SCHOOL OF ADVANCED TECHNOLOGY

SAT301 FINAL YEAR PROJECT

Security Investigation towards MAVLink Protocol for UAV

Final Thesis

In Partial Fulfillment

of the Requirements for the Degree of

Bachelor of Engineering

Student Name	:	Zichao Cong
Student ID	:	2035606
Supervisor	:	Wenjun Fan
Assessor	:	Wanxin Li

Abstract

Unmanned Aerial Vehicles (UAVs), or drones, have become integral across diverse sectors such as economic, commercial, recreational, military, and academic areas. To operate effectively, UAVs require secure communication with network entities like ground control stations, other UAVs, and air traffic control systems. These cyber-enabled connections expose UAVs to significant threats, making secure communication crucial to protect against data leaks and mission compromises. This study focuses on the MAVLink protocol, a prevalent open-source framework used for UAV communications. Despite its design for availability and safety, MAVLink's security features have been relatively overlooked. This research conducts a thorough examination of MAVLink's architecture and simulates potential attacks to assess its vulnerability, utilizing ArduPilot's Software in the Loop (SITL) for field experiment simulation due to the absence of physical drones. Key findings include identified vulnerabilities such as spoofing, forgery and collision, with proposed enhancements like improved key distribution mechanisms and a recommended key replacement frequency for users to mitigate these risks. The outcomes are vital for enhancing UAV communication security and ensuring more robust protection in various applications. To more accurately predict the frequency of key replacement, we also plan to use GPU computing to improve the efficiency attacker could have in reality.

Keywords: Unmanned Aerial Vehicles (UAVs); Micro Air Vehicle Link (MAVLink) protocol; ArduPilot Software in the Loop (SITL); Vulnerability Analysis; Spoofing Attack; Forgery and Collision Attack

Acknowledgements

I am deeply grateful for the support and encouragement I have received from numerous individuals during the course of this research project.

Firstly, I would like to extend my heartfelt thanks to my project supervisor, Wenjun Fan, whose expertise and insightful guidance were instrumental in shaping both the direction and the success of this study. His patience and commitment to excellence have been a constant source of motivation. More importantly, he has greatly influenced and helped me in my cognitive education and my future planning.

I am also indebted to Xin Wu and Xiaocheng Zhou, my university development advisors, for their invaluable advice and support throughout this journey. The recommendations were crucial in navigating the academic challenges in the university and enhancing my personal growth.

Special thanks are due to my psychological consultant, whose guidance was essential in helping me maintain my well-being and focus during challenging times. His professional input has not only been beneficial in persisting with this final year project but has also helped me realize my strengths.

I cannot express enough thanks to my family and friends for their understanding and endless love through the inevitable ups and downs of this research process. Their unwavering support and encouragement have been my anchor and a constant source of strength.

This project would not have been possible without the collective support and encouragement from all these individuals. I am profoundly grateful to each one of them for their contributions to my academic journey.

Contents

List of Tables

List of Figures

List of Acronyms

Term	Initial components of the term
UAV	Unmanned Aerial Vehicle
UAS	Unmanned Aircraft Systems
MAVLink	Micro Air Vehicle Link
SITL	Software in the Loop
GCS	Ground Control Station
ECDH	Elliptic-curve Diffie-Hellman
MITM	Man in the Middle
SHA-256	Secure Hash Algorithm 256-bit
CRC	Cyclic Redundancy Check

Chapter 1

Introduction

1.1 Background

In recent years, Unmanned Aerial Vehicles (UAVs), commonly referred to as drones, have become indispensable across a variety of sectors, including search and rescue operations, agriculture, package delivery, cinematography, and military uses [1, 2, 3]. The proliferation of these devices is expected to continue, with a significant increase in the deployment of small UAVs, many of which rely on protocols that are not sufficiently secure [4]. This expansion is accompanied by a growing concern over their potential misuse. For example, incidents like the unauthorized UAV that landed on the roof of the Japanese prime minister's mansion carrying radioactive soil have highlighted the risks associated with their operation [5]. Such events underscore the critical need for robust security measures as UAVs become more embedded in essential and sensitive functions.

