PX4 security analyse

PX4 Software stack

Overview

PX4 is open source community-based autopilot platform, founded by Lorenz Meier in 2008. At this time Lorenz was master's degree student, that wants to create UAV multicopter with possibility of autonomous flight base on computer vision. Today PX4 is industry leading platform, that allows companies to create their own UAVs for commercial usage. PX4 and compatible hardware is also price affordable for hobbyist. This brings very easy way how to construct UAV from cheap hardware using this open-source software stack. Nearly for anyone with basic knowledge of computer science and electronics it is possible to build quadcopter that can have any payload. This is ideal for hobbyist, but also it allows to create UAV as weapon to create terror.

One of biggest advantages of this platform is that it allows developers and users to create traditional multicopters, planes, rovers and 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 or video recording, but it is also possible to use lidars or multispectral cameras. This create new opportunities for new businesses to use these drones for precise agriculture, geodesy, during search and rescue operations or for military usage.

PX4 is not the only one open-source platform for unamend systems. There is also ArduPilot project, that was created in 2009. Today ArduPilot together with PX4 are one of most used autopilots software for lot of different types of vehicles. They both use mavlink protocol to create missions or provide telemetry data to pilot. Because they both use mavlink protocol, it is possible to plan missions or control both platforms from QGroundControl software.

In the beginning of this thesis I would like to describe one thing. Lot of peoples uses drone word in wrong way. All flying machines without pilots are shortly UAVs (Unmanned aerial vehicle). Only UAVs that are able to fly autonomously can be called drones. In this thesis I will be talking mainly about UAVs or UGV (unmanned ground vehicle), and only when vehicle will be able to do missions autonomously, then it will be called drone.

Whole PX4 Flight stack, that includes autopilot firmware is licensed under BSD 3-clausule licence. This licence is one of most free licence that is commonly used. It allows users and programmers to copy code without any limitation. There is no limitation at numbers of software copies, but also there is no limitation in next development, that includes modifying source code. There is also no need to share source codes like with GNU GPL licence. This means that this licence is very good for proprietary forks of open sourced software. Only thing that must be included in source code is

PX4 Flight stack 3-clause BSD licence

Copyright (c) 2012 - 2019, PX4 Development Team
All rights reserved.

Redistribution and use in source and binary forms, with or without
modification, are permitted provided that the following conditions are met:

* Redistributions of source code must retain the above copyright notice, this
  list of conditions and the following disclaimer.

* Redistributions in binary form must reproduce the above copyright notice,
  this list of conditions and the following disclaimer in the documentation
  and/or other materials provided with the distribution.

* Neither the name of the copyright holder nor the names of its
  contributors may be used to endorse or promote products derived from
  this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS"
AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
ARE
DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT HOLDER OR CONTRIBUTORS BE LIABLE
FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL
DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR
SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER
CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY,
OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE
OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

PX4 autopilot firmware

PX4 autopilot firmware is one of the most important part of whole PX4 platform. This piece of software is something like pilot, that watch all sensors values and listen to commands. Based on that data, it generates outputs to all actuator to control unmanned vehicle.

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 layer that handles all internal and external communications, but also provides drivers for supported hardware.

Flight stack

This layer is collection of all estimation and control algorithms. There are controllers for all different kinds of vehicles like rovers, multirotors, helicopters and VTOL. Using these controllers, it is possible to maintain position and attitude of vehicle.

FLIGHT STACK DIAGRAM

Middleware

Middleware layer provide all drivers for supported embedded sensors and communication with them. It also provides communication tools for all peripherals like 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 public-subscribtion system. This provide solution do distribute data where they are needed.

QGroundControl software

To setup and control unmanned system based on PX4 or ArduPilot, user needs to have some software, that is able to communicate with that system. In case PX4 and ArduPilot, there is software called QGroundControl. It is multiplatform mission planer, that allows users to setup their vehicles, change configurations and also provide way to plan missions. Also, it is possible to view telemetry data and actual position and status of plane.

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 Qt framework (Qt for application development), that brings Qt 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 company wants to develop proprietary software, it needs to pay for every software developer, that works on that software. If company don't want to pay, that it needs to provide source codes of their software, because software must be licensed under GPL or LGPLv3.

