PX4 Autopilot의 MAVLink 프로토콜 암호화*

이명진[O] 조진성
경희대학교 컴퓨터공학과
lumy0726@khu.ac.kr, chojs@khu.ac.kr

MAVLink protocol encryption for PX4 Autopilot

Myeongjin Lee[O], Jinsung Cho
Department of Computer Science and Engineering, KyungHee University

요 약

PX4 Autopilot 드론은 QGroundControl(지상제어국) 등의 구성 요소와 MAVLink 프로토콜을 사용하여 통신을 수행한다. 드론 사용은 계속 증가하고 있으므로 정보 탈취나 물리적 피해를 초래하는 악의적인 공격을 방어하는 것이 더욱 중요해졌다. 따라서 본 연구에서는 MAVLink 통신 프로토콜에 암호화 기능을 개발하고 이를 PX4와 QGC에 적용하는 연구를 진행하였다. 이를 통해 MAVLink의 전체 구조를 유지하면서 PX4와 QGroundControl이 AES128 CTR 암호화 방식을 이용하여 데이터를 송수신할 수 있다.

1. 서 론

PX4 Autopilot(이하 PX4)은 드론 제어용 오픈소스 소프트웨어로 많이 사용된다. 전 세계 드론 시장 규모는 2022년에 243억 9,000만 달러를 기록하였으며 2030년까지 약 5,045억 달러까지 성장할 것으로 예상된다[1]. 드론은 지형 촬영, 운반, 시설 관리, 농업, 개인 촬영, 군사 용도 등 여러 분야에 사용되고 있다. 이러한 드론이 악의적인 공격자에 의해 공격당할 경우 중요 정보가 탈취당하거나 물리적 피해가 발생할 수 있다. 드론 공격 피해를 예방하기 위한 보안 강화가 중요한 이유이다.

본 연구에서는 PX4 Autopilot에서 사용하는 MAVLink 통신에 암호화를 적용하는 연구를 진행하고 그 결과를 검증한다. PX4의 통신 대상으로는 지상제어국에 해당하는 QGC(QGroundControl)를 사용한다. 암호화는 MAVLink에서 주요 데이터를 담고 있는 payload를 대상으로 진행하였으며, MAVLink의 전체 구조를 유지하면서 암호화를 지원하도록 MAVLink 프로토콜을 수정한다. 이를 통해 기존 프로토콜 구조를 그대로 유지하면서 암호화된 MAVLink 통신을 수행할 수 있다.

통신에 암호화를 적용하면 악의적인 공격자가 통신 데이터를 가로채더라도 내용을 쉽게 확인할 수 없도록 하고, 데이터 변조를 통한 드론 공격을 예방할 수 있다. 이번 연구와 함께 다른 보안 기능도 같이 강화한다면 드론에 대한 공격은 쉽게 이루어지지 않을 것이다.

본 논문에서는 MAVLink에 대한 배경지식을 조사하고 소스 코드에서의 송수신 동작을 분석한 내용을 설명한다. 이후 MAVLink에 암호화 기능을 추가하기 위한 구현을 설명한다. 해당 기능의 검증을 위해 PX4와 QGC를 수정하여 MAVLink 암호화 기능을 적용하고 테스트를 수행하였으며 그 결과를 서술한다.

2. 배경 지식

2.1. MAVLink

2.1.1. MAVLink frame 구조[2]

MAVLink는 하나의 메시지를 frame으로 생성하여 전송한다. MAVLink의 버전은 v1.0와 v2.0이 있으며 그 frame 구조가 다르다. 현재는 대부분의 통신에 버전 v2.0이 사용된다.

