaget Bitcraze. Projektets mål är att minimera risken för att Crazyflie drönaren blir kapad i allmänna utrymmen. Då drönaren inte kan begränsa kontrollen till endast en styrenhet så kan den bli kapad av att någon i närheten använder kontroll-appen.

En lösning till detta problem är att kryptera kommunikationen från en drönare till en styrenhet. Implementationen av krypteringen är gjord på så sätt att en tunnel upprättas mellan drönaren och styrenheten. Varken koden på drönaren eller på klienten behöver förändras för att tunneln ska fungera. Krypteringsalgoritmen är AES 128 med GCM. Då drönarens minne är begränsad var det viktigt att minimera krypteringsbiblioteket storlek.

Resultatet av projektet blev en kryptering som stoppar att någon på en offentlig plats från att ladda ned ett styrprogram för att kapa drönaren. Opportunistiska anfall klarar även krypteringen att stoppa.

Nyckelord: Säkerhet, kryptering, AES, GCM, Tunnel, Öppen källkod

List of contents

1 Introduction

1.1 Background

The Crazyflie drone began as a competence development project in 2009. The project was started by Arnaud Taffanel, Marcus Eliasson and Tobias Antonsson. The Crazyflie is an open-source hardware and software drone. As the first Crazyflie was completed in 2011 the group started a company called Bitcraze to fund further development. This project is done for Bitcraze as a first step into encrypting the communication between the Crazyflie and Crazyradio.

1.1.1 Bitcraze

Bitcraze as a company was founded in early 2011 by the creators of the Crazyflie. The main purpose of Bitcraze is to fund the further development and production of the Crazyflie. Additional editions of the Crazyflie and Crazyradio have been released as well as new projects.[14]

The Crazyflie is not the only product Bitcraze produces however. CrazyRadio is a USB-radio card used to connect to the Crazyflie using Enhanced Shockburst among other products. Enhanced Shockburst is a radio protocol and explained in more detail in the Technical background section. One mentionable product is the android application to controll the Crazyflie 2.0 via Bluetooth Low Energy (BLE).

1.1.2 Needed functionality

Customers have a special use case for the small drone that is currently not accounted for. Some customers want to use the Crazyflie drone in publicity events or air shows. As the drone does not currently have any type of hijacking protection adding a layer of encryption to the controlling communication would provide sufficient protection for these events.

1.2 Purpose

To protect the drone against hijacking this project aims to encrypt the communication between the Crazyradio and the Crazyflie. The level of protection needed to reduce the risk of accidental or opportunistic hijacking is not very high.
Accidental hijacking is when two drones are in proximity with one another and share the radio space. This proximity results in either both radio controllers controlling both drones or they interfere causing loss of control over either drone.
Opportunistic hijacking is when someone with malicious intent actively acquires a controller in order to hijack a drone. This is usually the android application but could easily be the Crazyradio connected to a laptop.

1.3 Goals

Adding encryption to the communication between the drone and radio is the main goal of this project. The encryption should be added to existing code in such a way to maintain modularity. This means that the encryption functionality should be isolated so that it is easy to disable or remove encryption with minimal changes to the code. The way modularity was planned was to make the encryption act as a secure tunnel.

1.4 Problem definition

What is needed to implement protection for the drone is described in this section.

1. What encryption algorithm is to be used to encrypt the communication?
2. What programming language will be used to implement encryption on the client?
3. What programming language will be used to implement encryption on the drone?
4. What cryptographic library will be used on the Crazyflie?
5. What cryptographic library will be used on the client?
6. What does the secure tunnel protect against?

1.5 Motivation

Drones have become a large part of society in recent years and pose interesting questions when it comes to computer safety. Because drones are flying machines they almost exclusively are controlled by radio communication and because the publicly available bandwidth is rather limited it can become the limiting factor in drone usage. Hijacking a flying drone implies not only of a risk to personal property but also towards the health of all those in the surrounding area, although in this case the health risk is minute.
The most direct solution to these problems is to ensure that the drones are controllable is to make the communication unique to every drone.

1.6 Scope limits

Controlling the drone with the android application is not covered in this project, only using the USB-dongle.

The communication is to be encrypted using AES 128-bit Galois Counter Mode. Sharing of the key is limited to programming it in before flashing the drone.

There are a few exceptions to the encryption. The received signal strength indication(RSSI) message and flashing the drone are both exceptions to encryption.

The RSSI is a message that is sent from the drone when a packet is received and it has nothing to send back. The message will not be encrypted as it serves a purpose of checking whether or not the radio signal is strong enough to be received reliably in the first place.

Flashing is when the client writes new firmware on the drone via the Crazyradio. Encryption is handled in the firmware of the drone. If the data that is supposed to write the firmware is encrypted and there is no code for decrypting, the firmware will be corrupted.

2 Technical background

This chapter goes into detail about all of the important technical aspects of the project and is divided into five parts. The first part describes the Crazyflie drone components and what they do. The second part describes the Crazyradio and the Client and how they interact with one another. The third part describes what protocols are used. The fourth part describes how AES-GCM and the secure tunnel works. The fifth part describes the cryptographic libraries and the client. Figure 1 gives a general overview of where the different pieces of fit in relation to one another.