The reliance on inherently insecure command and control protocols by numerous small UAVs exacerbates the potential for breaches, making the development and implementation of secure communication systems a paramount concern. The communication between UAVs and their ground control stations, as well as inter-UAV communication, often utilizes the MAVLink protocol, a standard that, despite its widespread adoption for its efficiency and effectiveness in real-time command and control, offers insufficient security measures to safeguard against the spectrum of cyber threats these vehicles might encounter in operational environments.

1.2 Motivation

The diverse applications of UAVs pose unique security challenges. Specific threats include device capture attacks that could expose sensitive information and cryptographic keys,

severely compromising personal and national security [1]. The complexity of implementing robust cryptographic defences is amplified by the limited computational capabilities typical of many UAVs. Moreover, UAVs' reliance on wireless communication networks makes them particularly vulnerable to cyberattacks aimed at intercepting crucial telemetry and control information [5]. Previous studies have largely focused on general vulnerabilities within wireless networks, such as denial-of-service attacks, Telnet/FTP intrusions, ARP spoofing, and Man-in-the-Middle attacks [7]. However, there is a notable deficiency in research specifically addressing the security vulnerabilities inherent to UAV-specific communication protocols, such as MAVLink [8]. This protocol's lack of strong encryption and authentication mechanisms makes it an attractive target for malicious actors aiming to disrupt or control UAV operations illegally. Despite its utility, MAVLink version 1 is susceptible to security vulnerabilities, posing risks to UAV operations. Even though MAVLink version 2 adds support for message signing to verify that messages originate from a trusted source, there exists possible attacks to this function. Addressing these security gaps is essential for advancing UAV technology and ensuring their safe integration into the airspace.

1.3 Objective

This research aims to conduct a detailed analysis of the MAVLink protocol, identifying and addressing its security vulnerabilities, to enhance the safety and reliability of UAV communications. By focusing on MAVLink version 2 and deploying on Ardupilot framework to simulate a real UAS system, this study will explore potential security flaws and develop countermeasures to mitigate these risks effectively. The goal is to propose viable security enhancements that will safeguard against unauthorized access and control, ensuring that UAVs can perform their critical functions without compromise. This work is crucial for advancing UAV security standards and supporting their expanding role in society.

The remainder of the paper is structured as follows. Chapter 2 discusses related work that improves the trust and security of UAV communication. Chapter 3 describes the experiment's approach, while Chapter 4 goes into detail on the implementation procedure for attacking the vulnerabilities. The findings are presented and discussed in Chapter 5. Finally, Chapter 6 summarises the report and discusses future employment prospects.

Chapter 2

Literature Review

2.1 UAV Related Work

In the literature, extensive research has been conducted to enhance the trustworthiness and security of UAV communications. A trust channel model for UAV air-to-ground communication using multiple-input-multiple-output (MIMO) was introduced in [9]. The work by Su et al. [10] focused on a UAV trust evaluation method, aimed at boosting the reliability and security of communications. This method utilizes a trust threshold to isolate malicious UAVs, thus safeguarding the system's security. Yoon et al. [11] developed a novel concept involving an encrypted communication channel for UAV systems in compromised environments, utilizing Raspberry Pi to bolster data security. In another study, Alladi et al. [12] introduced a lightweight protocol for mutual authentication between UAVs and ground stations, alongside a scheme for UAV-to-UAV authentication. Li et al. [13] proposed a resilient trust management system designed to counteract malicious attacks and assess the trust levels of mobile nodes. While previous vulnerability analyses [14-17] predominantly addressed common wireless network vulnerabilities, they did not specifically focus on the unique technologies employed in commercial Unmanned Aerial Systems (UASs). Dahiya et al. [18] explored various vulnerabilities within UASs and recommended preventive measures at different levels. Their findings indicated that the most accessible point of attack is through flooding the UAV via radio communication, with the most severe vulnerability being the complete takeover of control through Man-in-the-Middle attacks. In this paper, a vulnerability analysis and attack on UASs using the Ardupilot framework and MAVLink protocol are performed. They represent the technology most commonly employed in UASs.