[그림 1] MAVLink v2.0 frame

[표 1] MAVLink v2.0 C fixed header 일부[3]

파일명	주요 역할
protocol.h	STX 값 등 정의
mavlink_types.h	MAVLink frame, parsing 관련 구조체 정의
mavlink_helpers.h	MAVLink frame의 생성 및 parsing 기능

[그림 1]은 MAVLink frame v2.0의 구조이다. Frame 구조에서 stx는 시작을 인식하는 목적으로 사용되며 8bit의 len 값은 payload의 길이를 저장한다. Inc flags는 수신 측이 이해해야 하는 플래그가 저장되며 이 값을 이해하지 못하는 경우 frame은 폐기된다. 현재 inc flags는 signature 기능의 사용 유무로 총 8bit 중 1bit만 사용되고 있다. 반면 cmp flags는 수신 측이 이해하지 못해도 문제없는 플래그가 저장된다. 이 두 종류의 플래그는 v1.0에는 존재하지 않는다. Msg id가 메시지의 값, payload가 메시지의 실제 데이터를 저장하며, 이외 seq, sys id, comp id, checksum, signature 등이 존재한다.

2.1.2. MAVLink frame 관련 소스 코드

MAVLink는 여러 프로그래밍 언어로 이용할 수 있으며, 파이썬으로 작성된 mavgen 모듈을 통해 MAVLink 프로젝트를 빌드 시 필요한 언어로 구현된 소스 코드를 얻을 수 있다. PX4와 QGC는 C 언어로 구현된 MAVLink v2.0을 사용한다.

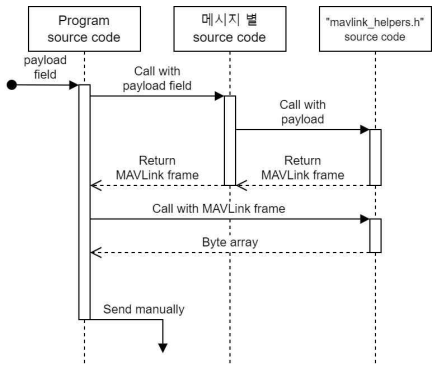
상단의 [표 1]은 mavgen을 통해 C 언어로 빌드된 MAVLink 소스 코드 목록 중 일부이며, 해당 파일은 미리 작성되어 있으므로 fixed header라고 지칭한다. [표 1]의 파일들은 MAVLink frame의 생성과 parsing에 있어서 중요한 역할을 한다. PX4와 QGC는 이들을 사용하여 MAVLink 통신을 수행한다.

2.1.3. MAVLink frame 생성과 parsing

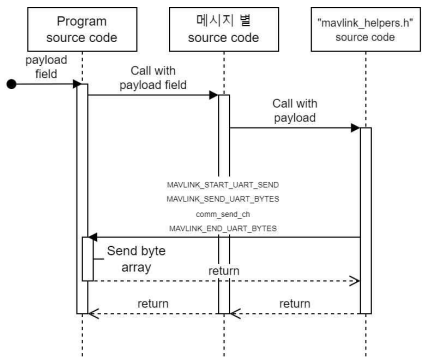
하단의 [그림 2]와 [그림 3]은 두 방식의 MAVLink frame 생성

---

* [CSE405-00] 졸업프로젝트 중간보고서



[그림 2] MAVLink frame 생성 방식 1



[그림 3] MAVLink frame 생성 방식 2

과정이다. [그림 2]에서 프로그램은 payload 데이터를 MAVLink에 전달하여 MAVLink frame 구조체를 얻어오고, 이를 MAVLink에 다시 전달하여 바이트 배열을 얻어온다. [그림 3]에서는 위 과정과는 달리 MAVLink가 구조체를 생성하지 않고 바이트 배열을 바로 생성하며, 프로그램이 구현한 함수를 직접 호출하여 frame을 송신하는 차이점이 있다. 두 방식 모두 "mavlink_helpers.h"에서 바이트 배열 형태로 완성된 payload로 MAVLink frame을 생성하는 과정이 있으며, 이를 ***finalize 과정***이라 지칭한다.