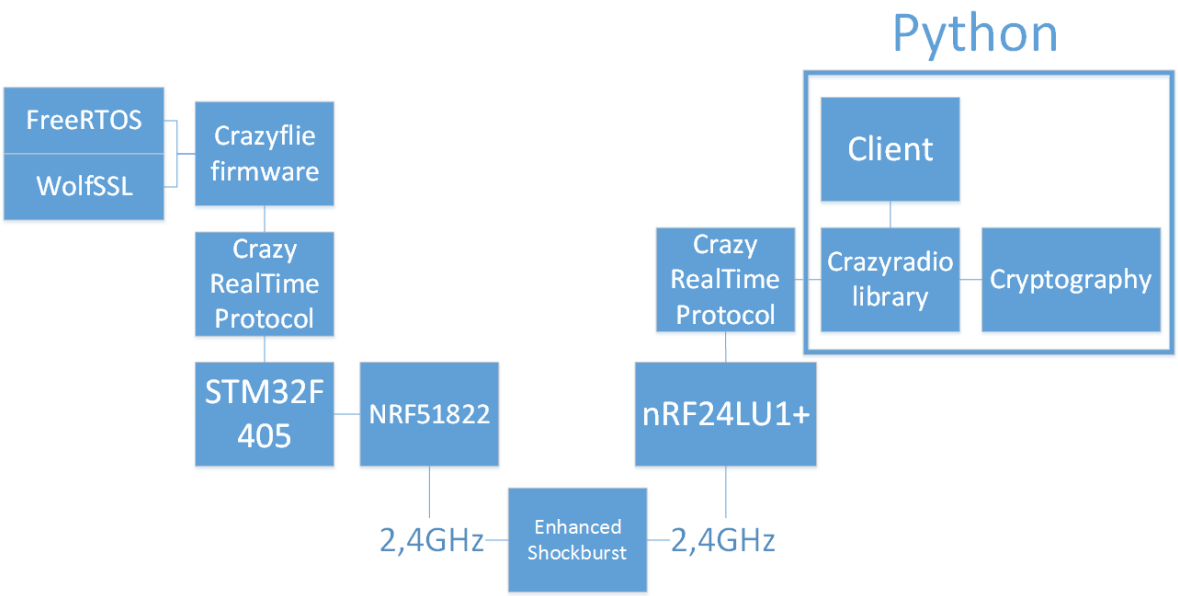

Figure 1. Complete overview of the technical setup and how they communicate.

2.1 The Crazyflie drone

This section describes the central parts of the hardware of the Crazyflie.
The Crazyflie is divided into two main components, the radio(nRF51822) and the main processing chip(STM32F405). The nRF51822 handles all of the raw radio communication. The received packets are then sent to the STM32F405 where the Crazyflie firmware handles them. Figure 2 gives an overview of the Crazyflie components.[13]

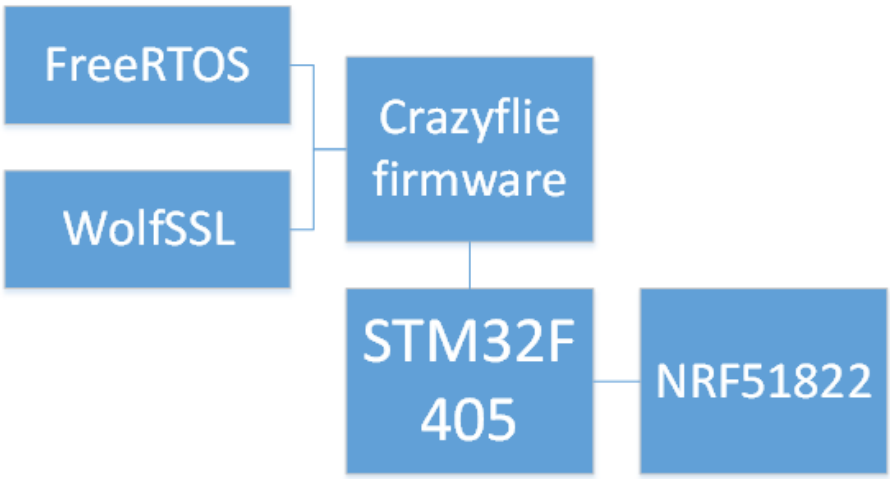

Figure 2. Overview of interactions between the Crazyflie firmware, cryptographic library and hardware.

2.1.1 nRF51822

The nRF51822 chip is the communications microcontroller used on the Crazyflie. The chip was developed by Nordic Semiconductor and is a chip specialising in 2.4 GHz wireless communication. The chip is based on an ARM Cortex M0 processing core with 128 Kbytes of flash memory and 16 Kbytes of RAM. On the Crazyflie the nRF51 sends and receives all of the communication whether by Bluetooth Low Energy or by the Nordic Semiconductor proprietary protocol Enhanced ShockBurst. The data protocol used within these physical protocols is the Crazy RealTime Protocol (CRTP). This protocol is quite simple as it is only a header byte and 31 databytes. The nRF51 comes with a selection of precompiled binaries called soft devices. The one used in the Crazyflie is Soft device 110.[6]

2.1.2 STM32F405

STM32F405RG is the main microcontroller used in the Crazyflie. The chip handles all of the flight controls and the majority of signal processing. The chip was developed by ST micro electronics and has a ARM 32bit Cortex M4 processing core. The chip has 1 Mbyte of flash memory, 129 Kbytes of SRAM and has a clock speed of 168MHz. The libraries used are the STM32F10x and STM324xx.[8]

2.1.3 FreeRTOS

FreeRTOS is essentially the operating system of the Crazyflie handling the scheduling of processes. FreeRTOS is a real time operating system used on the Crazyflie drone to control the flight calculations. It is owned, developed and maintained by Realtime Engineers Ltd. FreeRTOS is used for scheduling and buffering the incoming and outgoing packets.[4]

2.2 The Crazyradio

The CrazyRadio is a USB-radio used by the client. Connected to a computer the CrazyRadio is controlled through a software called Crazyflie Client. The Crazyflie Client is a python program that uses the Crazyradio Library to interact with the CrazyRadio. Figure 3 emphasises the portion of Python code and minimise the firmware for the nRF24LU1+.

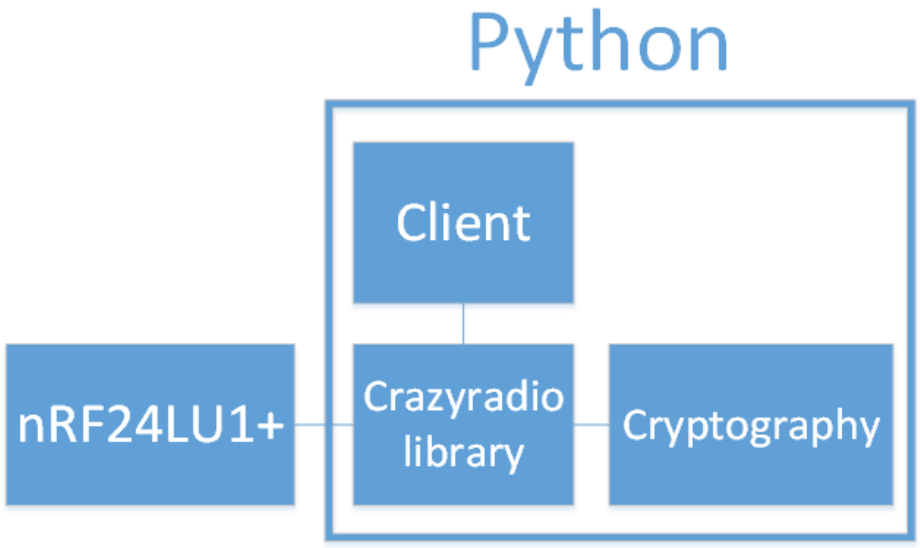

Figure 3. Overview of the client side of the project.

