VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

Fakulta elektrotechniky
a komunikačních technologií

SEMESTRÁLNÍ PRÁCE

Brno, 2019 Bc. Roman Ligocki

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ
FACULTY OF ELECTRICAL ENGINEERING AND COMMUNICATION

ÚSTAV TELEKOMUNIKACÍ
DEPARTMENT OF TELECOMMUNICATIONS

ZABEZPEČENÁ KOMUNIKACE V RÁMCI PLATFORMY PX4

SEMESTRÁLNÍ PRÁCE
SEMESTRAL THESIS

AUTOR PRÁCE — **Bc. Roman Ligocki**
AUTHOR

VEDOUCÍ PRÁCE — **doc. Ing. Petr Číka, Ph.D.**
SUPERVISOR

BRNO 2019

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ

Semestrální práce

magisterský navazující studijní obor **Informační bezpečnost**
Ústav telekomunikací

Student: Bc. Roman Ligocki ***ID:*** 169280
Ročník: 2 ***Akademický rok:*** 2019/20

NÁZEV TÉMATU:

Zabezpečená komunikace v rámci platformy PX4

POKYNY PRO VYPRACOVÁNÍ:

Nastudujte problematiku řízení dronů s úzkým zaměřením na platformu PX4 i protokol Mavlink. Proveďte kompletní analýzu zabezpečení komunikace společně s vektorizací útoků a návrhem zabezpečení jednotlivých částí komunikace. Návrh bude implementován a bude otestováno zabezpečení komunikace. Následně proběhne fyzické ověření bezpečnosti implementace pomocí testovacích scénářů. Výsledkem bude řešení poskytující zabezpečený přenos mezi řídicí jednotkou a dronem. Důraz bude kladen na efektivitu, nenáročnost a nízkou spotřebu.
V rámci semestrální práce proběhne potřebná analýza, dále bude vytvořena vektorizace útoků a návrh zabezpečení. Proběhne zprovoznění komunikace PX4/Mavlink a prvotní ověření funkčnosti jednotlivých prvků a měření náročnosti dostupných algoritmů (již implementovaných v rámci platformy).

DOPORUČENÁ LITERATURA:

[1] PETROVSKY, Oleg. Attack on the drones. In: Virus Bulletin Conference. 2015.

[2] ALLOUCH, Azza, et al. MAVSec: Securing the MAVLink Protocol for Ardupilot/PX4 Unmanned Aerial Systems. arXiv preprint arXiv:1905.00265, 2019.

Termín zadání: 23.9.2019 ***Termín odevzdání:*** 21.12.2019

Vedoucí práce: doc. Ing. Petr Číka, Ph.D.
Konzultant: Ing. Radek Fujdiak, Ph.D.

prof. Ing. Jiří Mišurec, CSc.
předseda oborové rady

DECLARATION

I declare that I have written the semestral project titled "Zabezpečená komunikace v rámci platformy PX4" independently, under the guidance of the advisor and using exclusively the technical references and other sources of information cited in the project and listed in the comprehensive bibliography at the end of the project.

As the author I furthermore declare that, with respect to the creation of this semestral project, I have not infringed any copyright or violated anyone's personal and/or ownership rights. In this context, I am fully aware of the consequences of breaking Regulation § 11 of the Copyright Act No. 121/2000 Coll. of the Czech Republic, as amended, and of any breach of rights related to intellectual property or introduced within amendments to relevant Acts such as the Intellectual Property Act or the Criminal Code, Act No. 40/2009 Coll., Section 2, Head VI, Part 4.

Brno ....

author's signature

Contents

List of Figures

Introduction

At this time it is very simple to build your unmanned vehicle. Components like autopilot boards, RC transmitters, ESCs, motors, batteries or propellers are available nearly for anyone. There is also a huge amount of open-source software platforms available to control that hardware. It takes only time to learn and to get enough information to find out which hardware you need and how to connect it. It means, that nearly anyone can have his own unmanned vehicle. After buying and connecting all hardware in a few hours it possible to have a functional unmanned vehicle. It could be a plane, multi-copter, VTOL plane or some kind of tracked or wheel vehicle. That means, that having drone is no longer the privilege of some research institutions or companies and we can expect, that numbers of DIY drones will be rising. An increasing number of flying unmanned aerial vehicles could be interesting for hackers, that might want to get control over those vehicles. After taking control of the aerial vehicle, it could be a potential danger to all around. Also, those vehicles could have a very expensive or important payload, that might be stolen or destroyed. Example of important payload could be blood packages. Nowadays Zipline company delivers different types of blood packages in Rwanda, where blood distribution using aerial vehicles is more secure, reliable and faster. To make those delivery systems less possible to be hacked, unmanned systems should use communication protocol with strong confidentiality and integrity of traffic, or at least use a secure communication channel. In this thesis, a goal is to do research of the PX4 platform and find as many vulnerabilities as possible in telemetry communication protocol. PX4 platform uses MAVLink communication protocol, to exchange information like an altitude, direction of flight, rotation in all axis or mission plans. This possibility brings vulnerabilities if a communication channel is unsecured. MAVLink protocol will be described in details and its vulnerabilities will be analyzed. At the end of this thesis, an idea of security implementation will be shown. This design should eliminate all possibilities, to take control of the vehicle by an unauthorized person, get some confidential information or to make the vehicle dangerous for its surroundings. Final design will be implemented into MAVLink and PX4 public repositories, to share solution with community around PX4 and make unmanned vehicles all around the world more secure.

1 PX4 Software stack

1.1 Overview

PX4 is open-source community-based autopilot platform, founded by Lorenz Meier in 2008. History of PX4 is described in [1] article. As article describes at this time Lorenz was a master's degree student, that wants to create *Unmanned Aerial Vehicle* (UAV) with autonomous flight feature base on computer vision. Today PX4 is an industry-leading platform, that allows companies to create their UAVs for commercial usage. PX4 and compatible hardware are also priced affordable for the hobbyist. This brings a easy way to construct UAV from cheap hardware powered by these open-source software stack. Nearly anyone with knowledge of computer science and electronics can build quad-copter that can have any kind of payload. To build and configure this UAV, it is possible to use PX4 user guide [2]. Inside this documentation, there is huge amount of tutorials to configure whole UAV. Based on user guide [2] one of the biggest advantages of this platform is that it allows developers and users to create traditional multi-copters, planes, rovers *Vertical Takeoff and Land* (VTOL) planes, but also there is possible to define custom frame design using few lines of code in actuator mixer. All these vehicles can carry lots of different payloads where we can find for example cameras for image capture, video recording, but also Lidars and multispectral cameras. This might create opportunities for new businesses to use these type of drones for precise agriculture, geodesy, during search and rescue operations or for military usage. PX4 is not the only open-source platform for unmanned systems. There is also an ArduPilot project, that was created in 2009. Today ArduPilot together with PX4 is one of most used platforms for a lot of different types of vehicles. ArduPilot has also good user documentation [3]. As both documentations [2][3] describes PX4 and ArduPilot and PX4 use a MAVLink protocol to create missions or provide telemetry data to the operator. Because of that, it is possible to plan missions or control both platforms from QGroundControl software. At the beginning of this thesis, it is important to describe one thing. A lot of peoples uses drone word in the wrong way. Based on article [4], all flying machines without pilots are shortly UAVs. The only UAVs that are able to fly autonomously can be called drones. In this thesis, I will be talking mainly about UAVs or *Unamenned Ground Vehicle* (UGV), and only when a vehicle will be able to do missions autonomously, then it will be called a drone. Whole PX4 Flight stack, that includes autopilot firmware is licensed under BSD 3-clause licence [5]. This licence is one of the freest licences that might be used. It allows users and programmers to copy code with only one limitation. There is no limitation at numbers of software copies, but also there is no limitation in future development, that includes modifying source

code. There is also no need to share source codes like with GNU *General Public License* (GPL) licence. This means that this licence is very good for proprietary forks of open-sourced software.

1.2 PX4 autopilot architecture

PX4 autopilot firmware is one of the most important parts of the whole PX4 platform. This piece of software is something like a pilot, that watch all sensors values and listen to commands. Based on that data, it generates outputs to all actuator to control the unmanned vehicle. In developers guide [6] there is description of autopilots components. Autopilot is based on two layers: Flight stack and Middleware. Flight stack is doing all sensor and user input fusion, estimations and flight control. Middleware creates a layer that handles all internal and external communications but also provides drivers for supported hardware. Flight stack layer is a collection of all estimation and control algorithms. There are controllers for all different kinds of vehicles like rovers, multi-rotors, helicopters and VTOL. Using these controllers, it is possible to maintain a position and attitude of the vehicle. Flight stack diagram is shown on fig **??**

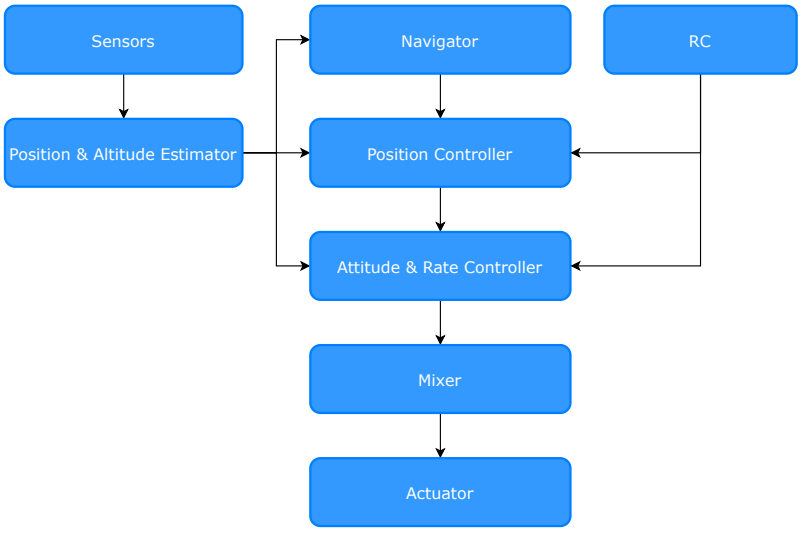


Fig. 1.1: PX4 flight stack diagram [6]

Middleware layer provides all drivers for supported embedded sensors and communication with them. It also provides communication tools for all peripherals like *Global Position System* (GPS), external magnetometer or companion computer. To make simple communication between different parts of autopilot firmware, Middleware has implemented uORB message bus, that works on the public-subscription system. This provide solution do distribute data where they are needed.