PX4 Hardware

Main and most important part of PX4 platform is Pixhawk autopilot. It is PCB with all sensors and connectors necessary to fly and control UAV. Mostly used Pixhawk 1 is based on 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.
Manufactures of Pixhawk:

- 3DR
- mRo
- mRobotics
- HobbyKing
- Holybro
- Drotek

- Hex (Pixhawk 2.1)

To control actuators, every PX4 autopilot board must have some kind of outputs. On Pixhawk board series there are two types of outputs: digital signal or communication bus. Most used digital signal is PWM, that are options like PWM signal, S.Bus or it is also possible to UART or CAN buses. Most used is PWM signal, that is very easy to use, but also there are tons 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 2.1 and Pixhawk 2.4.8 boards, that will be described in more detail in next subchapter.

Pixhawk 1 (2.4.8)

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

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
- 14x PWM output
- 1x RC input
- 1x RSSI input

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. 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 to create more complex control and estimation algorithms. This board comes with price 238USD per set.

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

“Cube” 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

“Cube” 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

Hardware setups

In this section there will descriptions of few hardware setups, that you might find between 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 link 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

- Mavlink 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
- Mavlink 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
- Mavlink 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

- Mavlink security issues
    - Encryption works (AES256)
    - No authentication
    - No access control

Devices in details

ČTU regulations: https://www.ctu.cz/cs/download/oop/rok_2010/vo-r_12-09_2010-12.pdf

SiK telemetry radio

SiK telemetry radios are devices based on open-source SiK firmware and cheap SiLabs S1000 SoC. As example I will describe in more details Holybro telemetry radio V3 433MHz. 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 other autopilot. With included antenna and with default configuration it is possible to control drone at a distance 300m. With default configured output power, that is 100mW, it is not legal to use this device in Czech Republic.

Source: http://www.holybro.com/product/transceiver-telemetry-radio-v3/

WIFI telemetry radio

This telemetry device is using WIFI standards (802.11 b/g/n) to create network where it broadcasts mavlink packets. Setup is very easy, because on ground station side there is no need to have any other device except computer. This solution is compatible with any Mac, Linux or Windows computer. Also, it is possible to use Android tablet or iPad. With output power 100mW it is legal to use this telemetry device in Czech Republic. If WPA2 is enabled, all telemetry is encrypted using pre-shared key with encryption algorithm AES-CCMP

Source: https://www.amazon.com/YKS-Pixhawk-Wireless-Telemetry-Replacement/dp/B015XO0BE2

RFD868 combo

RFD868 is one of best solution to communicate with unmanned systems. 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 has feature to passthrough PPM signal to control drone. All RFD868 radio sets allows to encrypt all mavlink and PPM data using AES128 encryption algorithm. There is also TXMOD package, that might be connected to JR socket in Frsky's radios. This TXMOD package provides WIFI access point, that creates WIFI network all-around and any WIFI compatible device might connect to this network and receive mavlink

communication. In Czech Republic is it legal to use this device with output power up to 500mW, but only on one channel (869,4 - 869,65MHz).

Source: http://store.rfdesign.com.au/rfd868-txmod-bundle/

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. 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 unamend system. On air unit, there are two HDMI input ports, where you can insert output from any HDMI camera. For 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 QGC, because modified version for Herelink isn't opensource.

Source: http://www.hex.aero/wp-content/uploads/2018/11/HereLink-Manual-EN.pdf

RFD868 in Multi-point mode

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

Here I should add information about bandwidth of multipoint radio and theoretical maximal number of nodes. Need some data from manufacturer, but right now I have no answer.

Conclusion

As It was described in this chapter, there are some setups where it is possible to encrypt all communication. Some of them doesn't have any encryption for mavlink messages. So, if someone wants to have encrypted communication, then there are some setups, but there is no setup where one side might authenticate if messages come from trusted source or has access control system, so there is option to create users with different privileges. Because of this reason I decided to create in this thesis implementation of encryption, authentication and access control. In next chapter I will describe in more details mavlink protocol and create security architecture for PX4 platform without dependencies on hardware. All parts of security implementation will be possible to use on any hardware.

MAVlink protocol

MAVlink (Micro Air Vehicle link) is very lightweight communication protocol to communicate with unmanned systems or with other devices onboard. For example, using MAVlink

protocol you can upload flight mission, get all flight data or change parameters of unmanned system. MAVlink is based on modern hybrid publish-subscribe or point-to-point design pattern. All data streams are send / published as topic, while configuration sub-protocols such as the mission protocol or parameter protocol are point-to-point with retransmission.