2.2.1 nRF24LU1+

The nRF24LU1+ is the microcontroller used on the CrazyRadio unit. The chip's main purpose is to communicate with the Crazyflie drone and thereby handling the lower level protocol Enhanced Shockburst. The chip has an 8-bit processing core at 16MHz with 2Kbytes + 256 Kbytes RAM and 16 to 32 Kbytes of flash memory.[5]

2.2.2 Crazyradio Library

The Crazyradio library is a library developed by BitCraze in Python to handle the communication with the Crazyflie via the Crazyradio. This library is where the encryption was implemented for the Crazyradio.[11]

2.3 Communication protocol

The protocols used by the Crazyradio and Crazyflie are Crazy RealTime Protocol and Enhanced Shockburst. Every packet sent from the Crazyradio from the Crazyflie is first encoded into a Crazy RealTime Protcol packet and then sent via the radio. The radio encodes the packet into the payload of an

Enhanced Shockburst packet. Figure 4 shows the network stack of the Crazyradio and Crazyflie.

Figure 4. The network stack of both the Crazyradio and Crazyflie. Enhanced Shockburst is the physical layer and Crazy Real Time Protocol is a layer above it.

2.3.1 Enhanced Shockburst

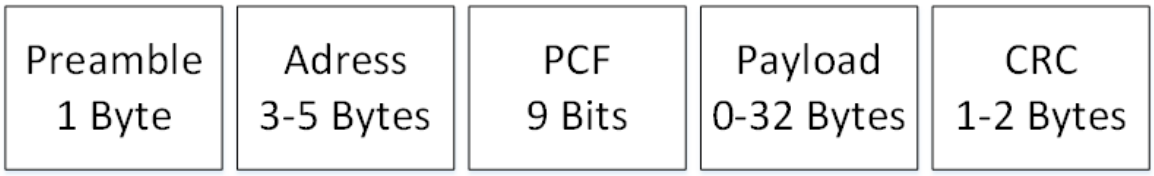

Figure 5. The Enhanced Shockburst packet divided into its five segments.

Enhanced Shockburst (ESB) is a proprietary radio protocol developed by Nordic Semiconductor. The protocol uses the 2,4 GHz portion of the industrial, scientific and medical (ISM) radio band. The protocol supports three data rates of 250 Kbps, 1 Mbps, and 2 Mbps. The packet is a maximum of 40 bytes and 9 bits. The packet has five segments where the payload is a segment with 32 bytes.[7]

The ESB packet is structured in five segments. The first segment is a preamble of one byte. The preamble segment is used to synchronise the receivers demodulator. The second segment is for the address and is between 3 and 5 bytes long. The address bytes are to specify which radio channel the transmitter is sending to. The third segment is the Packet Control Field(PCF) and is 9 bits long. The packet control field itself has three segments. The first segment describes the payload length and is 6 bits long. The second segment contains a PID that is 2 bits long. The PID indicates if the packet is a new packet or a resent packet. The last segment is the no-acknowledgement (NO_ACK) flag. The NO_ACK flag is used when the auto acknowledgement feature is used. When the flag is high, the receiver does not send an auto acknowledgement. The fourth segment is the payload, the payload is between

0 and 32 bytes long. There are two ways to handle the payload segment. Static mode where every packet has a 32 byte long payload segment or dynamic mode where the length of the payload segment is only as long as the actual data. The fifth segment is the CRC that is either one or two bytes long. The CRC is calculated over the PCF and Payload.[21]

2.3.2 Crazy Real Time Protocol

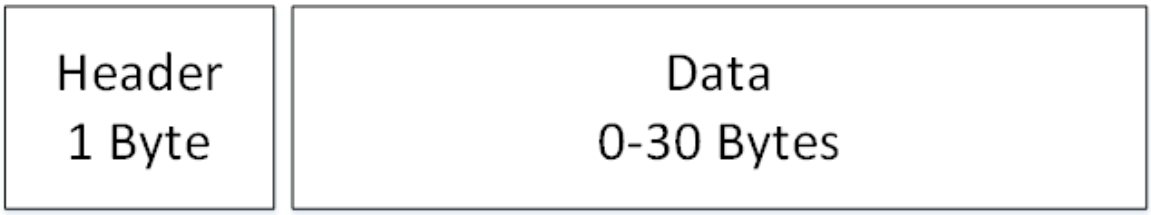

Figure 6. The Crazy Realtime Protocol packet divided into its two parts.

The Crazy RealTime Protocol (CRTP) is a protocol developed by Bitcraze to manage the radio communication over the physical layer. The CRTP packet has a maximum of 30 bytes of data and one byte of header information. The header information is split into 3 segments. The first segment containing the most significant bit is the port address. This segment is four bits long and specifies what subsystem the message is for. The second segment is two bits long and is reserved for future use. The last segment is also two bits long and contains the channel number. The channel number specifies what the subsystem is to do with the packet.[15]

2.4 AES-GCM and Secure Tunneling

This section goes into detail about the AES and its GCM mode. Both AES and GCM are standards published by NIST. As they are published standards the reference material for this section are the official publications. These publications are specified in reference number 1 and 2. Reference 3 is a book used by the report writer to gain a better understanding of the subject.[1,2,3]

2.4.1 Advanced Encryption Standard

Advanced Encryption Standard(AES) is an encryption standard published by the National Institute of Standards and Technology (NIST). It was published in November of 2001. AES is based on the Rijndael block cipher specifically the 128 bit block size version.

The 128 bit Rijndael cipher, called AES for the rest of the report, is a cipher based on Galois fields or finite fields specifically. The cipher works by first dividing the cleartext in to blocks of 16 bytes or 128 bits and then processes them individually at first and then scrambling them together. The cipher works

in four layers, the byte substitution layer, shift row layer, mix column layer, and key addition layer.

In the byte substitution layer the block has all of its bytes substituted using a matrix where its value in hexadecimal decided what byte it will be substituted with.

The shift row and mix column layers are there to disrupt the original order of the bytes and to spread them out evenly from every block to every other block. Key addition is done with a bitwise exclusive-or operation of the block and a subkey derived from the input key.

The layers are executed in a specific order called rounds. These rounds consist of in order the byte substitution layer, shift row layer, mix column layer, and the key addition layer. The number of rounds AES uses to encrypt any data is determined by the size of the key. The different key sizes AES supports are 128, 192, and 256 bits of length and the number of rounds for the respective keys are 10, 12, and 14 rounds. The orders are however not used identically in every round. In the first round there is an additional key addition layer before the byte substitution layer while the last round contains no mix column layer.