1.3 QGroundControl software

To set up and control the unmanned system based on PX4 or ArduPilot, the user needs to have some software, that is able to communicate with that system. These types of software are called ***GCS!*** (**GCS!**). In case PX4 and ArduPilot, there is a software called QGroundControl. It is multi-platform mission planner, that allows users to set up their vehicles, change configurations and also provide a way to plan missions. Also, it is possible to view telemetry data and the actual position or status of the plane. All information about QGroundControl are available on user guide website [7]. The main advantage of QGroundControl in comparison to other mission planners is that it runs on nearly every modern platform like Windows, Linux, macOS, Android but also on iOS and iPadOS. This is possible because QGroundControl is based on the Qt framework (Qt for application development), that brings Qt *Application Programming Interface* (API) and makes final code runnable on nearly any device. Software that is build using Qt framework have only to options in case of licence. Qt provides commercial and open-source licenses. For example, if a company wants to develop proprietary software, it needs to pay for every software developer, that works on that software. If the company don't want to pay, that it needs to provide source codes of their software because software must be licensed under GPL or *Lesser General Public License version 3* (LGPLv3).

2 PX4 Hardware

A main and most important part of PX4 platform is Flight controller. It is *Printed Circuit Board* (PCB) with all sensors and connectors necessary to fly and control UAV. Mostly used Pixhawk 1 is based on *Flight Management Unit version 2* (FMUv2) open hardware design. Next versions of FMU have newer sensors, more storage for flight firmware, sensors redundancy and more computation performance. All these improvements add new possibilities for hobbyist and commercial usage. There are also different FMUs, that are supported by PX4, but their hardware designs are not open-sourced. Information about all supported hardware are available in developer guide [8].

Manufacturers of PX4 flight controllers:

- 3DR (Pixhawk 1),
- mRobotics,
- HobbyKing,
- Holybro,
- Drotek,
- Hex (Pixhawk "Cube").

To control actuators like servos or *Electronic Speed Control* (ESC)s, every PX4 autopilot board must have some kind of outputs. On Pixhawk board series there are two ways to output actuators: using digital signal or over communication bus. Mostly used digital signal is *Pulse Width Modulation* (PWM), that are options like PWM signal, S.Bus or it is also possible to *Universal Asynchronous Receiver-Transmitter* (UART) or *Controller Area Network* (CAN) buses. Most used is PWM signal, that is very easy to use, but also there are dozens of servos or ESCs compatible with PWM signal. There are also modified versions of PWM, that offers higher precision and additional signal robustness. All implementations during this thesis will be tested on Pixhawk "Cube" and Pixhawk 1 boards, that will be described in more detail in the next section.

2.1 Pixhawk 1

Pixhawk 1 is one of the most commonly used autopilot board for PX4 or ArduPilot platforms based on FMUv2 hardware design. With price about 129 *United States Dollar* (USD) for original Pixhawk 1 manufactured in the United States of America or one of many clones that costs about 50 USD, it is one of the most used autopilot hardware on the market. It has all the necessary sensors, that UAV needs to control the attitude and altitude of the vehicle. This board doesn't have any communication hardware or GPS receiver onboard. To connect peripherals like GPS receiver, *Radio*

Controlled (RC) links or telemetry link, there are DF13 connectors, that are widely used in UAV industry. This board is targeted for hobby usage. It doesn't have features like *Inertial Measurement Unit* (IMU) temperature stabilization or IMU redundancy. All provided information are based on 3DR Pixhawk hardware manual [9].

Hardware specification:

- 32-bit STM32F427 Cortex M4 core with FPU,
- 168 MHz/256 KB RAM/2 MB Flash,
- 32-bit STM32F103 failsafe co-processor.

Pixhawk 1 IMU specification:

- 3-axis 16-bit gyroscope L3GD20,
- 3-axis 14-bit accelerometer/ magnetometer LSM303D,
- Invensense MPU 6000 3-axis accelerometer/gyroscope,
- MEAS MS5611 barometer.

Pixhawk 1 IO specification:

- 1x I2C (separate connectors),
- 2x CAN: CAN1 and CAN2,
- 5x UART: TELEM1, TELEM2, GPS, SERIAL4, SERIAL5,
- 1x HMI: USB extender,
- 14xPWM output,
- 1x RC input,
- 1x RSSI input.

Fig. 2.1: Pixhawk 1 autopilot board [9]

2.2 Pixhawk 2.1 - Cube

Pixhawk 2.1 also known as "Cube" is more advanced autopilot, that is widely used in industrial and commercial systems. Name "Cube" is based on two-part design, where one part is carrier board, that provides all wiring for autopilot and second part

is CPU board with IMUs installed in small cubic shape enclosure with integrated heater and vibration isolation. The second part with CPU and IMUs is removable. With features like triple-redundant IMU, vibration stabilization and temperature stabilization, this board is very good for commercial and industrial usage. This board also provide more RAM and Flash storage, that allows creating more complex control and estimation algorithms. This board comes with a price of 238 USD per set. All provided information are based on Hex Pixhawk 2.1 hardware manual [10] and [11].

Hardware specification:

- 32bit STM32F427 Cortex-M4F® core with FPU,
- 168 MHz / 252 MIPS,
- 256 KB RAM,
- 2 MB Flash (fully accessible),
- 32 bit STM32F103 failsafe co-processor.

IMU specification:

- Onboard fixed IMU,
  - 3-axis gyroscope / accelerometer MPU9250,
  - barometer MS5611,
- Two vibration isolated and heated IMU.
  - 3-axis accelerometer/magnetometer LSM303D,
  - 3-axis gyroscope L3GD20,
  - 3-axis gyroscope / accelerometer MPU9250 or ICM 20xxx,
  - barometer MS5611.

IO specification:

- 2x I2C,
- 2x CAN: CAN1 and CAN2,
- 5x UART: TELEM1, TELEM2, GPS (I2C 1 embedded), SERIAL4(I2C 2 embedded), SERIAL5,
- 1x HMI: USB extender,
- 14x PWM output,
- 1x RC input,
- 1x RSSI input.

Fig. 2.2: Pixhawk 2.1 autopilot board [10]

2.3 Hardware setups

In this section there will descriptions of few hardware setups, that are used by hobbyists or professionals. All of them contains autopilot that is necessary to run PX4, telemetry radio set for MAVLink protocol communication and RC radio to control rover. RC radio is commonly secured using proprietary communication system created by manufacturer on transmitter and receiver. Because of this, I will describe in more details only MAVLink connection systems and their security issues.

Setup #1 - Hobbyist:

- Rover hardware
  - Pixhawk 2.4.8
  - SiK telemetry radio - rover
  - FlySky FS-iA6 - receiver
- GCS hardware
  - Windows, Linux or MacOS computer
  - Sik telemetry radio – ground
  - FlySky FS-i6 2.4G 6CH – transmitter

Setup security issues:

- No authentication
- No encryption
- No access control

Setup #2 - Hobbyist:

- Rover hardware
  - Pixhawk 2.4.8
  - Wifi telemetry radio
  - Frsky X8R receiver
- GCS hardware
  - Android tablet

  - Frsky Taranis X9D Plus

Setup security issues:

- Encryption is optional (WPA2)
- No authentication
- No access control

Setup #3 - Professional:

- Rover hardware
  - Pixhawk 2.1 Cube
  - RFD868x - air
  - Frsky X8R receiver
- GCS hardware
  - iPad air (3rd generation)
  - Frsky Horus X10S
  - RFD868 TXMOD

Setup security issues:

- Encryption is optional (WPA2)
- No authentication
- No access control

Setup #4 - Professional:

- Rover hardware
  - Pixhawk 4
  - Herelink - air
- GCS hardware
  - Herelink – ground station

Setup security issues:

- No information are provided about security

2.4 PX4 compatible radios

At this section, radios mentioned earlier will be introduced. They are working on a variety of frequencies. If some device will be "legal", then it is meant, that this device is compatible with regulations only in the Czech Republic. Other countries were not part of the research. Regulation for free frequencies are described in document released by *Czech Telecommunication Office* (CTO)[12].

2.4.1 SiK telemetry radio

SiK telemetry radios are devices based on open-source SiK firmware and cheap SiLabs S1000 *System on a Chip* (SoC)[13]. As an example, Holybro telemetry radio

V3 433MHz will describe in more details Holybro telemetry radio. This telemetry set consists of two devices. Both of them have micro-USB connector for PC or tablet connection and 6pin JST-GH connector to connect radio into Pixhawk 2.4.8 or another autopilot. With the included antenna and with default configuration it is possible to control the drone at a 300m distance. With default configured output power, that is 100mW, so it is not legal to use this device in the Czech Republic.

2.4.2 WIFI telemetry radio

This telemetry device is using *Wireless Local Area Network* (WLAN) to create a network where it broadcasts MAVLink packets encapsulated into TCP/IP. Setup is very easy because on the ground station side there is no need to have any other device except the computer with Wi-Fi. This solution is compatible with any Mac, Linux or Windows computer. Also, it is possible to use an Android tablet or iPad. With output power 100mW it is legal to use this telemetry device in the Czech Republic. If *Wi-Fi Protected Access 2* (WPA2) is enabled, all telemetry is encrypted using the pre-shared key with encryption algorithm AES-CCMP

2.4.3 RFD868 combo

RFD868 is one of the best solutions to communicate with unmanned systems[14]. It provides very reliable and long-range telemetry link. It is possible to create telemetry link up to 80km using pitch antennas. This radios also have the feature to passthrough *Pulse Phase Modulation* (PPM) signal to control the drone. All RFD868 radio sets allow to encrypt all MAVLink and PPM data using *Advanced Encryption Standard* (AES) encryption algorithm. There is also TXMOD package, that might be connected to JR socket in Frsky's radios. This TXMOD package provides Wi-Fi access point, that creates WIFI network all-around and any Wi-Fi compatible device might connect to this network and receive MAVLink communication. In the Czech Republic is it legal to use this device with output power up to 500mW, but only on one channel (869,4 - 869,65MHz).

2.4.4 Herelink HD Video transmission system

Herelink is all-in-one solution, that provides long range link (up to 16km) for telemetry, PPM signal but also video link [15]. Herelink ground station is android tablet, that has two 2-axis joystick, wheel and six buttons to control QGroundControl software. Radio link is integrated and there is no need to buy additional hardware to control unmanned system. On air unit, there are two HDMI input ports, where you can insert output from any HDMI camera. For *First Person View* (FPV) video

you need to use A/D converter. Currently QGroundControl is only GCS software, that is available for Herelink, but there is no option to customize QGroundControl, because modified version for Herelink is not for now open-source.

2.4.5 RFD868 in Multi-point mode

As it is described in manual [14], RFD868 modem can create a very reliable and fast connection between two modems. It has also feature to create a multi-point network or asynchronous non-hopping mesh. Difference between these two modes are, that multi-point networks allow to communicate with a node that is not in range from the main station. In that case, communication is retransmitted using a node that is available in a range of the main station. In asynchronous non-hopping mesh mode, it is possible to communicate only with multiple nodes within range of the main station.