This protocol is lightweight because there is very small overhead per packet. MAVlink v1 has only 8 bytes of header for every message. In MAVlink v2, there is 14 bytes overhead, but is more secure and extensible. In headers there are bytes like

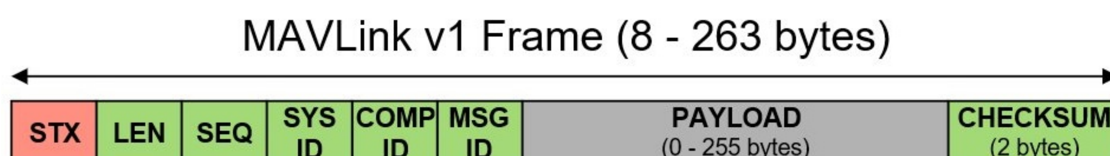
All messages are specified using XML in multiple dialects, where every message has its own definition. Dialects allows to create different sets of messages for different types of systems that use mavlink protocol as communication tool.

EXAMPLE of MAVLINK message definition from Common.xml

```
<message id="30" name="ATTITUDE">
   <description>The attitude in the aeronautical frame (right-handed, Z-down, X-front, Y-
right).</description>
   <field type="uint32_t" name="time_boot_ms" units="ms">Timestamp (time since system
boot).</field>
   <field type="float" name="roll" units="rad">Roll angle (-pi..+pi)</field>
   <field type="float" name="pitch" units="rad">Pitch angle (-pi..+pi)</field>
   <field type="float" name="yaw" units="rad">Yaw angle (-pi..+pi)</field>
   <field type="float" name="rollspeed" units="rad/s">Roll angular speed</field>
   <field type="float" name="pitchspeed" units="rad/s">Pitch angular speed</field>
   <field type="float" name="yawspeed" units="rad/s">Yaw angular speed</field>
  </message>
```

Mavlink V1

MAVlink 1.0 was first released in 2009 by Lorenz Meier. Its 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).

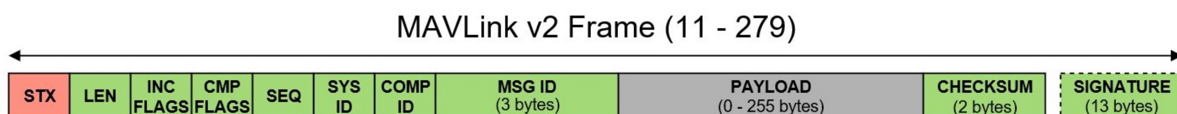


Header description:

- 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.

Mavlink V2

Compared to the older version of mavlink, version 2 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.



Compared to Mavlink 1.0 in Mavlink 2.0 there were added new bytes into 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

Packet signature solution

Mavlink 2.0 adds message signing feature, that allows to authenticate if message comes from reliable source. To do this SIGNATURE were added, where sign is stored. This create security layers where attackers suppositious messages are not accepted on receiver side, because there is no way to create valid signature for fake message without 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 HMAC system and computed based on SHA256 algorithm. Commonly SHA256 hash algorithm return array of bits with length of 256. To lower length of signed message final signature of mavlink is shorten to 48 bits. In case of security shorter signature doesn't lose signature strength. There is very low probability (1/2^48bit), to find signature that will suits some message.

Signature is computed base on this formula:

Signature = 256to48bits(SHA256(secret key + header + payload + CRC + link-ID + timestamp))

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.
- Timestam is older than 1 minute.

Mavlink 2.0 security issues

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

Message encryption
Next disadvantage of Mavlink protocol is lack of encryption. All data that might be send over radio are send in plaintext. This allows 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 pilot might have. In hobby or commercial industrial this might not be huge problem, but in case of military or police usage, mission plans and information might be classified. This brings 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 signature system, symmetric encryption needs key, that must be distributed to all devices in network. This brings same vulnerability as signature system, where disaffection of any device gives key to all encrypted messages. To eliminate this distribution problem and to create key exchange system, there is solution to add private key infrastructure into MAVlink protocol.