2.4.2 AES Galois Counter Mode

Galois Counter Mode (GCM) is a combination of the AES counter mode and a method to generate a Message Authentication Code (MAC). These functions are called GCTR and GHASH. GCTR is simply AES in counter mode and GHASH is a hashing function using galois fields.

2.4.2.1 Counter mode

CTR or Counter mode is a mode of AES encryption.
Every block of data to be encrypted is combined with an initial value and a counter, they are added by the use of an or-operation. There are a few important aspects of the CTR mode. CTR is able to be computed in parallel. Because none of the blocks are dependant on any other and only on the counter and the initial value the encryption is able to be scaled up. The initial value used is an important factor. The value is based on the block size of the data to be encrypted. Due to AES only supporting 128bit block size the initial value is a 96 bit string the counter is the remaining 32 bits to ensure the entire string is the same size as the block. The initial value is special in the sense that it may only be used once, a so-called nonce.

2.4.2.2 Galois MAC generation

The MAC is generated through a process of hashing and encrypting the already encrypted data and some additional data. The MAC is generated by using a function called GHASH and then encrypting the resulting output. GHASH works by multiplying the blocks of the data with the blocks of a hash key, then with an exclusive-OR operation add these blocks together. The hash key is a string of blocks the same size as the data that needs a MAC generated. After the GHASH function is complete the resulting block is then encrypted with AES-CTR to generate the message authentication tag. The resulting tag is 128 bits long but may be truncated down to seven different lengths. Five of the aforementioned lengths are 128, 120, 112, 104 and 96 bits long and do not require additional security concern. The remaining two are 64 and 32 bits long and require consideration of the length of input data and the lifetime of the hash key.

2.4.3 Secure tunnelling

Secure tunnelling is the practice of adding security to already existing protocols and communication systems by adding a protection layer. Security is a subjective term more easily defined by what it is not. A connection is not considered secure if it does not provide enough secrecy or protection as the user needs. Security in this project is defined as the protection of one drone with one controller from another controller. The two parts of this secure tunnel is encryption management and key management.

2.4.3.1 Encryption management

Encryption is what makes the tunnel secure. How encryption is implemented important. Data sent from either the Crazyflie or the Crazyradio is encrypted as the data is about to be sent. The encrypted package is given a special header and is then packed into a CRTP packet, that packet is put into an ESB packet.

The encryption makes the tunnel secure as specified. Due to the memory restriction, replay attacks are possible. To protect against replay attacks the packet IV needs to be stored. At four bytes the number of IV permutations are over 4 billion. The type of data structures that allow insertion and searching with a time complexity of one simply requires more space than is available on the drone.

2.4.3.2 Key management

Keeping the key secret is paramount to keeping the tunnel secure. The keys are not shared through communications. The keys are shared before any communication is made between the drone and the client. Specifically the key

the drone encrypts packages with is flashed to its firmware before use. The key on the client side however is available in cleartext directly in the library.

2.5 Cryptographic libraries and Python

This section goes into detail about the cryptographic libraries used for both the drone and the client.

2.5.1 WolfSSL

WolfSSL is a cryptographic library for the C programming language. It was developed by the WolfSSL company and written in ANSI C. ANSI C is a standard of the C programming language published by the American National Standards Institute. Due to it being a published standard it is supported by most of the modern C compilers.

The WolfSSL library is designed to be used in embedded devices. To allow this WolfSSL is able to have its memory footprint be reduced rather substantially. The minimal memory size is around 20Kbyte and runtime memory size is about 1Kbyte.

The cryptographic engine used in WolfSSL is the wolfCrypt library. A cryptographic engine is the part of the library that houses the code implementation of encryption algorithms. For example, The WolfSSL library has support for DTLS, DTLS is a communication protocol that uses the AES encryption algorithm. WolfCrypt contains the AES encryption algorithm and the WolfSSL library contains the necessary parts for DTLS to function.

Versions 3.6.0, 3.6.1, 3.6.6 of the wolfCrypt library have a FIPS 140-2 certificate. The FIPS 140-2 certificate is an indication of security level assessed by the CMVP. FIPS is the Federal Information Processing Standard and CMVP is the Cryptographic Module Validation Program, both are operated under NIST.[9]

2.5.2 Python

Python was created by Guido van Rossum as a successor of a language called ABC. It began development during december 1989 and its first version was published 1991. In this project Python 3.4.3 is used to send commands to the Crazyflie via the CrazyRadio using the CrazyRadio Library. Code example 1 is of Python code in section 4.3.1. [20]

2.5.3 Cryptography

Cryptography is a cryptographic library for the Python programming language. It is an open source project available on github with over 100 contributors. The repository is controlled by the Python Cryptographic Authority (PyCa). The functionality of Cryptography is more than adequate for the scope of the project.[16, 17]

3 Methodology

There were three phases in this project. These were the research and analysis phase, the prototyping phase and the implementation and testing phase.

The research and analysis phase has two parts namely gathering information and analysing the problem definition. Gathering information was done through reading the official documentation. Analysis was done as to determine the number and severity of problems to solve and how to do it.

The Prototyping phase consisted of setting up the development environment, adding the cryptographic libraries, setting up testable sub-systems on the drone and learning to use the cryptographic libraries.

The implementation and testing phase has two parts. The implementation phase was where the tunnel and encryption were implemented. During the testing of the software a few test programs were run.

3.1 Research and Analysis Phase

Firstly, the problems the project is supposed to solve have to be described. The main problem is hijacking, i.e., when an attacker takes control of a drone from the owner of the drone. In this case a USB radio is used to control the drone and another USB radio is trying to take control of the drone.

Before this project, the attacker could quite easily take control of the drone. To stop this the owner needs to stop any other radio from communicating with the drone and to do this an encryption algorithm will be introduced.

The chosen encryption algorithm needs to have a certain set of properties. Because the size of the packet is limited, the encryption algorithm's memory footprint needs to be small. The speed of encrypting and decrypting is important as the part of the drone that handles ciphering also handles the flight controls. The cipher needs to be cryptographically strong to ensure that the key or keys are not easily solved. The underlying radio communication protocol is asynchronous so this too had to be taken into account. DTLS was an early contender for cipher algorithm, however due to the software structure of the drone and USB-radio there was a lack of support systems requiered to have fully functional DTLS. The choice of cipher was AES-GCM as it had all of the prerequisite properties needed for this project.