Parsing 과정에서 프로그램은 수신한 바이트 배열을 한 바이트씩 MAVLink로 넘겨주고 MAVLink는 해당 값을 이용하여 parsing을 진행한다. 먼저 STX 값을 읽어서 MAVLink frame의 시작을 인식하여 header 데이터를 읽고, header에 포함된 길이 정보를 이용하여 payload 데이터를 읽는다. 이때 현재 상태를 가지고 있기 위해 ***MAVLink status 구조체***(mavlink_status_t)를 사용한다. 이 구조체는 현재 parsing 진행도, 프로토콜 버전, 놓친 MAVLink frame 수 등의 정보를 가지고 있다. Parsing이 완료되면 MAVLink는 frame 데이터를 구조체로 변환하여 프로그램으로 전달한다.

3. 본론

3.1. MAVLink 암호화 구현

본 항목에서는 MAVLink를 암호화 지원 기능을 구현한 내용을 서술한다. 구현 위치는 MAVLink v2.0 C fixed header이다.

3.1.1. 암호화 지원 API

암호화를 지원하기 위해 우측의 [그림 4]와 같이 세 개의 함수를 선언하며, 이는 각각 암호화의 적용 유무 판단, payload 암호화 수행, 복호화 수행을 담당한다. PX4나 QGC 등의 프로그램은 이를 구현하여 암호화를 적용할 MAVLink frame을 결정하고 실제 암호화 알고리즘을 실행하여야 한다.

```
MAVLINK_HELPER uint8_t mavlink_mesl_crypto_condition(
    mavlink_status_t* status,
    uint32_t msgid,
    uint8_t system_id,
    uint8_t component_id,
    const char *payload,
    uint8_t len
    );
MAVLINK_HELPER int32_t mavlink_mesl_encrypt(
    uint8_t crypto_method,
    const char *src,
    char *dst,
    uint8_t len,
    uint8_t maxlen
    );
MAVLINK_HELPER int32_t mavlink_mesl_decrypt(
    uint8_t crypto_method,
    const char *src,
    char *dst,
    uint8_t len,
    uint8_t maxlen
    );
```

[그림 4] 구현한 MAVLink 암호화 지원 API

```
/*
  incompat_flags bits
 */
// #define MAVLINK_IFLAG_SIGNED  0x01
// #define MAVLINK_IFLAG_MASK    0x01 // mask of all understood bits
#define MAVLINK_IFLAG_SIGNED  0x01
#define MAVLINK_IFLAG_MESL_CRYPTO_METHOD 0xe0
#define MAVLINK_IFLAG_MASK    0xe1 // mask of all understood bits
```

[그림 5] 수정한 inc flags 정보2

암호화의 적용 유무를 판단하는 함수는 payload 데이터와 MAVLink status 구조체 등의 정보를 인수로 받아서 정수 값을 리턴한다. 리턴값이 0인 경우 해당 payload에 대해 암호화를 적용하지 않으며, 1에서 7 사이이면 각 정수에 따라서 정의된 암호화 방식을 사용한다. 암호화 및 복호화 함수는 암호화 방식, payload 데이터, 현재 길이, 최대 길이 정보 등을 인수로 받아서 암호화 혹은 복호화를 적용한 후 그 길이를 리턴한다. 따라서 payload의 길이는 변할 수 있으나, frame에 저장될 길이 정보는 8bit이므로 MAVLink의 구조를 유지하기 위해 최대 길이는 255로 제한된다.

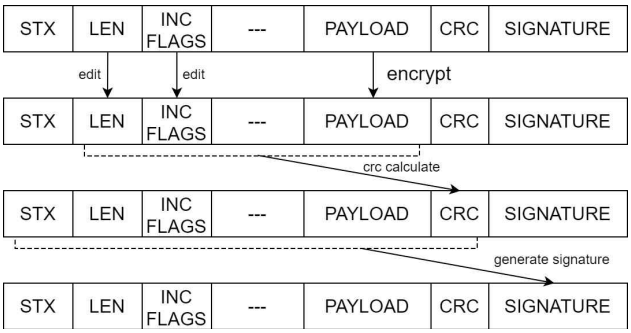
암호화 여부와 방식을 frame에 저장하기 위해 총 8bit의 inc flags 중 사용하지 않는 3bit를 이용한다. 따라서 암호화 방식으로 총 7개를 사용할 수 있다. 이를 위해 상단의 [그림 5]와 같이 inc flags 정보를 수정한다. 따라서 MAVLink는 parsing 과정에서 해당 플래그로 암호화 여부와 방식을 판단하고 복호화를 수행한다.