Key exchange
If there are only two devices in MAVlink network, then encryption and singing keys distribution is very simple. If key is changed than for example there are only two keys to change. In ground control station and in drone. In case of 100 drone fleet, operator will need in case of key reseeding to change new key in all devices. This will very time consuming and

not very flexible solution. To solve this problem there is private key infrastructure system with its key exchange system for all devices network. This means that every device must have its own 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 network. Using this certificate, it is possible to

Access control

MAVlink lacks also from access control system, that will allow to create groups of devices with different access rights. As example there is military group of soldiers that has fleet of drones to operate. In Mavlink network there are 5 devices: Ground control station handled by main pilot, monitor to control and view cameras. There are also 3 UAVs, that are on missions. Main pilot handles mission planning. Secondary pilot that has monitor controls cameras. In that case there should be access control system to separate what each device can do with other devices. It is undesirable to be able to control UAV from monitor, that is handled by non-authorized person and vice versa It is undesirable to control camera and watch output video when pilot should be focused on UAVs control. Also, 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.

Sources: https://mavlink.io/en/

Encryption libraries

In previous chapter there were showed all security issues, that

Encryption libraries

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

LibHydrogen

LibHydrogen is cryptographic library based on LibSodium library. It aims to be easy to use and to have good performance any supported architecture. It has implemented only two necessary cryptographic 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. These two cryptographic building blocks will be described in more details in next subchapters.

Gimli

Gimli is a 384bit permutation designed with idea to have encryption algorithm with good performance on nearly all platforms. It is possible to implement this algorithm for 64bit processors made by Intel or AMD, but also ARM 32-bit processors or 8-bit controllers like ATMega based on AVR architecture.

Sources https://gimli.cr.yp.to/gimli-20170627.pdf

EC25519

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.

ChaCha20 ()

Poly1305

Blake2b

X25519

EdDSA

Comparison

Both libraries have same nearly all functionalities with only small differences. In this subchapter there will be in details described differences in both libraries.

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, 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 public key of other side. Session key is generated also on one side. When session key is encrypted using public key from other side it is send to other side, where it is decrypted.
- KK key exchange – At this variant both sides needs to know each other public key. Client first sends encrypted random value to server. Server computes session key using decrypted received random value and then sends it back to client in ciphertext. After Client receives and decrypts session key, both sites have 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. First Client sends initial random value to Server. Server receives random number, then it adds additional number and sends it back to client. Client computes session key, encrypts it and sends it to Server. After receiving and decrypting session key, both sides have same session key.

Monocyphers key exchange system works little bit different. It has only one system, that needs only remote public key and local secret key, to generate session key. Remote public key is meant as public key from other device. Local s This system is completely deterministic, but could be simply upgraded to non-deterministic using both sides agreed 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, key exchange mechanism is simpler, because there is no agreement system. That means that both sides only need to broadcast their certificate with public key and random nonce. Nonce agreement might be based on XOR operation of nonces from both sides.

There is also difference in size of final encrypted data in authenticated encryption. LibHydrogen adds 36 additional bytes (random nonce and authentication tag) to final ciphertext. That means, that if message will have size of 279 bytes, there will be no space in mavlink packet, where this ciphertext might be saved. In the LibHydrogen API there is no way to split ciphertext, random nonce and authentication tag.

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

SIZE ?????

PERFORMANCE

Type of operation	ARM (STM32F427)	X86 (Intel 5257U)

	LibHydrogen	MonoCypher	LibHydrogen	MonoCypher
Certificate signing			0.982	0.294
Certificate validation			1.546	0.524
Session key encryption			1.923	
Session key decryption			1.000	
Session key generation				0.404
Message encryption			0.070	0.015
Message decryption			0.047	0.031

Conclusion

Both libraries were compared in functionality and performance to find out which will suit more into PX4 platform. Main monocipher is significantly faster over LibHydrogen. For some tasks Monocypher speed was up to five times faster than LibHydrogen. Differences are bigger on 64 bit processors based on x86 architecture ?????. Also, key Exchange system in monocypher is simpler because both sides computes session key based on accessible remote public keys and local and public agreed nonce.

Based on speed and features provided by Monocypher will be chosen as cryptography library for security implementation in PX4.

Random number generation on PX4 platform

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

But where this random numbers come from? There are two types of random generators:

- Pseudo-random generators
- True random generators

Both types will be described in details and then we will find which type should we use for key generation.