How to best implement the encryption is an important question. Either there had to be a change to the entirety of the code base that handles incoming and

outgoing packets or there could be an added after each packet has been completed and is about to be sent where encryption can take place. Obviously a tunnel was chosen as the easiest and most modular way to implement encryption.

Communication with Bitcraze was done by face-to-face meetings or through the messaging and VoIP service Skype. As the Bitcraze offices are not in the same city as the thesis worker, most communication was done through Skype.

The environment in which the programming took place was using the virtual machine software VirtualBox. VirtualBox ran the official Bitcraze Virtual Machine version 2016-06-06. This virtual machine had almost everything needed to develop for the Crazyflie. The one thing it was lacking was the encryption libraries.

As previously mentioned, the easiest way to implement the encryption is to make a tunnel between the radio and the rest of the system. For the Crazyflie this means that an encryption “link” is made which can slot in for the radio link. Figure 9 gives a visual representation of the “link”.

For the Crazyradio however it is not quite that simple. Changes in the Python library have to be made in the specific send and receive functions. This has the drawback of not being as modular in the way of turning off or disabling the encryption for whatever reason but it does still leave the rest of the code the same as changes only needs to be done in one file.

3.2 Prototyping phase

Prototyping the possible solutions is the next step to make a complete solution. In this phase there were a lot of trial and error implementation of the different parts of the final solution.

The first step in prototyping was making a subsystem where messages could be sent and have the expected response. The subsystem made would simply take a packet with a certain header setup and respond with a fixed string at first. Later this subsystem was used for debugging and error responses as it was available and separate from any other system.

Due to the memory limitation on the drone the WolfSSL library needed to be scaled down to as small a memory footprint as it could possibly be. There was unfortunately no good way to determine the size of the library on the memory. The way of scaling down the size of WolfSSL is simply following the

documentation and realising what parts of the library that are important. The most important parts of WolfSSL were the AES-GCM functions.

3.3 Implementing phase

During the implementing phase the lessons from the previous two phases are put into action.

Maintaining the modularity of the code proved to be a significant problem when it came to the USB-radio code as the code structure was completely different. The structure of the code on the drone was so that communication was done through a variable link connected to the radio and then the rest of the system. The link in this case could consist of different types of communications channels such as USB or radio.

When the implementation of the tunnel was nearing completion testing began. Testing was initially done by simply starting the drone and seeing if it crashed. The drone has four LEDs. The LEDs show the status of the drone. There are a three types of crashes that are visible via the LEDs. If the Crazyflie does not boot, one blue LED shines continously. If the Crazyflie boots but boots incorrectly a red LED blinks 5 times fast. If the Crazyflie crashes during operation a LED will stop blinking. This test gives no real information on what went wrong but does tell when something goes wrong.

After the software stopped crashing testing was done by using a Python script developed by Bitcraze to ramp the drone rotors. The script is called Ramp.py and does exactly what it says, it ramps the engines up and then down once. This test was to ensure both sides could encrypt and decrypt the messages sent. The result of the script is a visual confirmation that the communication is received and produces a functional result.

Testing the drone in proximity to another drone was difficult as the number of drones was limited to one. Testing was done by using the android application to connect to the drone. The result of the android testing was that it was unable to act on any commands.

The code is available on Github and is linked to in the appendix.

3.4 Reference criticism

All references with the exception of reference [3] are official documentation and therefore can be considered reliable.

Reference [3] is the book Understanding Cryptography. Understanding Cryptography is written by Christof Paar and Jan Perlzl. The book is published by the Springer company. Christof Paar and Jan Perlzl are both Professors of the Ruhr university in Bochum, Germany.

“**Prof. Dr.-Ing. Christof Paa**r has the Chair for Embedded Security at the University of Bochum...” - Springer[22].
“**Prof. Dr.-Ing. January Pelzl** started his career at Bosch Telecom GmbH. He has a Ph.D. in applied cryptography...” - Springer[22].

Due to the expertise of the authors the book can be considered reliable.

4 Analysis

4.1 Requirements

The number of requirements for this project was rather small. This gave the developer a large amount of freedom to choose the best solution for any problem.

The project began with a need for protection against hijacking. This was quickly recognised as a need for encryption. The nature of the project dictates the requirements two and three below. The last two requirements were specified by Bitcraze to simplify the development for the Crazyradio. Making changes in the code for the firmware of the Crazyradio requires specialist equipment that is expensive and not easily available.

Requirements:

1. The communication between the Crazyflie and the Crazyradio must be encrypted.

2. The developed code must work on the Crazyflie drone.

3. The code developed for the drone must be written in the C programming language.

4. The developed code must work with the Crazyradio dongle.

5. The code developed for the client must be written in the Python programming language.

4.2 Choice motivation

The choice of cryptographic library is an important decision to make when encryption is the main focus of a project. Therefore the decision of cryptographic library is detailed in this section.

4.2.1 WolfSSL

There are three big contenders for cryptographic C libraries and those are Mbed TLS, WolfSSL and OpenSSL. As OpenSSL is not designed to be ported and work on embedded devices. Choosing the cryptographic library to use on the drone was essentially a choice between Mbed TLS and WolfSSL. Both were good choices with good forum support, documentation, and both were according to their documentation easy to port to new embedded hardware.

WolfSSL was chosen as a request by Bitcraze when both were given as viable and suitable options.

4.2.2 Cryptography

The number of choices for cryptographic libraries available for Python were unfortunately quite limited. The closest comparable to a stable and supported release was for the Cryptography library originally by the Python Cryptographic authority (PyCa). This github library has the largest number of contributers to a single cryptography library that had support for AES-GCM. The bigest drawback about open source projects where no official group has claimed responsibility is that there is no certificate of functionality or security to hold the responsible organisation to if it fails.

4.2.3 Client side encryption

Encryption on the client side could have been implemented differently. It could have been implemented in the firmware of the Crazyradio. This would have made the project larger in scope. The needed hardware and development environment are different from developing for the Crazyflie drone and client. This would have made the programming language for the client side C as well.

4.3 Substantial problems and their sollutions

4.3.1 Multi packet “packet”

ESB is the main limiting factor. This caused CRTP to have a limit. ESB is asynchronous so CRTP is aswell. Because encryption adds data to be sent a packet with 31 databytes has to be split into two packets instead. This forced the need for package ID. Package ID is used to combine two packets before decryption to ensure that everything is as it should.

```
try:
    self.out_queue.put(fp, True, 2)
except queue.Full:
    if self.link_error_callback:
        self.link_error_callback('RadioDriver: Could not send packet'
                                 ' to copter')
'''
If a original packet is too large the packet is split,
the second part is sent with the following code.
'''
if multipacket:
    fp.data = bytearray([pk.get_header()])
    fp.data += bytearray([pidbyte])
    fp.data += bytearray(ciphertext[dataLength:])
    try:
        self.out_queue.put(fp, True, 2)
    except queue.Full:
        if self.link_error_callback:
            self.link_error_callback('RadioDriver: Could not send packet'
                                     ' to copter')
```

Code example 1. This code example shows how the Crazyradio library code replaces all of the content of the packet with the second part of the encrypted packet.