2.2 MAVLink Related Work

The standard UAS consists of two main components: the drone and the Ground Control Station (GCS). The GCS is responsible for planning and remotely commanding the drone, which has the computational capacity to complete its own missions autonomously. The MAVLink protocol is the most often used protocol for communication between GCS and Drones.

The MAVLink protocol is divided into MAVLink 1.0 and the newer MAVLink 2.0 [8, 19]. In [3], the paper introduced the message structure of MAVLink 1.0. In this current paper, we unveil the MAVLink 2.0 message structure and conduct a comparative analysis of its resemblances and disparities when compared to the MAVLink 1.0 protocol.

Fig. 1 illustrates the structure of the MAVLink 2.0 message packet, comprising a total of 11 fields. The initial field, denoted as STX, represents the symbol at the commencement of the MAVLink message packet, with the distinctive symbol for STX in MAVLink 2.0 being 0xFD. The second field, LEN, specifies the payload's length in bytes, ranging from 0 to 255. The third and fourth fields, INC FLAGS (Incompatibility flags) and CMP FLAGS (Compatibility flags) respectively, have distinct purposes. The former signifies features that a MAVLink library must support for packet handling. When the INC FLAGS value is 0x01, it indicates that a signature has been appended to the message, signifying its being marked. The latter, CMP FLAGS, pertains to compatibility flags that do not impact the message packet's structure. Even if the flag's content cannot be interpreted, it does not affect packet processing.

Fig. 1. Structure of the MAVLink 2.0 message packet

The fifth field, SEQ, denotes the message sequence number, within the same value range as LEN. Upon reaching 255, the sequence number resets. The sixth field, SYS ID, represents the system ID, with each device possessing its unique system ID, and the ground control station typically assigned the ID 255. The seventh field, COMP ID, refers to the system component sending the message. The eighth field, MSG ID, symbolizes the

message types in the payload. When MSG ID is 0, it designates the message packet as a heartbeat packet, confirming the operational status between the ground control station and the UAV. The payload follows MSG ID, storing command information indicated by MSG ID, with a maximum capacity of 255 bytes. CHECKSUM appears at the message packet's end to ensure the received message's correctness. SIGNATURE, an optional 13-byte signature field at the message packet's end, is related to INC FLAGS.

Comparing it with the MAVLink 1.0 protocol, MAVLink 2.0 inherits all its fields, introduces INC FLAGS and CMP FLAGS, and alters the size of certain fields. The combination of INC FLAGS and SIGNATURE enables message authentication, guarding against tampering with the link. These enhancements bolster the security of the MAVLink protocol. CMP FLAGS may lead to more rational message packet processing order. Furthermore, it expands MSG ID's capacity, accommodating a greater number of message types, up to 16,777,215. MAVLink 2.0 message packets range from a minimum length of 11 bytes to a maximum of 279 bytes. The size of the payload is variable; it is determined by the parameters supplied or received during communication.

MAVLink 2 message signing, allowing an MAVLink system to ensure that communications come from a trusted source, is the protocol’s only real protection against message spoofing. When the user of the ground control station wants to activate the message signing function, the user should first input a key, and then this key is hashed by SHA-256. The hashed value is the secret key to be shared between the two systems, and this process is the key generation process.

Then, this secret key is distributed by the SETUP_SIGNING message, whose payload is shown in Table 1, and this process is the key distribution process.

Field Name	Type	Description
target_system	uint8_t	system ID of the target
target_component	uint8_t	component ID of the target
secret_key	uint8_t[32]	signing key

initial_timestamp	uint64_t	initial timestamp

Table 1. SETUP_SIGNING Message Payload

After the key is generated and distributed, the signature will be calculated and added to the end of each message. To calculate, the hashed key is first serially connected to each message and then hashed again for a signature. In reality, the first 48 bits of this signature are appended to the packet to be sent. The receiver computes the signature and compares it with the signature to verify authentication.