Pseudorandom generators

Pseudorandom number generators (PRNG) also known as deterministic random bit generators (DRBG) are groups of algorithms to deterministically generate numbers with nearly random properties. Result of PRNG is always based on input seed, that must be inserted before any other random number generation. These pseudorandom number generators 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, then no one is able to find out generated random number. That doesn't apply to devices that know seed and number of iterations. Both libraries that were described in details in

previous chapter used this method of random number generation. They need initial seed and then using hash function like Gimli or Blake2b, they create random number.

True-random generators

As it was said in previous subchapter, both libraries need initial random number as 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 test of randomness. This includes tests like frequency test, where number of zeros and ones should be same in block of bits or Run test, where it measures how many bits with same value is generated in row. Using this test, it is possible to determine, that generator has enough entropy and that we can rely on it. True random generators tests will be described more in next chapters, where are results of random generators on different platforms.

Both libraries have implemented true random number generator drivers on most platform, that are used for IoT or Embedded solution, but not on NUTTX OS which is used as RTOS on PX4 autopilot. Adding support for NUTTX OS in both libraries will be described in next chapters.

Tests of true random generator

At this section it will be described in more detail tests of ARM and x86 true random generators. All tests will be described details and both platforms will be tested.

Results of true random numbers for both architectures (320000 random bits):

Tests	ARM (STM32F427)		X86 (Intel 5257U)	
	Result value	Result	Result value	Result
2.01			0.9537486285283232	True
2.02			0.21107154370164066	True
2.03			0.5619168850302545	True
2.04			0.7189453298987654	True
2.05			0.3061558375306767	True
2.06			0.8471867050687718	True
2.07			0.07879013267666338	True
2.08			0.11043368541387631	True
2.09			0.282567947825744	True
2.10			0.8263347704038304	True
2.111			0.766181646833394	True
2.112			0.46292132409575854	True
2.12			0.7000733881151612	True
2.131			0.6698864641681423	True
2.132			0.7242653099698069	True
2.151				True

2.152				True

2.01. Frequency Test: (0.9537486285283232, True)
2.02. Block Frequency Test: (0.21107154370164066, True)
2.03. Run Test: (0.5619168850302545, True)
2.04. Run Test (Longest Run of Ones): (0.7189453298987654, True)
2.05. Binary Matrix Rank Test: (0.3061558375306767, True)
2.06. Discrete Fourier Transform (Spectral) Test: (0.8471867050687718, True)
2.07. Non-overlapping Template Matching Test: (0.07879013267666338, True)
2.08. Overlappong Template Matching Test: (0.11043368541387631, True)
2.09. Universal Statistical Test: (0.282567947825744, True)
2.10. Linear Complexity Test: (0.8263347704038304, True)
2.111(2). Serial Test: ((0.766181646833394, True), (0.46292132409575854, True))
2.12. Approximate Entropy Test: (0.7000733881151612, True)
2.132. Cumulative Sums (Forward): (0.6698864641681423, True)
2.131. Cumulative Sums (Backward): (0.7242653099698069, True)
2.15. Random Excursion Test:

STATE	xObs	P-Value	Conclusion
'-4'	3.8356982129929085	0.5733056949947805	True
'-3'	7.318707114093956	0.19799602021827734	True
'-2'	7.861927251636425	0.16401104937943733	True
'-1'	15.69261744966443	0.007778723096466819	False
'+1'	2.4308724832214765	0.7868679051783156	True
'+2'	4.7989062888391745	0.44091173664620265	True
'+3'	2.3570405369127525	0.7978539716877826	True
'+4'	2.4887672641992014	0.7781857852321322	True

2.15. Random Excursion Variant Test:

STATE	COUNTS	P-Value	Conclusion
'-9.0'	1450	0.8589457398254003	True
'-8.0'	1435	0.7947549562546549	True
'-7.0'	1380	0.5762486184682754	True
'-6.0'	1366	0.4934169340861271	True
'-5.0'	1412	0.6338726691411485	True
'-4.0'	1475	0.9172831477915963	True
'-3.0'	1480	0.9347077918349618	True
'-2.0'	1468	0.8160120366175745	True
'-1.0'	1502	0.8260090128330382	True

'+1.0'	1409	0.13786060890864768	True
'+2.0'	1369	0.20064191385523023	True
'+3.0'	1396	0.4412536221564536	True
'+4.0'	1479	0.939290606067626	True
'+5.0'	1599	0.5056826821687638	True
'+6.0'	1628	0.4459347106499899	True
'+7.0'	1619	0.5122068856164792	True
'+8.0'	1620	0.5386346977772863	True
'+9.0'	1610	0.5939303958223099	True

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 unauthorized person. Also, message might be modified without any key and receiver doesn't know, where message comes. At this chapter there will be described PX4 security vulnerabilities and how MonoCypher cryptography might fix them.