5 Results

The results of this project was to implement a secure tunnel between the Crazyradio and the Crazyflie. This was accomplished by introducing encryption. The detailed description of how and where encryption was implemented is described in the following subchapters.

5.1 Ciphered packet structure

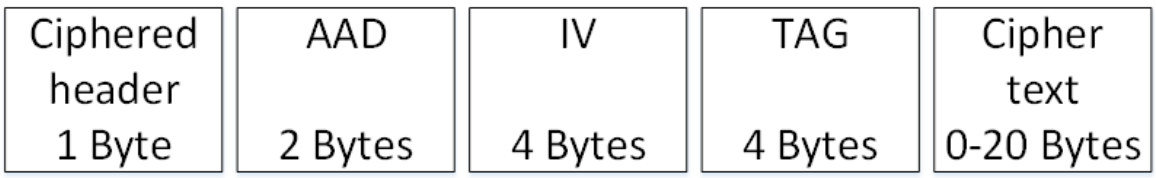

Figure 7. The Ciphered packet divided into its 5 segments.

The ciphered packet is separated into five parts. The first part is the header of one byte. The header byte is always the same and is the hex B3. The second part is the Additional Authenticated Data (AAD). The AAD contains the header of the unencrypted packet and a packet identifier(PID).The PID is a single byte where the first five bits is the length of the cipher text, the next two bits are a packet number and the last bit tells if the packet is a split packet. The third part contains the Initialisation Vector(IV). The IV is a specific number used for encryption and decryption with AES-GCM. The IV is a nonce. The fourth part contains the authentication tag. The authentication tag is generated by the AES-GCM algorithm. The fifth part contains the cipher text. The cipher text is the encrypted data.

5.1.1 Split packet

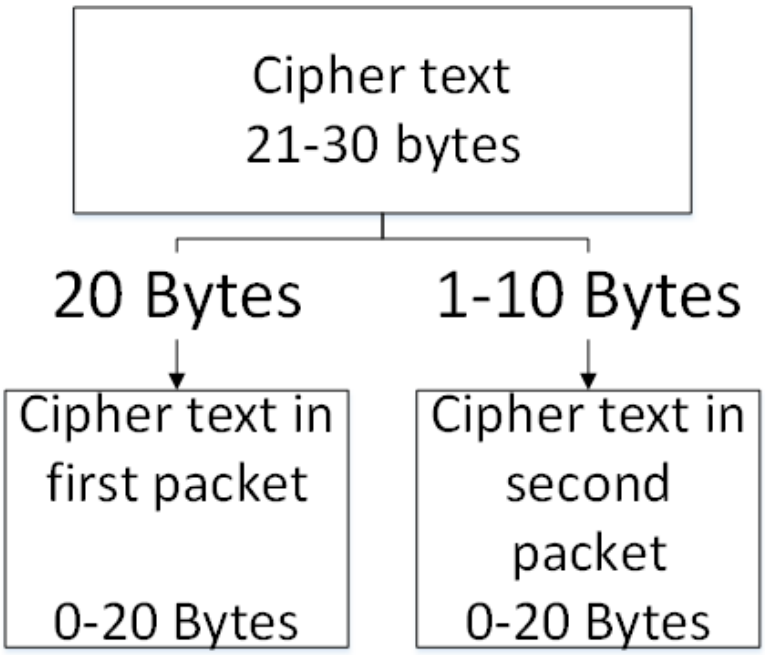

Figure 8. A cipher text is split into two packets.

When more than 20 bytes of data is in the original packet to be encrypted, the packet might have to be split into two packets. These two packets contain the same AAD, IV and tag. There is a change to the PID. As the PID contains one bit to tell if the packet has been split, that one bit simply flips to a one.

5.2 Drone

The drone side has a specific subsystem that is interchangeable with other methods of sending and receiving packets. This subsystem is called a link.

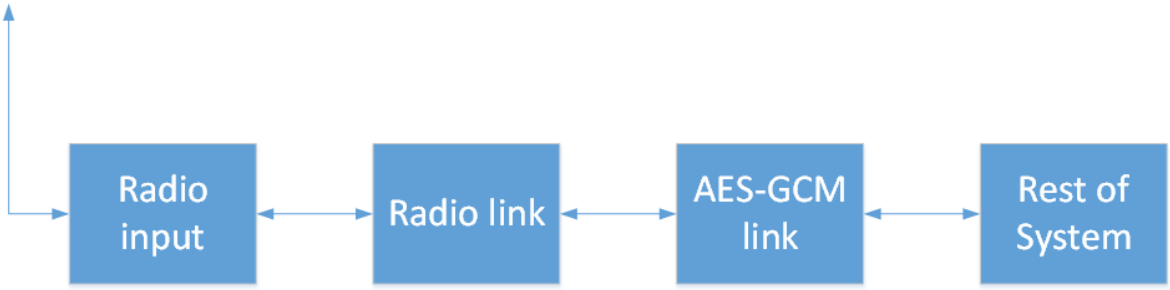

Figure 9. The flow of packets through the drone.

5.2.1 Receiving

The AES-GCM link receives a raw packet from the radio link, if the packet is a split packet it stores the IV, TAG, AAD, and data into buffers, waits for the next packet with the same PID number. If the next packet does not contain the

same PID number the new packet overwrites the first packet, if it does have the same PID number the new packet has its data added to the buffer of the first packet data. The IV, AAD, and TAG are then used to decrypt the data in a decrypt function from the WolfSSL library. The packet is then reencoded in to the unencrypted version with its previous header and unencrypted data and passed into the system via the CRTP service or a FreeRTOS qeue.

5.2.2 Sending

The AES-GCM link receives an unencrypted packet from the rest of the system. It seperates the header from the rest of the data. The header is added as additional data to the encryption process together with a PID that contains the multipacket-bit, the PID number and the length of the unencrypted data. The Initializing vector is produced. Using the IV, AAD and the data the encrypted packet is created. A ciphered packet header is added to the packet. The AAD, IV and TAG are added to the packet. If the length of the data was more than 20 bytes the first packet only contains the first 20 encrypted bytes. A second packet then gets created and has the two AD bytes first and then the remaining encrypted data bytes.