2.4.6 Conclusion

It was described in this section, there are some setups where it is possible to encrypt whole communication. But some of them doesn't provide any encryption for MAVLink messages. So, if someone wants to have at least encrypted communication, then there are some solutions. Problem is that there is no hardware set up where one side might authenticate if messages come from a trusted source or have an access control system that would allow creating users with different privileges. Because of this reason, it was decided to create in this thesis implementation of encryption, authentication and access control.

3 MAVLink protocol

MAVLink (Micro Air Vehicle link) is a very lightweight communication protocol to communicate with unmanned systems or with other devices on board. For example, using MAVLink protocol you can upload flight mission, get all flight data or change parameters of the unmanned system. MAVLink is based on modern hybrid publish-subscribe or point-to-point design pattern. All data streams are sent/published as the topic, while configuration sub-protocols such as the mission protocol or parameter protocol are point-to-point with retransmission. As source of information about MAVLink protocol a MAVLink website has been used [16]. This protocol is lightweight because there is very small overhead per packet. MAVLink 1.0 has only 8 bytes of a header for every message. In MAVLink 2.0, there are 14 bytes overhead, but provides more security and is more extensible. All messages are defined using *Extensible Markup Language* (XML) in multiple dialects, where every message has its definition. Dialects allow creating different sets of messages for different types of systems that use MAVLink protocol as a communication tool. Mostly used dialect is "common.xml". Example of the message written in XML format is possible to see in appendix A.

3.1 MAVLink 1.0

MAVLink 1.0 was first released in 2009 by Lorenz Meier. His goal was to create very reliable communication protocol for varied vehicles, communication environments (radios with low bandwidth or high latency/noise channels). It also provides detection system for lost packet and corrupted packets. MAVLink 1.0 need very small amount of management. Only 8 bytes are needed to create packet with no payload. Nowadays MAVLink is ported for many different programming and interpreting languages like C, C++, Python, Java and Swift and also is running on many platforms (ARMv7, ATMega, dsPic, STM32, Windows, MacOS, Linux).

Header description:

- STX – Start byte of value 0xFD,
- LEN – Number of bytes saved in PAYLOAD part of packet,
- SEQ - Sequence number of packet. Provides way to detect packet loss,
- SYS ID – System ID of sending device. Used to address device in network,
- COMP ID – Component ID of sending device. Indicates type of device in network,
- MSG ID – Message ID. Needed for payload deserialization,
- PAYLOAD – Message serialized into array of bytes,
- CHECKSUM – Cyclic redundancy check. Used for corruption detection.

3.2 MAVLink 2.0

Compared to the older version of MAVLink, version 2.0 provides new features like compatibility and incompatibility flags, extended message ID. These features allow to create more types of messages and also provide flags, that indicates if packet should be handled in different way than normal packet. Also, MAVLink 2.0 adds packet signing, that provides basic authentication. Next feature is empty-bytes payload truncation, that removes last zero bytes from serialized payload. Using this technique, MAVLink 2.0 lower amount of payload bytes send over channel. MAVLink 2.0 is still not ported into all programming languages, that has MAVLink 1.0 available.

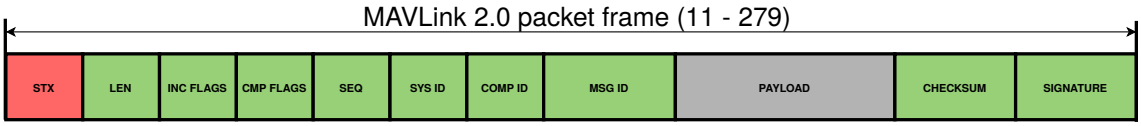


Fig. 3.1: MAVLink 2.0 packet frame [16]

Compared to MAVLink 1.0 in MAVLink 2.0 there were added new bytes into the header to allow new features.

- STX byte was changed from 0xFD to 0xFE.
- INC (Incompatibility) FLAGS byte was added.
- CMP (Compatibility) FLAGS byte was added.
- MSG ID was enlarged from one byte into three bytes.
- SIGNATURE array was added.

3.2.1 Packet signature

MAVLink 2.0 adds message signing feature, that allows authenticating if the message comes from a reliable source. To do this SIGNATURE were added, where a sign is stored. This creates security layers where attackers suppositious messages are not accepted on the receiver side because there is no way to create a valid signature for a fake message without the pre-shared secret key.

SIGNATURE has three parts:

- LinkID – ID of link on which message is send
- Signature – Six-byte signature computed using SHA256 algorithm
- Timestamp – Six-byte number with units of 10 microseconds since 1st January 2015 GMT

Signature works on *Keyed-Hashing for Message Authentication* (HMAC) system and computed based on *Secure Hash Algorithm* (SHA)256 algorithm. Commonly SHA256 hash algorithm return array of bits with the length of 256. To lower length of signed message, signature of MAVLink is shorten to 48 bits. In case of security shorter signature doesn't lose signature strength. There is a very low probability ($1/2^{48}$ bit), to find a signature that will suit some message. Formula how signature is computed can be seen on formula 3.1.

$$Sig = 256to48bits(SHA256(secKey{+}header{+}payload{+}CRC{+}linkID{+}timestamp)) \tag{3.1}$$

Signed message is accepted only under these conditions:

- Timestamp is older than timestamp of previous received packet form same device.
- Computed signature doesn't match with signature from received message.
- Timestamp is older than 1 minute.

3.3 MAVLink 2.0 security issues

3.3.1 Message signing

MAVLink protocol provides a very good way to transport a huge amount of data with very low overhead but provides only very basic security features. In current state MAVLink only provides a simple authentication system based on HMAC. This means that drone manager must add the same symmetric key into all devices of network. This creates a security issue, that if one of the devices and its symmetric key are compromised, the whole authentication system is not reliable. Then messages might be suppositious and there is no way to find it out.

3.3.2 Message encryption

Next disadvantage of MAVLink protocol is lack of encryption. All data that might be sent over radio are sent in plain text. This allows an attacker to read all messages and to know where exactly vehicle or drone might be and what it does. Also, it is possible to find out what plans the pilot might have. In hobby or commercial industrial, this might not be a huge problem, but in the case of military or police usage, mission plans and information might be classified. This brings the idea to add encryption into MAVLink communication and provide confidentiality to all

classified data, but that brings new problem. Just like in the case of a signature system, symmetric encryption needs a key, that must be distributed to all devices in the network. This brings the same vulnerability as the signature system, where disaffection of any device gives the key to all encrypted messages. To eliminate this distribution problem and to create a key exchange system, there is a solution to add private key infrastructure into the MAVLink protocol.

3.3.3 Key exchange

If there are only two devices in the MAVLink network, then encryption and signing keys distribution are very simple. If the key is changed then, for example, there are only two keys to change. In-ground control station and in the drone. In the case of 100 drone fleet, the operator will need in case of key reseeding to change the new key in all devices. This would be very time consuming and not very flexible solution. To solve this problem there is a private key infrastructure system with its key exchange system for all devices network. This means that every device must have its asymmetric key pair for key exchange and its certificate, that contains information about device, maintainer and public key. This certificate must be signed by authority, that is trusted by all devices in the network. Using this certificate, it is possible to verify other devices.

3.3.4 Access control

MAVLink lacks also from the access control system, that will allow creating groups of devices with different access rights. As an example, there is a military group of soldiers that has a fleet of drones to operate. In MAVLink network there are 5 devices: Ground control station handled by the main pilot, monitor to control and view cameras. There are also 3 UAVs, that fly different missions. The main pilot handles mission planning. The secondary pilot that has monitor controls cameras. In that case, there should be an access control system to separate what each device can do with other devices. It is undesirable to be able to control UAV from the monitor, that is handled by the non-authorized person and vice versa It is undesirable to control the camera and watch output video when the pilot should be focused on UAVs control. Also, a device like UAV should not have any rights to generate any MAVLink messages, that might change mission plans or steer other UAVs. This example shows that MAVLink needs to be able to provide different levels of access rights for any device in network based on operator fleet architecture.

4 Cryptography libraries

To add encryption, authentication and access control system, there was a need to find C library with cryptography primitives, that will be easy to use and allow systems with lower performance to be compatible with other devices. There is a huge number of open-source libraries, that are designed for 32-bit processors. For example, that library would be good for PX4 autopilot based on Pixhawk 1, that uses a 32-bit processor. Problem is that same library will not work with 64bit Intel or AMD processors. To remove this problem there was a need to find multi-platform cryptography library, that would be compiled on any device used by PX4. These types of devices must have enough performance to do all main tasks, but also, they need to have enough performance to add security layer into communication. So, there will be a search to find the encryption library with all necessary functions to implement all security features described in previous chapters. Also, the library must work on any device that might be part of the setup. In this chapter, there will be two libraries described in more details to find out which will most suit PX4 platform.

4.1 LibHydrogen

LibHydrogen is cryptography library inspired by LibSodium library. It aims to be easy to use and to have good performance any supported architecture. It has implemented only two necessary cryptography primitives to have all necessary functions, but also to lower its size as possible. For key exchange there is Curve25519 algorithm. For hashing and encryption there is Gimli algorithm. User manual is available on GitHub of LibHydrogen library [17].

4.2 MonoCypher

MonoCypher is next C library, that is mainly designed to be as simple as possible to use with very small footprint. With nearly 2500 lines of code it provides all necessary algorithms to have authenticated encryption, key exchange and signing system, but also hash functions. User manual is available on MonoCypher website [18].

Both libraries are very similar, but still they have some differences. In this section there will be in details described differences between both libraries.

4.2.1 Features

Both libraries implements:

- Authenticated encryption,
- Hashing,
- Password key derivation,
- Key exchange,
- Public key signature,

In additional LibHydrogen has implemented true random number generation for many different platforms (Windows, Linux, Mac, AVR, *Advanced RISC Machine* (ARM)). Most important difference in both libraries in case of PX4 security implementation is key exchange system.

LibHydrogen offers three schemes to safely exchange:

- N key exchange – At this variant, only one side needs to know the public key of another side. The session key is generated also on one side. When the session key is encrypted using the public key from the other side it is sent to the other side, where it is decrypted.
- KK key exchange – At this variant, both sides need to know each other public key. Client first sends encrypted random value to the server. The server computes session key using decrypted received random value and then sends it back to the client in cypher-text. After the Client receives and decrypts session key, both sites have the same session key.
- XX key exchange – This variant is constructed for anonymous key exchange. Both sides don't need to know each other public keys. The first Client sends initial random value to Server. The server receives a random number, then it adds an additional number and sends it back to the client. The client computes the session key, encrypts it and sends it to Server. After receiving and decrypting session key, both sides have the same session key.