3.1.2. Frame 생성 시의 암호화

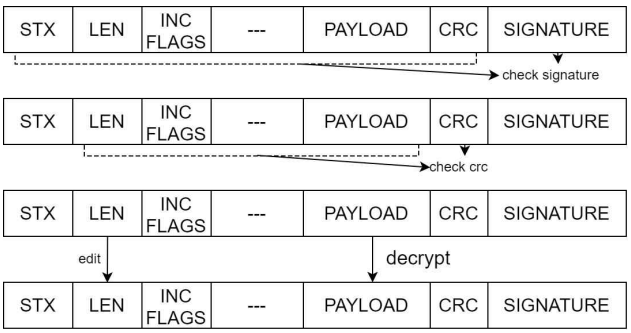
MAVLink frame은 생성 시 모두 finalize 과정이 이루어진다. 따라서 “mavlink_helpers.h"의 소스 코드에서 해당 과정을 수정하여 암호화 지원 API를 호출하는 코드를 작성하였다. 이때 하단의 [그림 6]와 같이 먼저 payload 데이터의 암호화가 먼저 이루어진 뒤 CRC 검사와 signature 동작이 수행되도록 하였다. 따라서 CRC 값이나 signature 값으로 payload 데이터가 간접적으로 노출될 가능성을 예방하고, 신호 간섭 등을 예방하기 위한 CRC 목적이 정상적으로 동작하도록 구현하였다.

3.1.3. Frame parsing 시의 복호화

MAVLink parsing은 "mavlink_helpers.h"의 소스 코드에서 "mavlink_frame_char_buffer" 함수에서 이루어진다. 이때 하단의 [그림 7]와 같이 CRC 검사와 signature 검사를 모두 마친 후 복호화가 이루어질 수 있도록 parsing 절차를 수정하고 복호화 지원 API를 호출하도록 작성하였다.



[그림 6] Frame 생성 시 암호화, crc, signature



[그림 7] Frame parsing 시 복호화, crc, signature

4. 검증

4.1. PX4 & QGC 암호화 구현

PX4와 QGC에서 암호화 구현을 추가한 MAVLink를 빌드하여 사용하도록 프로젝트 구조를 수정하고, 암호화 지원 API 함수 3개를 구현하여 상호 통신에 암호화를 적용할 수 있도록 하였다. 적용한 암호화 방식은 2가지로, 첫 번째는 테스트 용도의 간단한 XOR 연산이며 두 번째는 AES128 CTR 암호화이다. 이때 양측이 이미 대칭 암호화 키를 모두 가지고 있다고 가정하고 임의의 키를 사용하였다.

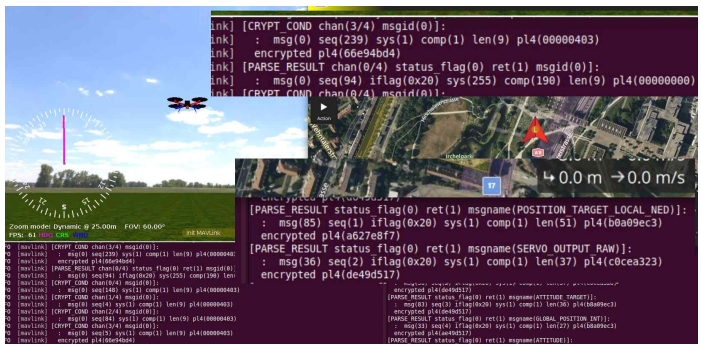
4.2. 암호화된 frame 통신 테스트

먼저 Ubuntu 22.04 환경에서 PX4, QGC, 시뮬레이터를 실행하고 테스트를 수행하였다. 그 결과 우측의 [그림 8]과 같이 암호화된 frame 통신과 드론 제어가 정상적으로 이루어짐을 확인하였다. [그림 8]의 우상단에서 PX4에서 16진수 66e9로 시작하는 payload를 받아서 복호화하였으며, 그 결과 payload 앞 4바이트가 0으로 시작하는 메시지를 받은 것을 확인할 수 있다. 우하단에서는 비슷하게 QGC에서 "SERVO_OUTPUT_LAW" 메시지를 복호화하여 받은 것을 확인할 수 있다.