5.3 Radio and client

On the client side there is no specific link or subsystem, instead the encryption and decryption is in the send and receive functions of the Python radio interface.

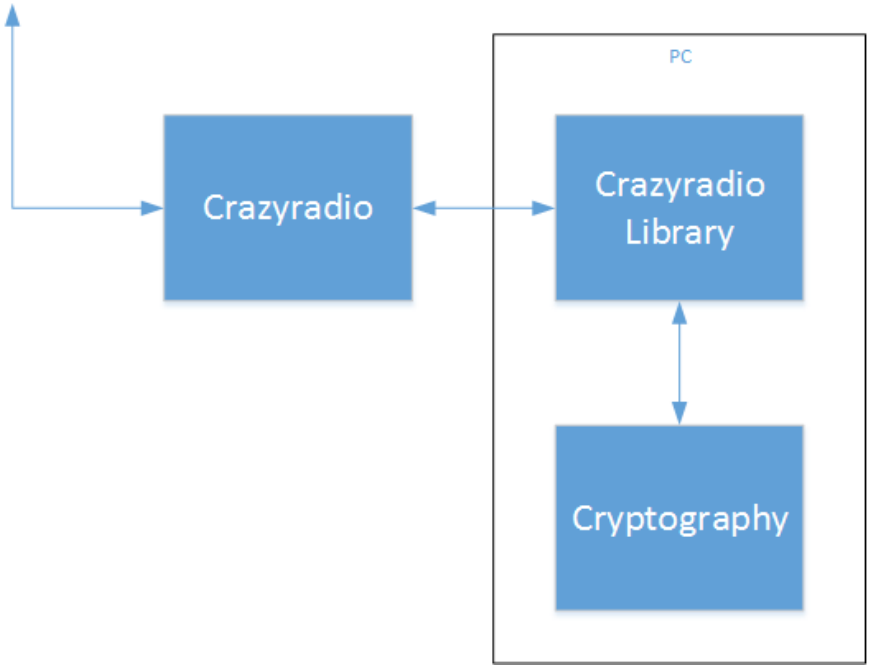

Figure 10. The flow of packets between the Crazyradio and the Crazyradio Library. Only the cipher text, IV, AAD and TAG are dealt with by Cryptography.

5.3.1 Receiving

When the Crazyradio Library receive function receives a packet it stores the header, the AAD bytes, IV bytes and TAG bytes in buffers. The data is put in a different buffer, if the received packet has more than 20 bytes of encrypted data there will be a second packet with the same PID. The second packet will add its encrypted data to the data buffer. A custom function using the Cryptography library will decrypt the data. Using the decrypted data a new packet with the first byte in the AAD is used as the header and the decrypted data as the payload.

5.3.2 Sending

When the Crazyradio Library send function receives a packet to send the data is separated from the header and put in a buffer. The header is used together with a generated PID to produce the additional data for AES-GCM encryption. A IV is randomised, the IV and the AAD are used to encrypt the data. The new packet gets a new header and its data buffer gets filled with AAD, IV, TAG and then the encrypted data. Depending on the size of the unencrypted data the packet can be split in two if it is more than 20 bytes of data.

6 Conclusion

This project was to minimise the possibility of hijacking the Crazyflie drone when flown in public spaces. This was done by adding encryption to the communication. The encryption added was AES using GCM mode.

The encryption was added in such a way as to make a secure tunnel. Both sides are encrypted before the packet is sent via the radio and decrypted before entering any packet management. This way there needed to be almost no change to how the rest of the drone functions.

AES was chosen as it is a highly secure cipher and can be made rather small. There was no AES implementation already on the drone or the client so this had to be added. They became available by adding cryptographic libraries to the programs. WolfSSL was chosen as it could configured to reduce its memory footprint to a small size. WolfSSL is a C library with FIPS 140-2 certificate and was deemed more than sufficiently cryptographically sound. For the client program the library Cryptography was chosen. Cryptography was chosen as it was deemed to be sufficiently secure.

6.1 Summary of the Problem Solution

1. What encryption algorithm is to be used to encrypt the communication?

The chosen encryption algorithm is AES using GCM.

2. What programming language will be used to implement encryption on the client?

As the client and Crazyradio library are both in python it was the logical choice to use python. The encryption could be implemented on the radio and is further explained in section 4.2.3.

3. What programming language will be used to implement encryption on the drone?

The firmware of the drone is written in C, therefore C was the language used add encryption to the drone. Alternative programming languages are not really applicable for the drone.

4. What cryptographic library will be used on the Crazyflie?

The cryptographic library used on the Crazyflie is the WolfSSL library.

5. What cryptographic library will be used on the client?

The cryptographic library used on the client is Cryptography library.

6. What does the secure tunnel protect against?

The secure tunnel protects against other clients trying to use a already controlled drone.

6.2 Ethical reflection

6.2.1 Secrecy or security through obscurity

The number of ethical implications of this project are minimal however there is one big question of Security through obscurity as a type of IT security. This project is open source and therefore has no wall for people to run into when doing security analysis.

Security through obscurity is the concept of having obscure or hidden source code and design of the software. This as a means of security is probably enough to dissuade any benign “attackers”. Any other type of “attacker” is more then likely not to either report the issue found or if it has any benefit for them use it to their advantage. The problem with this that it can give a false sense of security to both developers and users as the security holes will more than likely not be published or reported. Another real issue with the security through obscurity line of thought is that no program is perfect. There are always problems with every program to ever need any type of security.

6.3 Further development

The way AES GCM was implemented to fit a pre-existing protocol posed some challenges. The solution to those challenges was to shorten the tag and the initialisation vector. This is not a problem in and of itself if done thoroughly. Unfortunately there was one NIST recommendation out of scope for this project. Namely the changing of the hash-subkey at a frequent interval. This is controlled by the WolfSSL library and was therefore not in the hands of the developers of this project. This is a matter to look into however and therefore is fit to be further development.

As mentioned in the scope limit there are a list of more features that are desired. Making the client program able to change the encryption key and write it to the drone via USB is one of the future projects of the Crazyflie. Another perhaps more necessary than the previous one is to enable and disable encryption when using the Crazyradio library.

Future development for the Crazyflie hardware is in progress and is being actively discussed at Bitcraze. One of the changes that could be made is the size of the packets. The size of the packets could be larger to accommodate both the encryption and the data required for the secure communication between the drone and the Crazyradio.