MonoCyphers key exchange system works a little bit different. It needs only remote public key and local private key, to generate a session key. The remote public key is meant as a public key from another device. Local s This system is completely deterministic but could be simply upgraded to non-deterministic using both sides agreed on nonce, that might be used to XOR session key. With knowledge of nonce, it is not able to find out session key on which XOR operation was applied with nonce unless session key is known. Also, the key exchange mechanism is simpler, because there is no agreement system. That means that both sides only need to broadcast their certificate with the public key and random nonce. The nonce agreement might be based on the XOR operation of nonces from both sides.

There is also a difference in the size of final encrypted data in authenticated en-

cryption. LibHydrogen adds 36 additional bytes (random nonce and authentication tag) to final cypher-text. That means, that if the message will have the size of 279 bytes, there will be no space in MAVLink packet in payload part, where this cypher-text might be saved. In the LibHydrogen API, there is no way to split cypher-text, random nonce and authentication tag.

MonoCypher also needs nonce and authentication tag (code) to encrypt message into the payload. The difference is, that bytes, that contains nonce and authentication tag must be provided. That brings new ways to make encryption system compatible with MAVLink.

4.2.2 Performance comparison

To find out which library has better performance, an example code was created. Every API function, that might be used in final implementation was executed on ARM and X86 architecture. Same data were used as input for both libraries. Performance results are based on usage of 100% of CPU performance. The main difference in performance test was in session key exchange system. As it was described in the previous subsection MonoCypher library generates a session key based on the remote public key and local private key. Message encryption was benchmarked on message with the size of 87 bytes.

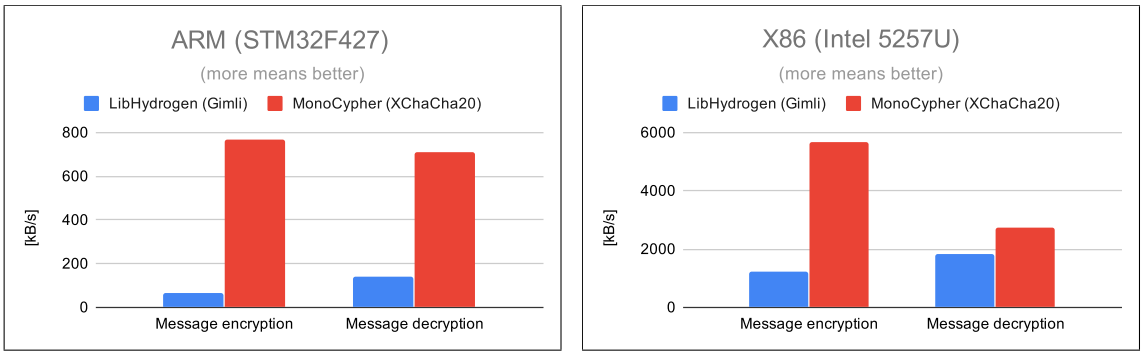


Fig. 4.1: Encryption performance of 87 bytes message, left - ARM, right - X86

4.2.3 Conclusion

Both libraries were compared in functionality and performance to find out which will suit more into PX4 platform. Main MonoCypher is significantly faster over LibHydrogen. For some tasks, MonoCypher speed was up to seventy times faster than LibHydrogen. Differences between those two libraries are bigger in performance on ARM based processor. Also, key exchange system in MonoCypher is simpler because both sides compute session key based on provided remote public keys, local private

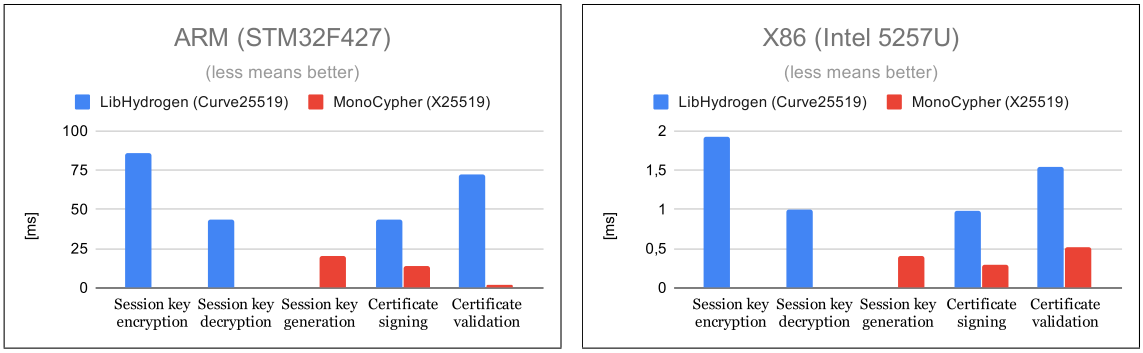


Fig. 4.2: Other tasks performance, left - ARM, right - X86

key and publicly agreed nonce. Based on speed and features that MonoCypher has, it was chosen as the cryptography library for security implementation in PX4.

5 Random number generation on PX4 platform

After choosing an encryption library, that would be compatible with any device that might use MAVLink protocol, the next very important step is to create keys with enough randomness. This means that keys must be based on the sequence of random bits from the random generator with enough entropy. Then the attacker is not able to guess any part of generated keys using side-channels or brute force attacks.

There are two types of random generators:

- Pseudo-random generators,
- True random generators.

5.1 Pseudo-random number generators

Pseudorandom Number Generators (PRGN) as described in paper created by *National Institute of Standards and Technology* (NIST) organization [19], also known as *Daterministic Random Number Generators* (DRGN) are groups of algorithms to deterministically generate numbers with nearly random properties. Result of PRGN is always based on input seed, that must be inserted before any other random number generation. PRGN are mostly based on hash functions, where we are no able to find out what was input number based on output number. That means, that if seed value has enough entropy, the no one can find out generated a random number. That doesn't apply to devices that know seed and number of iterations. Both libraries that were described in details in the previous chapter used this method of pseudo-random number generation. They need only initial seed and then using the hash function like Gimli or Blake2b, they can create a pseudo-random number. These pseudo-random numbers are not safe for cryptography.

5.2 True-random number generators

As it was said in the previous section, both libraries need an initial random number as a seed to create keys with enough entropy, so that attacker is not able to guess which combination of zeros and ones were generated. To create these random numbers, we need true-random number generators. These generators must also pass the test of randomness. This includes tests like frequency test, where a number of zeros and ones should be same in the block of bits or Run test, where it measures how many bits with the same value is generated in a row. Using this test, it is possible to determine, that the generator has enough entropy and that we can rely on it.

True random generators tests will be described more in the next chapters, where are results of random generators on different platforms.

5.2.1 True number generator on Pixhawk

Pixhawk 1 and Pixhawk 2.1 has onboard STM32F427 CPU. This processor has integrated *True Random Number Generator* (TRNG), that is based on an analogue circuit and allows to generate 32-bit random numbers based on analogue noise[20]. This circuit is constructed from ring oscillators. Outputs from those oscillators are XORed with a dedicated clock. To understand how whole TRNG works on STM32F427 processor, it is possible to inspect figure 5.1. To enable this circuit in NUTTX OS it is necessary to add CONFIG_STM32_RNG=y in firmware defconfig file. After enabling TRNG it is possible to read random numbers from /dev/random.

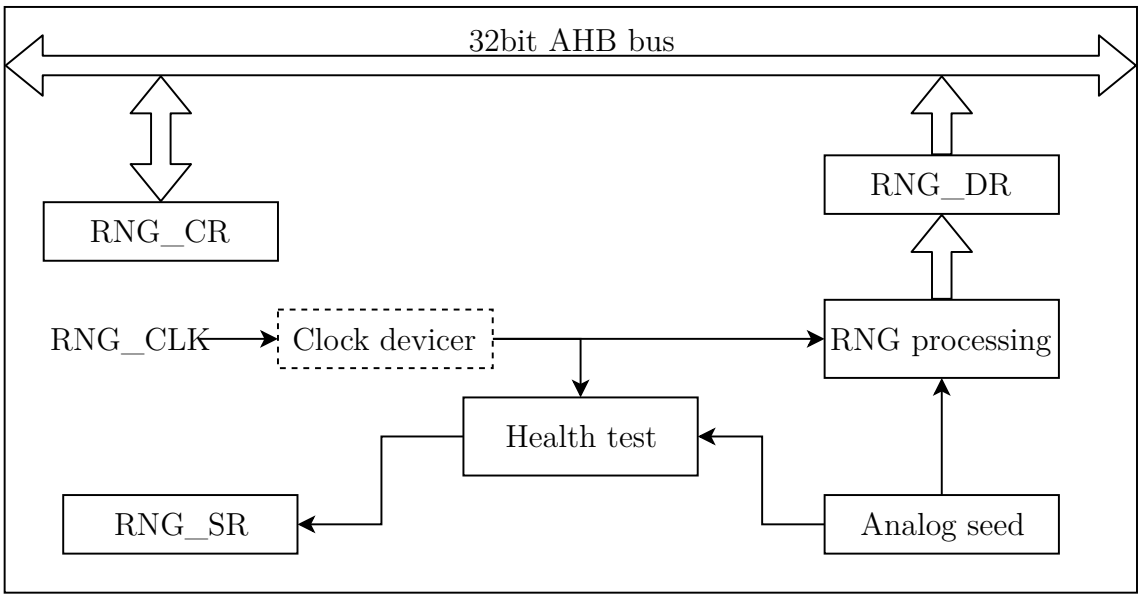


Fig. 5.1: Diagram of TRNG on STM32F427[20]

5.2.2 True number generator in QGroundControl

To generate safely random numbers in QGroundControl, Qt frameworks offer QRandomGenerator class. This class is documented on Qt website [21]. Using a static public member system() it is possible to call generate() function, which accesses systems cryptographically-safe random generators. Public member system() on Unix systems is reading from /dev/urandom or use system calls getrandom() to get a random number.

5.3 NIST tests of randomness

To find out, if TRNG are truly random, NIST created a group of statistical tests. This group of tests consists of 15 statistical tests, that each of them search different non-random binary sequences. For each test there is statistical p-value as output. P-value can be from 0 to 1. Parameter , that is also from 0 to 1 indicates how many percent of sequencues can be found non-random. If generator outputs passes all tests near value, it is considered as suitable TRNG. dependes on application. In cryptography is about 0.01. To test huge number of zeros and ones, it needs to be devided into smaller groups. Each of these small groups need to go through all tests, and if results p-value of each test is bigger then , tested group is considered as random. Also to verify if hypothesis that sequences are random, p-values needs to be distributed from 0 to 1[19]. As it can be seen on histogram in this section, distribution in all tests is good enough for both tested devices. Only results, that looks suspicious are from Maurers universal tests. Visualized results of NIST tests are avaiblable in appendix B. At this appendix on left side all plots are results of random generated sequences on Pixhawk. On right side it is possible to see results of tests based on random sequences generated on X86 processor.