As example there were created attacks on communication between GCS (QGroundControl, Windows PC) and Pixhawk 1. Devices communicate over SiK radios 443MHz.

Listening to mavlink communication (Confidential information)

Shutting down vehicle during flight (Destruction)

Changing vehicles flight mission (Theft)

Changing vehicle parameters (Destruction)

Taking control over vehicle (Theft)

Conclusion

PX4 security architecture

Main goal of this thesis is to create design and then try implement it into mavlink protocol and PX4 software. Idea is to eliminate all known vulnerabilities of mavlink communication, but also to make security implementation compatible with mavlink 2.0 or make it compatible with only few minor changes.

Security design

In this subchapter design of security implementation will be describe in more details. This subchapter will be conclusion of authors ideas during working on semesterly thesis.

Creating and signing vehicles certificate

Before any system in network will be able to securely communicate with other devices, supervisor of all devices in network must create its authority signing keys. With these keys it is possible to sign devices certificates, that then might be trusted. Next step is to create certificates for all devices. Every certificate needs to have its devices name, maintainer, privilege level, public key and signed hash of all previous parts of certificate. Certificate, key exchange keypair 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 secret key and saved as file to SD card of device. Secret key for decryption is prebaked in source code of firmware. Reason to make secret key fixed is that autopilot boards doesn'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.

```
typedef struct certificate{
   char name[20];
   char maintainer[20];
   uint8_t privilages;
   uint8_t pubK[32];
   uint8_t sign[64];
} certificate;
```

Certificate broadcasting

After powering on, autopilots decrypt file with certificate and key exchange keys. To do that it has prebaked secret key in source code. Then it loads all data from 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 other device to verify remote devices with authority signing public key.

```
typedef struct CERTIFICATEmessage{
   uint8_t systemID;
   char name[20];
   char maintainer[20];
   uint8_t privilages;
   uint8_t pubK[32];
   uint8_t sign[64];
   uint8_t nonce[32];
} CERTIFICATEmessage;
```

Validation of certificate and public key

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

Session key agreement

In MonoCypher it is possible to generate session key based on public key of remote device and local private key. This way it is possible to generate same session key on both sides. For every session this key will be 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 same random nonce and are able to XOR session key with random nonce. That allows to have different session key for every session.

Message encryption

After both sides have same session key, it is possible to encrypt all messages. Ecxept messages, that might be publicly broadcasted like HEARTBEAT or CERTIFICATE, every message will be encrypted using 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 decryption needs three parameters: key, nonce, mac. As it was already described in previous section, key is agreed between both sides and then XORed with key exchange nonces. Nonce is used to make same messages during encryption different. Because this nonces must be same on both sides and there is no space left in MAVlink packet, it needs to be used some part of MAVlink packet header. In mavlink packet, there is packet sequence number, that changes in every packet. So as nonce byte of sequence number might be used as nonce. Mac is used to authenticate source of encrypted message, but because MAVlink has already 13 bytes part for signature that could be used for same thing, mac could be array of zeros for all messages in any device. After those three parameters are prepared, process of encryption and decryption might start.

Message signing

Because mavlink packet is missing a destination address it is not possible to find out if received message was encrypted using locally known session key. This bring problem, that it is necessary to decrypt all received messages and then validate result if right key was use. To solve this problem, message signing might help. Mavlink currently offers solution to sign every message. If every message will be encrypted and then signed with session key. It is possible to validate received packet by hashing it with session key and then validating. If session key used for encryption differs with session key for decryption, final hashes will differs.

Example of security design

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.

Example code is possible to check in git repository hosted on address:
https://github.com/rligocki/px4_libhydrogen_example/tree/b42faf885f858b6d955467f1295ef24e8e98557c

PX4 security implementation

Propose