Another hardware change on the drone could be adding more memory. This could allow the secure tunnel to store a history of received packets IV's and discriminate what packets are decrypted. A packet with an IV that has already been received could simply be discarded. However, with longer IV's there would need to be even more memory allocated to storing the history of received packets.

7 Terminology

AES – Advanced Encryption Standard. An encryption standard published by FIPS.

AAD – Additional Authenticated Data.

CRC – Cyclic Redundancy Check.

CRTP – Crazy Realtime Protocol.

DTLS – Datagram Transport Layer Security.

ESB – Enhanced Shockburst.

FIPS – Federal Information Processing Standards.

GCM – Galois Counter Mode, a mode of AES operation

GMAC – Galois Message Authentication Code

Hash – the result of a hash function.

Hash function – A function that maps any data with arbitrary size to data with a fixed size.

IV – Initialisation vector.

MAC – Message Authentication Code

NIST – National Institute of Standards and Tehcnology. An institute under the United States Department of Commerce.

RAM – Random Access Memory.

TLS – Transport Layer Security.

8 References

1. Announcing the Advanced Encryption Standard (AES).
   http://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.197.pdf
   Federal Information Processing Standards Publication 197. (2001).
   Federal Information Processing Standards.

2. Recomendation for Block Cipher Modes of Operation: Galois/Counter mode (GCM) and GMAC.
   http://nvlpubs.nist.gov/nistpubs/Legacy/SP/nistspecialpublication800-38d.pdf
   Morris Dworkin. (2007)
   National Institute of Standards and Technology.

3. Understanding cryptography.
   Christof Paar, Jan Pelzl. (2010).
   Springer.

4. About FreeRTOS.(26-09-2017)
   http://www.freertos.org/RTOS.html
   Real Time Engineers ltd. (2017).
   Real Time Engineers ltd.

5. nRF24LU1+ Single Chip 2.4 GHz Transceiver with USB Microcontroller and Flash Memory.
   https://www.nordicsemi.com/eng/Products/2.4GHz-RF/nRF24LU1P
   https://www.nordicsemi.com/eng/content/download/2724/34051/file/nRF24LU1P_Product_Spec_v1_1.pdf
   Nordic Semiconductor. (2010).
   Nordic Semiconductor.

6. nRF51822 Multiprotocol Bluetooth low energy/2.4 GHz RF System on Chip. Product Specification v3.3
   https://www.nordicsemi.com/eng/Products/Bluetooth-low-energy/nRF51822
   http://infocenter.nordicsemi.com/pdf/nRF51822_PS_v3.3.pdf
   Nordic Semiconductor. (2014).
   Nordic Semiconductor.

7. Intro to ShockBurst/Enhanced ShockBurst. (27-09-2017)
   https://devzone.nordicsemi.com/blogs/783/intro-to-shockburstenhanced-shockburst/

Daniel Veilleux. (2015).
Nordic Semiconductor.

8. ARM Cortex-M4 32b MCU+FPU, 210DMIPS, up yo 1MB Flash/192+4KB RAM, USB OTG HS/FS, Ethernet, 17TIMs, 3 ADCs, 15 comm. Interfaces & camera
http://www.st.com/en/microcontrollers/stm32f405rg.html
http://www.st.com/content/ccc/resource/technical/document/datasheet/ef/92/76/6d/bb/c2/4f/f7/DM00037051.pdf/files/DM00037051.pdf/jcr:content/translations/en.DM00037051.pdf
STMicroelectronics. (2016).
STMicroelectornics.

9. Validated FIPS 140-2 Cryptographic modules search engine certificate number 2425.(27-09-2017)
https://csrc.nist.gov/projects/cryptographic-module-validation-program/validated-modules/search
Due to the nature of the site searching for the certificate number is required to find the certificate.
National Institute of Standards and Technology.

10. Cryptographic module validation program.(27-09-2017)
http://csrc.nist.gov/groups/STM/cmvp/
National Institute of Standards and Technology.(2017)
National Institute of Standards and Technology.

11. Python Crazyradio Library.(27-09-2017)
https://wiki.bitcraze.io/projects:crazyradio:python_lib
Arnaud Taffanel. (2015).
Bitcraze.

12. Bitcraze Virtual Machine.(27-09-2017)
https://wiki.bitcraze.io/projects:virtualmachine:index
Arnaud Taffanel, Kristoffer Richardsson, Marcus Eliasson, Björn Mauritz, Tobias Antonsson. (2017).
Bitcraze.

13. Crazyflie 2.0 System Architecture.(27-09-2017)
https://wiki.bitcraze.io/projects:crazyflie2:architecture:index
Arnaud Taffanel. (2015).
Bitcraze.

14. Bitcraze.(27-09-2017)
https://www.bitcraze.io/about/
Bitcraze. (2017).
Bitcraze.

15. Crazy RealTime Protocol.(27-09-2017)
https://wiki.bitcraze.io/projects:crazyflie:crtp
Arnaud Taffanel, Marcus Eliasson. (2017).
Bitcraze.

16. Cryptography.(27-09-2017)
https://github.com/pyca/cryptography
Each of the 136 individual contributors. (2017)
Github.com

17. pyca/cryptography.(27-09-2017)
https://cryptography.io/en/latest/
Each of the 136 individual contributors. (2017).
Cryptography.io

18. WolfSSL Porting Guide.(27-09-2017)
https://www.wolfssl.com/wolfSSL/Docs-wolfssl-porting-guide.html
WolfSSL. (2017).
WolfSSL.

19. WolfSSL (formerly CyaSSL) Manual.(27-09-2017)
https://www.wolfssl.com/wolfSSL/Docs-wolfssl-manual-toc.html
WolfSSL. (2017)
WolfSSL.

20. General Python FAQ.(27-09-2017)
https://docs.python.org/3/faq/general.html#why-was-python-created-in-the-first-place
Python Software Foundation. (2017)
Python Software Foundation.

21. Ultra-low Power Wireless System On Chip Solution.
http://infocenter.nordicsemi.com/pdf/nRF24LE1_PS_v1.6.pdf
Nordic Semiconductor.(2010)
Nordic Semiconductor.

22. Understanding Cryptography – A textbook for students and | Christof Paar | Springer (01-10-2017)
https://www.springer.com/us/book/9783642041006

Springer.(2017)
Springer.

9 Appendicies

The code of the project is available on github through two repositories. Both repositories are linked here.

https://github.com/Lursidon/crazyflie-firmware
https://github.com/Lursidon/crazyflie-lib-python