6 PX4 security vulnerabilities

As it was described in previous chapters, PX4 has no security layers, that might protect telemetry messages confidentiality, integrity and authenticity. That means that information in any send telemetry message is available or might be disclosed by an unauthorized person. Also, the message might be modified without any key and the receiver doesn't know, where the message comes. At this chapter, there will be described PX4 security vulnerabilities and how MonoCypher cryptography might fix them. As an example, there were created attacks on communication between control station (QGroundControl, Windows) and Pixhawk 1 autopilot. Devices communicate over SiK radios on 443MHz. The attacker (QGroundControl, macOS) uses also SiK radio on the same frequency.

6.1 Preparations

After starting up control station and autopilot, both can communicate with each other. From GCS it is possible to plan a mission, change parameters, arm vehicle and change flight modes. After connecting of attacker into a network and starting QGroundControl software, it takes a few moments to load all necessary data. Then attackers QGroundControl software works as the main control station.

6.2 Listening to MAVLink communication (Confidential information)

Right after connecting into the vulnerable vehicle it is possible to look into MAVLink inspector, where are stored all newest messages. It is possible to view current position, altitude, airspeed or altitude of the vehicle. During this connection QGroundControl based on received messages draws vehicles flight path. From this flight path, it is possible to guess the next route. Next data, that might be confidential is flight mission, that after downloading parameters is drawn into QGroundControl map.

6.3 Shutting down vehicle during flight (Destruction)

When QGroundControl is connected to the vulnerable vehicle, there is a possibility to disarm vehicle without any validation. After clicking on Arm button. The small slider appears at bottom of QGroundControl. If UAV is landed, then there is no warning and right after activation of the slider, vehicle disables all PWM outputs,

that control motors. If UAV flies, then small slider appears at bottom of QGroundControl with a short warning. If the slider is activated PWM outputs are disabled and motors stop to rotate.

6.4 Changing vehicles flight mission (Theft)

During the time, when QGroundControl is connected, it is possible to create a mission in mission planer tab and sent a whole mission to a connected vehicle. Attacker QGroundControl then asks if he wants to fly a new mission. After confirming plane flies to new waypoints.

6.5 Changing vehicle parameters (Destruction)

During a flight is it possible to change flight parameters. This vulnerability brings big advantage for tuning PID controllers because during a few flights it is possible to tune the whole plane or multi-copter. On the other hand, it gives the attacker a tool to make stabilization or control very difficult. Also, this might end with a crash. In-flight logs there is no information about changing parameters during flight, so investigation of the crash might be difficult.

6.6 Taking control over vehicle (Theft)

In MAVLink protocol, there is message type named "RC_CHANNELS_OVERRIDE", that overrides received PPM from RC input in autopilots rail. This message allows controlling plane like it was RC transmitter, even when there is an active real RC transmitter that sends PPM. After taking control, there is no way to override these messages and attacker might fly with UAV whether he wants.

6.7 Conclusion

If anyone uses SiK radio or MAVLink Wi-Fi module or other radios without encryption, then their vehicles might be hacked and stolen or might crash without any clue. The only way to detect attacker is to watch if any malicious messages appear in QGroundControl in MAVLink inspector tab, where all sent messages in network are shown. To remove all these vulnerabilities, it is necessary to at least have whole network communication encrypted using asymmetric encryption algorithm. That solves problems, but in a bigger network, it brings additional key management.

7 New PX4 security architecture

7.1 Idea of new security architecture

In this subsection design of security, implementation will be described in more details. This subsection will be the conclusion of authors ideas during working on the semestral thesis.

7.1.1 Creating and signing vehicles certificate

Before any system in the network will be able to securely communicate with other devices, supervisor of all devices in the network must create its authority signing keys. With these keys, it is possible to sign devices certificates, that they might be trusted. Next step is to create certificates for all devices. Every certificate needs to have its devices name, maintainer, privilege level, the public key and signed the hash of all previous parts of the certificate. Certificate, key exchange key pair are generated in GCS software then hashed and then final certificate hash is signed by authority private key. Then certificate with authority public key and devices exchange keys are encrypted with a secret key and saved as a file to SD card of the device. The secret key for decryption is prebaked in source code of firmware. Reason to make secret key fixed is that autopilot boards don't have Trusted platform modules to store keys safely. After inserting SD card into autopilot board with all necessary data, autopilot is ready to be powered on. Key generation is shown on figure **??** .

7.1.2 Certificate broadcasting

After powering on, autopilots decrypt the file with certificate and key exchange keys. To do that it has prebaked secret key in source code. Then it loads all data from the decrypted file to ram and then starts to MAVLink module. With messages like HEARTBEAT or RADIO_STATUS, it starts to broadcast also newly designed message CERTIFICATE. That allows another device to verify remote devices with authority signing public key. Diagram of CERTIFICATE message broadcast is shown on figure 7.2.

7.1.3 Validation of certificate and public key

To validate other devices certificate, every device has authority signing public key encrypted in their certificate file on the SD card. After receiving CERTIFICATE message from another device, verification needs to be done. To do all parts of

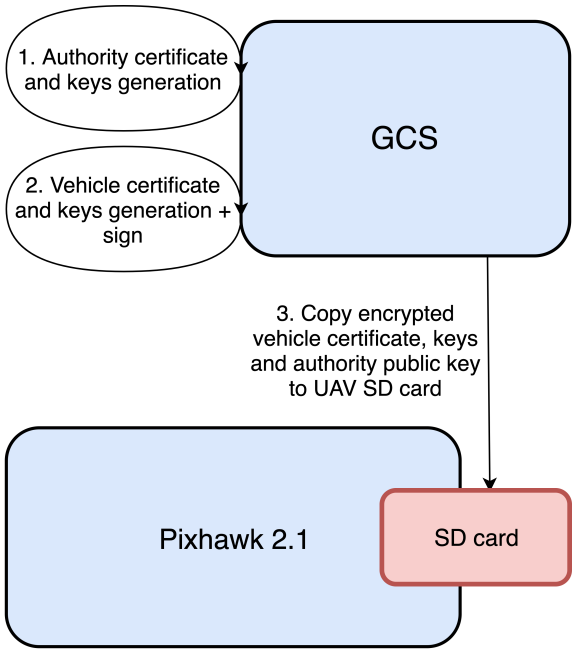


Fig. 7.1: Diagram of key generation

CERTIFICATE message like name, maintainer, privilege number and key exchange public key must be hashed. Then signed hash should be decrypted and compared with locally created hash. If they are same, then provided certificate is valid and that means that key exchange public key could be used to generate a session key.

7.1.4 Session key agreement

In MonoCypher it is possible to generate a session key based on the public key of the remote device and local private key. This way it is possible to generate the same session key on both sides. For every session, this key will be the same after generation. To make session key different, certificate message has also random nonce, that is generated after starting of device. After both nonces are known on both sides they need to be XORed. Then on both sides devices have the same random nonce and they can XOR session key with the random nonce. That allows having different session key for every session. Idea of session key agreement is visible on figure 7.3.

7.1.5 Message encryption

After both sides have the same session key, it is possible to encrypt all messages. Except for messages, that might be publicly broadcasted like HEARTBEAT or CERTIFICATE, every message will be encrypted using the session key. To make encryption design compatible with MAVLink 2.0, only payload part will be encrypted without any additional bytes. Function in MonoCypher libraries for encryption and

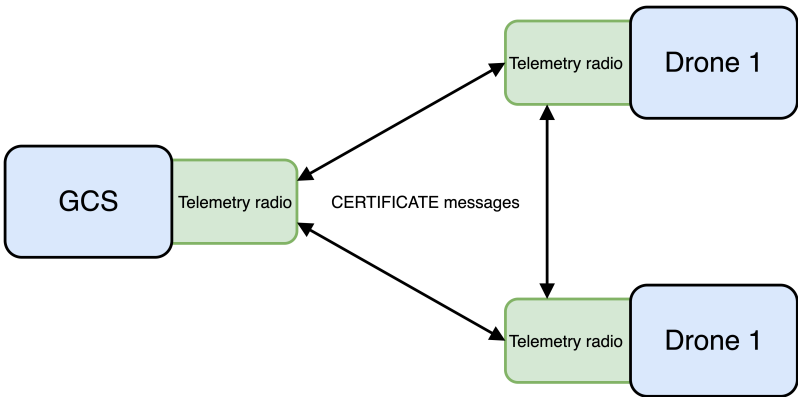


Fig. 7.2: Certificate broadcasting diagram

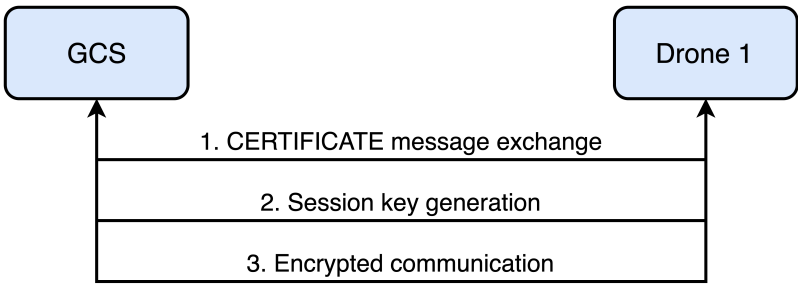


Fig. 7.3: Process of key exchange and encypted communication

decryption needs three parameters: key, nonce, mac. As it was already described in the previous section, the key is agreed between both sides and then XORed with key exchange nonces. The nonce is used to make the same messages during encryption different. Because this nonce must be same on both sides and there is no space left in MAVLink packet to send it to another device, some part of MAVLink header needs to be used for this purpose. In the MAVLink packet, there is a packet sequence number, that changes in every packet. So as nonce byte of sequence number might be used as nonce. Mac is used to authenticating source of an encrypted message, but because MAVLink has already 13 bytes part for the signature that could be used for the same thing, mac could be an array of zeros for all messages in any device. After those three parameters are prepared, the process of encryption and decryption might start.

7.1.6 Message signing

Because MAVLink packet is missing a destination address it is not possible to find out if the received message was encrypted using locally known session key. This brings a problem, that it is necessary to decrypt all received messages and then validate result if a right key was used. To solve this problem, message signing might

help. MAVLink currently offers a solution to sign every message. If every message will be encrypted and then signed with the session key. It is possible to validate the received packet by hashing it with the session key and then validating. If session key used for encryption differs with the session key for decryption, final hashes will differ.

7.2 Example of security implementation

To demonstrate how security implementation should works an example has been created based on design described in previous chapter. In example there is short code for each reviewed library, that shows how final communication system should work. Example was designed before choosing MonoCypher as library for final implementation. Example is mainly oriented to demonstrate how messages should be encrypted and signed and how key exchange should work. Also, it is possible to see there how certificate signing and verification should work. After looking at this example code person with basic knowledge of cryptography should understand what was meant and should be able to imagine how newly secured communication should works. Example code was written for both platforms. This example code was used to measure performance of both libraries.