또한 실제 드론에 사용되는 flight controller인 Pixhawk 6C mini 기기와 Ubuntu 22.04에서 실행되는 QGC를 USB를 통해 Serial 통신으로 연결한 뒤, 동일하게 암호화된 frame 통신이 정상적으로 이루어짐을 확인하였다.

4.3. 오버헤드 검사

암호화로 인한 오버헤드의 정도를 측정하기 위해, 시간 측정 코드를 삽입하여 Pixhawk 6C mini 기기의 MAVLink frame 송수신 시간을 측정하였고 우측 [표 2]의 결과를 얻었다. 측정된 frame의 수는 각각 250회 이상이며, 2초 이상 시간을 소모한 frame은 제외하였다. 그 결과 AES 암호화를 적용하기 전까지는 유의미한 시



[그림 8] 암호화된 MAVLink frame, SITL 환경

[표 2] MAVLink 암호화 오버헤드 테스트 결과

조건	송신 시간 (단위: us) 최소/평균/최대	수신 시간 (단위: us) 최소/평균/최대
시간 측정 코드만 삽입	0/26/1199	0/11/147
암호화 지원 기능 포함	0/27/1283	0/15/639
XOR 암호화 기법 적용	0/27/1158	0/13/163
AES 암호화 기법 적용	0/67/1330	0/35/308

간을 소모하지 않는다는 결과를 얻었다. AES 암호화 적용 전 frame 수신 시간은 편차가 있는데 이는 하나의 frame이 온전히 수신될 때까지 지연되는 것에 의한 증상이다. AES 암호화 적용 후에는 어느 정도 시간을 소모하였으나 비행 제어에 있어 문제가 될 수준의 오버헤드는 아님을 확인하였다.

5. 결론 및 보완점

드론 시장은 계속 확대되고 있으며 무인기에 대한 통신 공격을 막기 위한 여러 방면에서의 보안 강화가 필요하다. 따라서 본 논문에서는 MAVLink 통신 프로토콜에 대해 payload 암호화를 적용하는 연구를 진행하였다.

그 결과 MAVLink의 전체 구조를 유지하면서 inc flags를 이용하여 payload 암호화 지원 기능을 구현하고, PX4와 QGC 간 통신에 AES128 암호화를 성공적으로 적용하였다. 암호화된 MAVLink frame이 정상적으로 송수신되고 기체 제어가 정상적으로 동작하였다. 따라서 기존의 MAVLink 프로토콜을 크게 변경하지 않고도 payload가 암호화된 MAVLink 통신을 수행할 수 있음을 확인하였다.

본 연구를 통해 악의적인 공격을 한 단계 더 방어할 수 있으며 더 안전한 드론 체계를 만들 수 있다. 이와 함께 다른 보안 강화도 같이 수행한다면 더욱 효과적일 것이다.

본 논문에서는 MAVLink 프로토콜 버전 2.0의 C 언어를 대상으로 하였으므로 다른 언어에 대해서는 동일한 형태로 추가 구현하여 사용할 수 있다. 또한 inc flags가 없는 프로토콜 버전 1.0에 대해서는 암호화를 수행하지 않는 한계점이 있다. 향후 연구에서는 이를 보완할 계획이다.

[참고 문헌]

[1] 정보통신산업진흥원, 드론 시장동향 보고서 2023, 2023, p.6.
[2] https://mavlink.io/en/
[3] https://github.com/ArduPilot/pymavlink/tree/master/generator/C/include_v2.0