Conclusion

In this thesis main goal was to analyse security of PX4 platform and then create idea how to fix as many as possible of those security vulnerabilities. To do this, first hardware and software of PX4 platform has been described in details. Software includes autopilot firmware, ground control station software QGroundControl and MAVLink protocol. Based on deeper knowledge of PX4 platform it was possible to choose suitable C cryptography library. As main cryptography library the MonoCypher library has been chosen. It allows to encrypt and sign messages, but also provides solution to exchange session keys. These features brings way to safely exchange MAVLink messages. Part of idea was that new encryption system will be compatible with current MAVLink packet frame. Using only few minor changes whole implementation will be possible. On Pixhawk hardware results of performance tests shown, that on autopilot hardware it is possible to encrypt and decrypt messages with speed up to 800kB per second. Another important part of thesis was to verify if PX4 hardware has safe TRNG. To do this, tests on NIST test suit has been done for ARM and X86 platform. Results shown, that autopilot boards based on STM32 processors generates sequences of bits with enough entropy. QGroundControl based on Qt framework was also able to generate random numbers with enough entropy. That means that all main parts of PX4 platform are able to safely generate keys for encryption and decryption.

Final part was all about introducing new security implementation and how it should work. To make it possible to implement all features into PX4 code, all parts of new security implementation were described in details. To make it simple to understand all ideas, descriptions includes simple diagrams.

Bibliography

[1] Lorenz Meier. How i accidentally created the most used standards in the drone industry, 2019. URL: https://auterion.com/the-history-of-pixhawk/.

[2] PX4. Px4 user guide. https://github.com/PX4/px4_user_guide, 2019.

[3] ArduPilot Dev Team. Ardupilot documentation, 2019. URL: https://ardupilot.org/ardupilot/index.html.

[4] Shawn Herrick. What's the difference between a drone, uav and uas?, 2017. URL: https://www.identifiedtech.com/blog/uav-surveying/drone-technology-ending-the-drone-vs-uav-debate-drone-basics-101/.

[5] •, •. *Open Source Licensing*, 2004. •. URL: http://dl4a.org/uploads/pdf/ebooksclub.org__Open_Source_Licensing___Software_Freedom_and_Intellectual_Property_Law.pdf.

[6] DroneCode. Px4 architectural overview, 2019. URL: https://dev.px4.io/master/en/concept/architecture.html.

[7] mavlink. Cross-platform ground control station for drones (android, ios, mac os, linux, windows). https://github.com/mavlink/qgroundcontrol, 2019.

[8] DroneCode. Flight controller selection, 2019. URL: https://docs.px4.io/v1.9.0/en/getting_started/flight_controller_selection.html.

[9] 3D Robotics, 1608 Fourth Street Berkeley, CA 94710. *PIXHAWK AUTOPILOT - QUICK START GUIDE*, 2014. •. URL: https://3dr.com/wp-content/uploads/2017/03/pixhawk-manual-rev7-1.pdf.

[10] Hex Technology, Hong Kong. *PIXHAWK 2 AUTOPILOT - QUICK START GUIDE*, 2016. •. URL: http://www.hex.aero/wp-content/uploads/2016/09/PIXHAWK2-Assembly-Guide.pdf.

[11] Hex Technology, Hong Kong. *Pixhawk v2 Feature Overview*, 2016. •. URL: http://www.hex.aero/wp-content/uploads/2016/07/DRS_Pixhawk-2-17th-march-2016.pdf.

[12] Czech Telecommunication Office, Praha. *Všeobecné oprávnění č. VO-R/10/01.2019-1 k využívání rádiových kmitočtů a k provozování zařízení krátkého dosahu.*, 2019. •. URL: https://www.ctu.cz/sites/default/files/obsah/ctu/vseobecne-opravneni-c.vo-r/10/01.2019-1/obrazky/vo-r10-012019-1.pdf.

[13] ArduPilot. Tools and firmware for the si1000. https://github.com/ArduPilot/SiK, 2018.

[14] RFDesign Pty Ltd, 6/97 Jijaws Street Sumner Park, QLD 4074. *RFD900 Radio Modem Data Sheet*, 2013. •. URL: http://files.rfdesign.com.au/Files/documents/RFD900%20DataSheet.pdf.

[15] Hex Technology, Hong Kong. *HereLink - User manual*, •. •. URL: https://fccid.io/2ARLU-HA06071/User-Manual/User-Manual-4389016.pdf.

[16] mavlink. Mavlink developer guide. https://github.com/mavlink/mavlink-devguide, 2019.

[17] jedisct1. A lightweight, secure, easy-to-use crypto library suitable for constrained environments. https://github.com/jedisct1/libhydrogen, 2019.

[18] LoupVaillant. An easy to use, easy to deploy crypto library. https://github.com/LoupVaillant/Monocypher, 2019.

[19] National Institute of Standards and Technology, Gaithersburg. *A Statistical Test Suite for Random and Pseudorandom Number Generators for Cryptographic Applications*, 2010. •. URL: https://nvlpubs.nist.gov/nistpubs/Legacy/SP/nistspecialpublication800-22r1a.pdf.

[20] STMicroelectronics, •. *STM32 microcontroller random number generation validation using the NIST statistical test suite*, 2019. •. URL: https://www.st.com/content/ccc/resource/technical/document/application_note/4a/6a/82/05/8e/9e/4e/94/DM00073853.pdf/files/DM00073853.pdf/jcr:content/translations/en.DM00073853.pdf.

[21] The Qt Company Ltd., •. *QRandomGenerator Class*, 2019. •. URL: https://doc.qt.io/qt-5/qrandomgenerator.html.

[22] GINARTeam. Nist's statistical test suite for random number generator (rng) that apply to ginar rng. https://github.com/GINARTeam/NIST-statistical-test, 2019.

[23] ArduPilot. Arduplane, arducopter, ardurover source. https://github.com/ArduPilot/ardupilot, 2019.

List of symbols, physical constants and abbreviations

AES Advanced Encryption Standard
AES-CCMP Advanced Encryption Standart-Counter Cipher Mode with Block Chaining Message Authentication Code Protocol
API Application Programming Interface
ARM Advanced RISC Machine
CAN Controller Area Network
CTO Czech Telecommunication Office
CPU Central Processing Unit
DIY Do It Yourself
DRGN Daterministic Random Number Generators
ESC Electronic Speed Control
FMU Flight Management Unit
FMUv2 Flight Management Unit version 2
FMUv3 Flight Management Unit version 3
FPU Floating Point Unit
FPV First Person View
GCS Ground Control Station
GMT Greenwich Mean Time
GPL General Public License
GPS Global Position System
HMAC Keyed-Hashing for Message Authentication
HMI Human Machine Interface
IEEE Institute of Electrical and Electronics Engineers
IMU Inertial Measurement Unit
IoT Internet of things
LGPLv3 Lesser General Public License version 3
MAVLink Micro Aerial Vehicle Link
MPU Magnetic Pickup Unit
MPU Magnetic Pickup Unit
NIST National Institute of Standards and Technology
OS Operation System
PCB Printed Circuit Board
PPM Pulse Phase Modulation
PRGN Pseudorandom Number Generators
PWM Pulse Width Modulation

PX4 Name of software flight stack, PX is shortage for Pixhawk, which was name of development team, that created PX4. Number 4 means fourth rewrite of the PX flight control software

RAM Random-Access-Memory

RC Radio Controlled

RSSI Received Signal Strength Indication

RTOS Real Time Operation System

SD Secure Digital

SHA Secure Hash Algorithm

SoC System on a Chip

TRNG True Random Number Generator

UART Universal Asynchronous Receiver-Transmitter

UAV Unmanned Aerial Vehicle

UGV Unamenned Ground Vehicle

USB Universal Serial Bus

USD United States Dollar

VTOL Vertical Takeoff and Land

WLAN Wireless Local Area Network

WPA2 Wi-Fi Protected Access 2

XML Extensible Markup Language

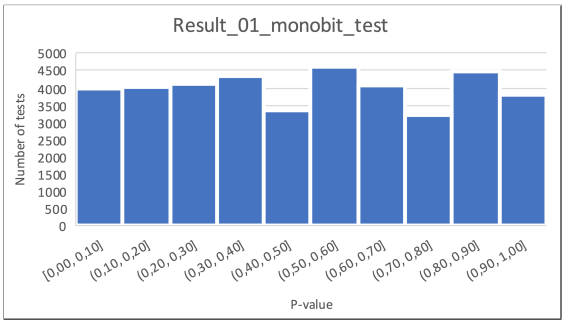
List of appendices

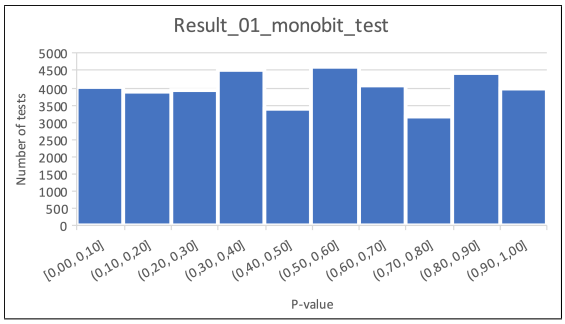
A Example of CERTIFICATE message in XML format

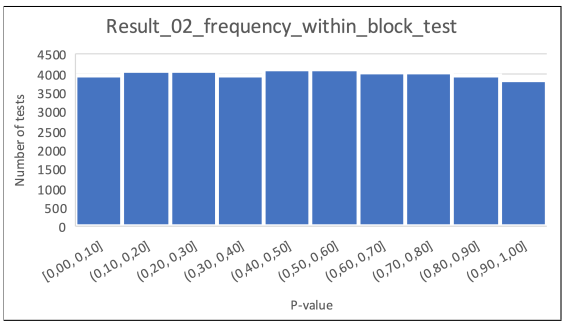
Listing A.1: Certificate stucture in C langugage

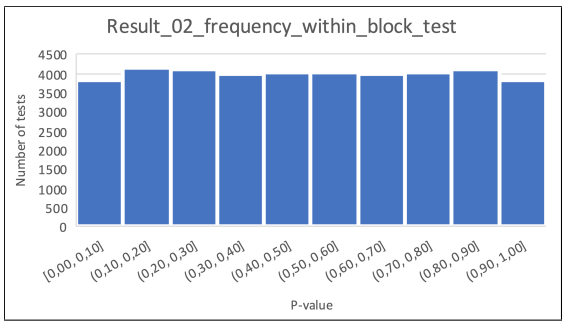
```
<message id="1000" name="CERTIFICATE">
<description>Signed certificate by authority, that is meant to
validate another device public key and privileges. If
certificate is trusted, then it is possible to communicate
with other device</description>
    <field type="uint8_t" name="System␣ID" units="">
        Network address</field>
    <field type="char[20]" name="Device␣name" units="">Name of
        machine</field>
    <field type="char[20]" name="Maintainer" units="">Name of
        maintainer</field>
    <field type="uint8_t" name="Privileges" units="rad">
        Privilages of device</field>
    <field type="uint8_t[32]" name="pubK" units="rad">Public
        key of device</field>
    <field type="uint8_t[32]" name="Nonce" units="rad/s">
        Random nonce to generate different key each time
        </field>
    <field type="uint8_t[64]" name="Sign" units="rad/s">Hash
        of previous fields signed by authority </field>
</message>
```

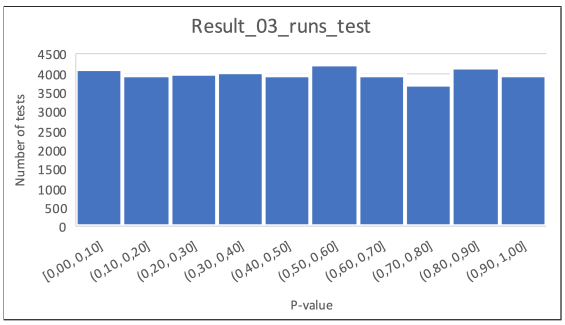
B Plots of results based on generated numbers

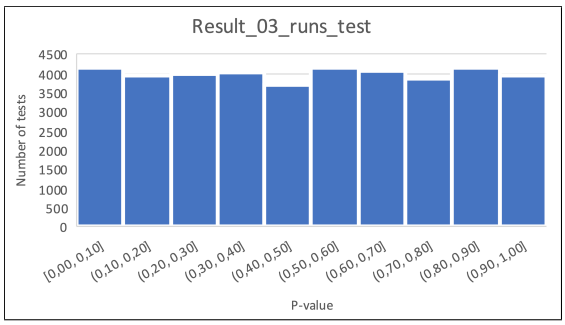


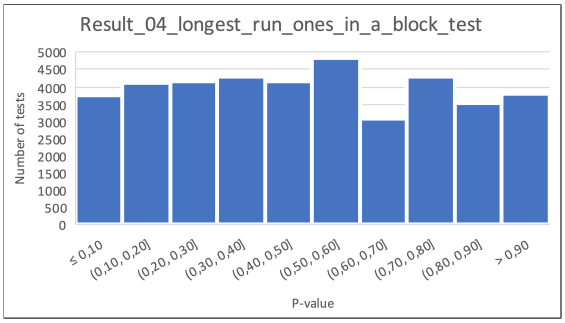


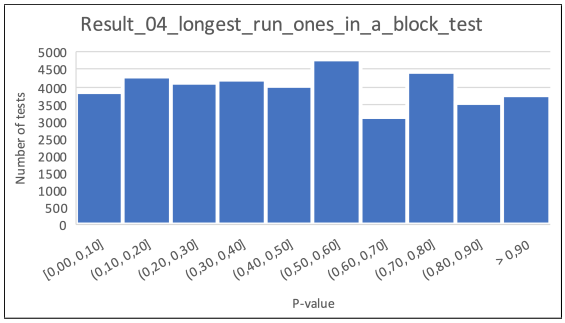


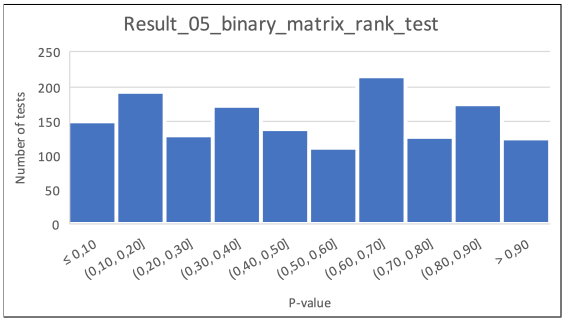


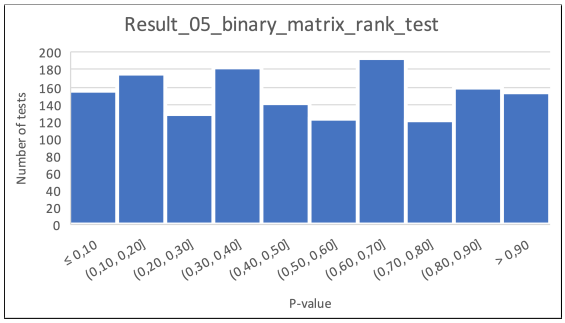


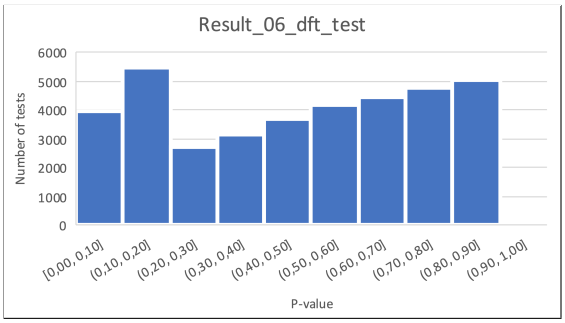


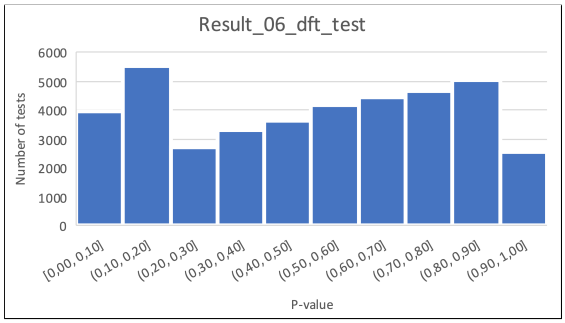

Result_04_longest_run_ones_in_a_block_test
Number of tests
P-value

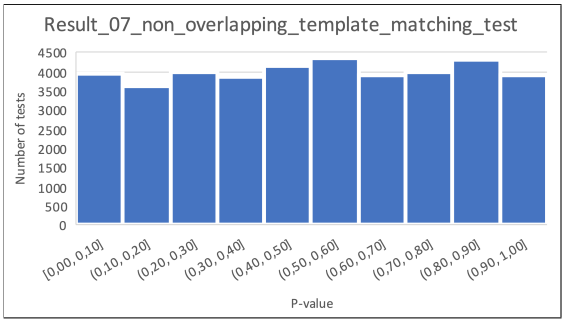

Result_04_longest_run_ones_in_a_block_test
Number of tests
P-value

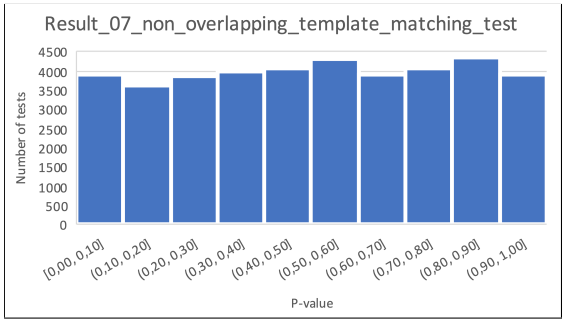

Result_05_binary_matrix_rank_test
Number of tests
P-value

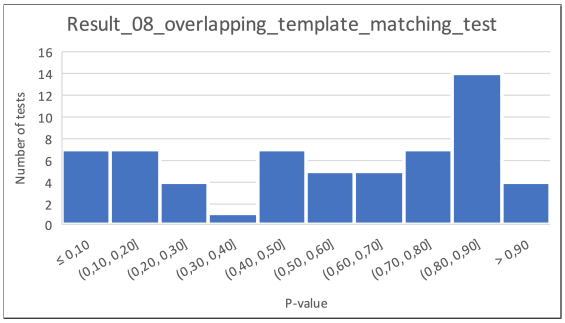

Result_05_binary_matrix_rank_test
Number of tests
P-value

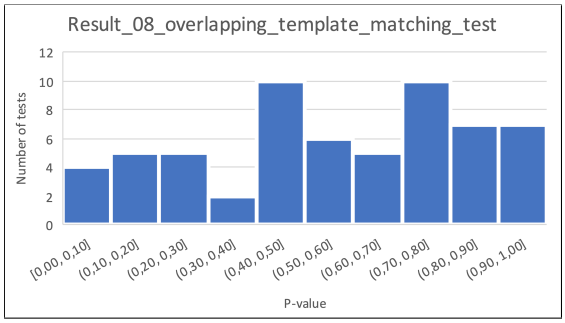

Result_06_dft_test
Number of tests
P-value

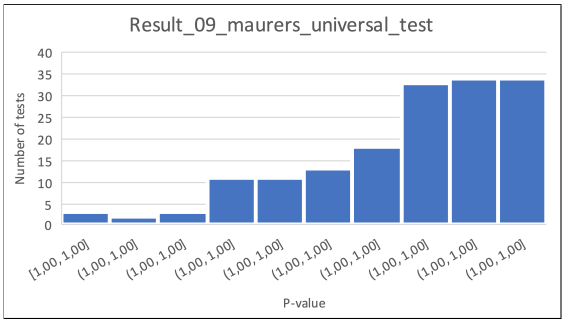

Result_06_dft_test
Number of tests
P-value

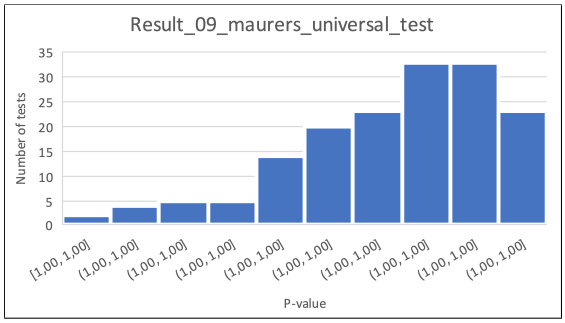

Result_07_non_overlapping_template_matching_test
Number of tests
P-value

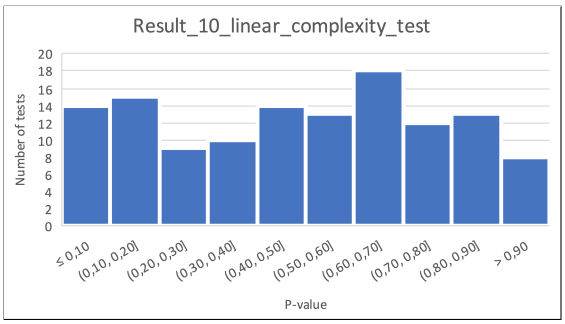

Result_07_non_overlapping_template_matching_test
Number of tests
P-value

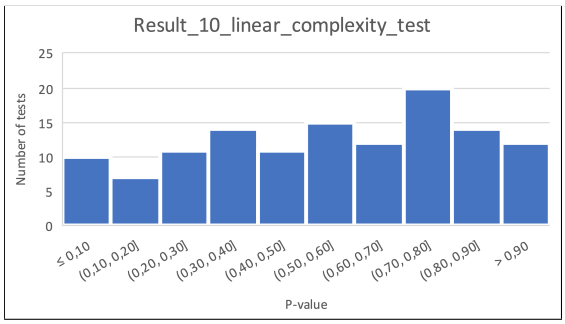

Result_08_overlapping_template_matching_test
Number of tests
P-value

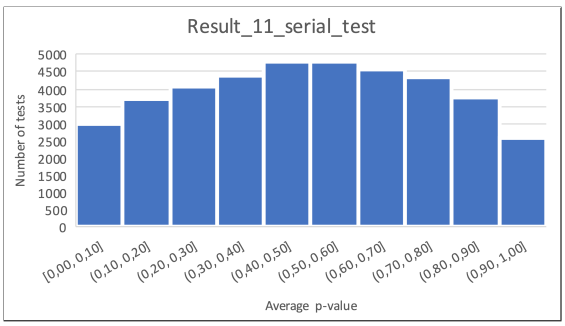

Result_08_overlapping_template_matching_test
Number of tests
P-value


Result_09_maurers_universal_test
Number of tests
P-value

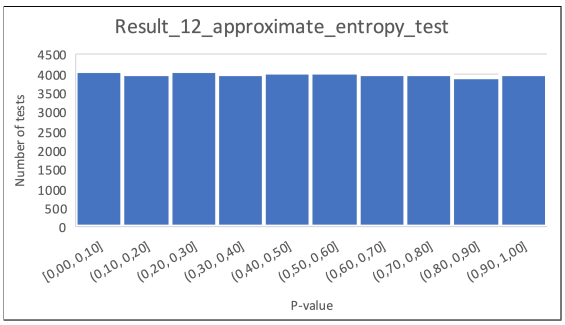

Result_09_maurers_universal_test
Number of tests
P-value

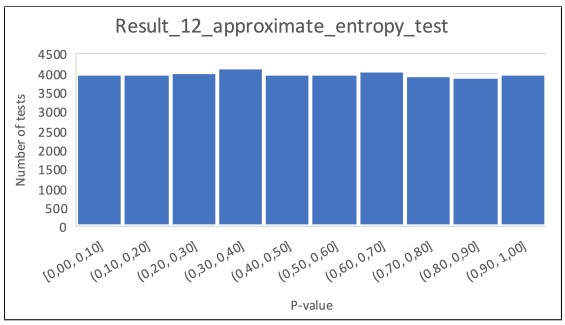

Result_10_linear_complexity_test
Number of tests
P-value

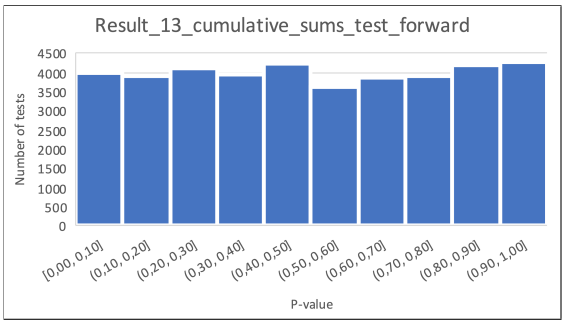

Result_10_linear_complexity_test
Number of tests
P-value

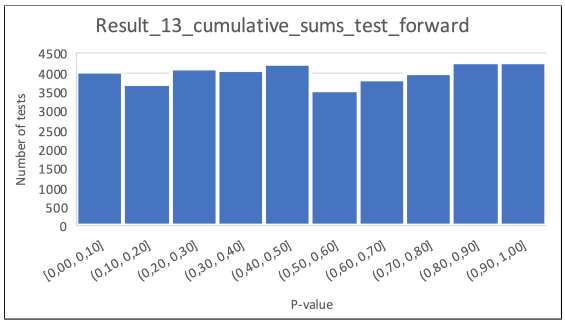

Result_11_serial_test
Number of tests
Average p-value

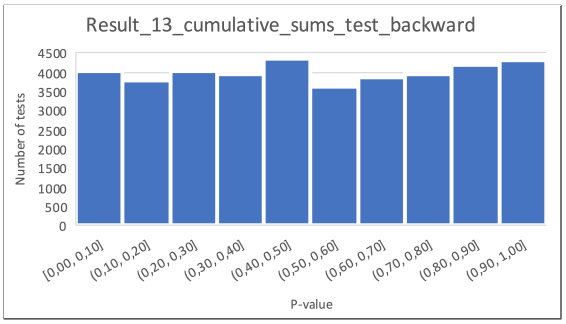

Result_11_serial_test
Number of tests
Averagep-value

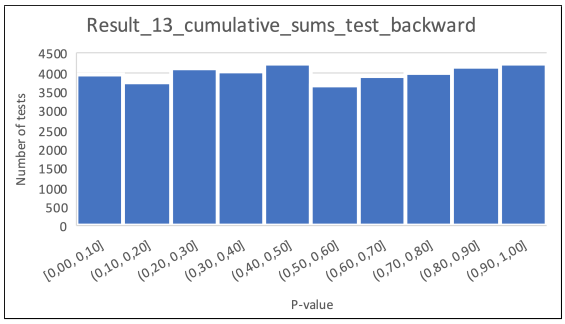

Result_12_approximate_entropy_test
Number of tests
P-value

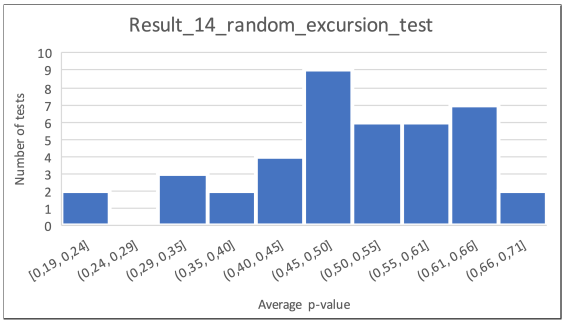

Result_12_approximate_entropy_test
Number of tests
P-value

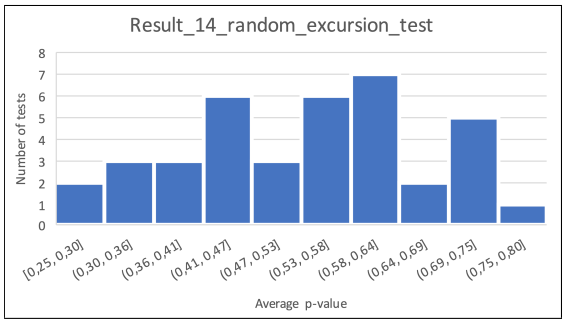

Result_13_cumulative_sums_test_forward
Number of tests
P-value

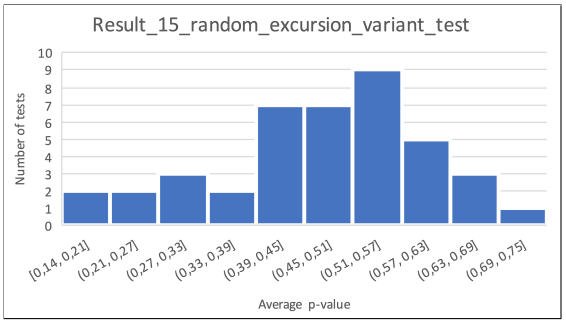

Result_13_cumulative_sums_test_forward
Number of tests
P-value

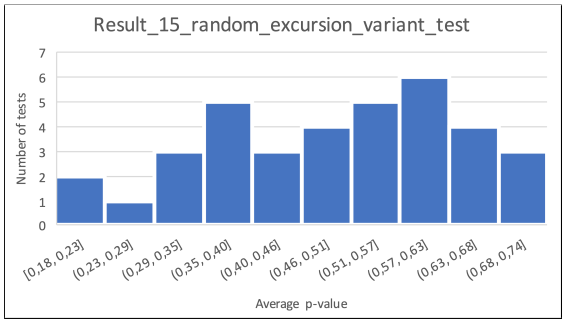

Result_13_cumulative_sums_test_backward
Number of tests
4500
4000
3500
3000
2500
2000
1500
1000
500
0
[0,00, 0,10]
[0,10, 0,20]
[0,20, 0,30]
[0,30, 0,40]
[0,40, 0,50]
[0,50, 0,60]
[0,60, 0,70]
[0,70, 0,80]
[0,80, 0,90]
[0,90, 1,00]
P-value


Result_13_cumulative_sums_test_backward
Number of tests
4500
4000
3500
3000
2500
2000
1500
1000
500
0
[0,00, 0,10]
[0,10, 0,20]
[0,20, 0,30]
[0,30, 0,40]
[0,40, 0,50]
[0,50, 0,60]
[0,60, 0,70]
[0,70, 0,80]
[0,80, 0,90]
[0,90, 1,00]
P-value


Result_14_random_excursion_test
Number of tests
10
9
8
7
6
5
4
3
2
1
0
[0,19, 0,24]
[0,24, 0,29]
[0,29, 0,35]
[0,35, 0,40]
[0,40, 0,45]
[0,45, 0,50]
[0,50, 0,55]
[0,55, 0,61]
[0,61, 0,66]
[0,66, 0,71]
Average p-value


Result_14_random_excursion_test
Number of tests
8
7
6
5
4
3
2
1
0
[0,25, 0,30]
[0,30, 0,36]
[0,36, 0,41]
[0,41, 0,47]
[0,47, 0,53]
[0,53, 0,58]
[0,58, 0,64]
[0,64, 0,69]
[0,69, 0,75]
[0,75, 0,80]
Average p-value


Result_15_random_excursion_variant_test
Number of tests
10
9
8
7
6
5
4
3
2
1
0
[0,14, 0,21]
[0,21, 0,27]
[0,27, 0,33]
[0,33, 0,39]
[0,39, 0,45]
[0,45, 0,51]
[0,51, 0,57]
[0,57, 0,63]
[0,63, 0,69]
[0,69, 0,75]
Average p-value


Result_15_random_excursion_variant_test
Number of tests
7
6
5
4
3
2
1
0
[0,18, 0,23]
[0,23, 0,29]
[0,29, 0,35]
[0,35, 0,40]
[0,40, 0,46]
[0,46, 0,51]
[0,51, 0,57]
[0,57, 0,63]
[0,63, 0,68]
[0,68, 0,74]
Average p-value