The function to generate signatures is shown below, where || represents concatenation and the timestamp is a 48-bit number with units of 10 microseconds since January 1, 2015 GMT.

signature = sha256(secret key || header || payload || CRC || link-ID || timestamp)

Fig. 2. Message with Signature

Regarding the safety aspect, the MAVLink protocol refrains from encrypting message packets to enhance transfer efficiency, despite potential security risks. The protocol relies on checking specific key fields within message packets for classification. When a message is encrypted, the system may fail to recognize it due to alterations in the message packet's value, leading to additional time overhead when decrypting. Consequently, the MAVLink protocol does not incorporate encryption. Additionally, the protocol does not verify the source of message packets, potentially leaving an opening for attackers to gain control of the UAV.

Chapter 3

Methodology

3.1 Approach

The approach of this research is designed to meticulously identify and exploit vulnerabilities within the MAVLink protocol. By integrating theoretical frameworks with practical attack simulations, the methodology addresses both the 'what' and the 'why' of each chosen method. Each vulnerability, such as the key distribution weakness and signature forgery, was selected based on a comprehensive analysis of the protocol's architecture, prioritizing areas where breaches could have the most severe impact on UAV operations. This strategic focus ensures that the research not only tests MAVLink's security but also contributes to its enhancement by proposing viable and tested solutions.

3.1.1 Vulnerability 1: During key distribution, plaintext secret key could be eavesdropped if using unsecured channel.

Such a vulnerability is caused by two features that Mission Planner does not sign messages by default (when open the software) and the incorrect implementation of the SETUP_SIGNING message.

3.1.2 Vulnerability 2: Packet Injection with signature broken by spoofing or forgery and collision

Such a vulnerability is caused by the fact that MAVLink signature uses a prefix of a Hash function and the signature which is the only ciphertext part in the MAVLink message frame is obtained by only the first 48 bits to generate a whole package.

3.2 Attack Scenarios

3.2.1 Eavesdropping

The attacker station was able to sniff the communication among the GCS and the UAV during the startup phase. If the SETUP_SIGNING message is captured by the attacker, the plaintext secret key will be known by the attacker.

3.2.2 Brute Force (Imaginary Situation)

Suppose that the attacker eagers to get the secret key already shared by the UAV and GCS, we first simply imagine that all 256 bits of the signature is visible. The attacker can iterate all possible secret keys and calculate the hash value. Once the value is same to the signature, that secret key is the correct key.

With the need to iterate and calculate hash values, the worst time complexity is $O(2^{256})$ and average is $O(2^{255})$

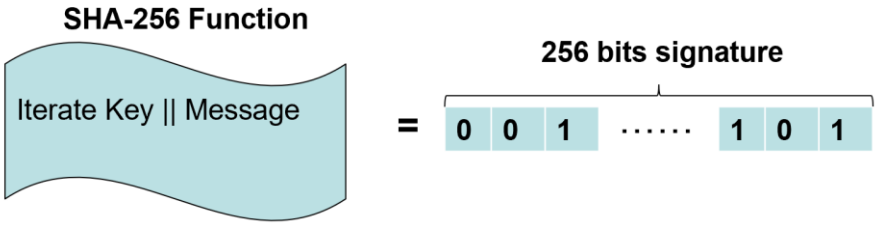

Fig. 3. Scenario 1 – All signature bits visible

3.2.3 Brute Force and Forgery and Collision

Suppose that the attacker also eagers to get the secret key already shared by the UAV and GCS, there are two steps that the attacker can finally reach the goal: the attacker should get all possible keys from the first one packet and then fix the certain key by getting another packet afterwards. In this scenario, we assume that n bits of signature are hidden.

As a result, the attacker can first list all possible keys that the first 256-n bits of hash value is same as the signature, mathematically, there are 2^n possible keys which could be regarded as overhead. Then, the attacker attempts to use the key in another packet to verify the correctness. The best situation is that the first attempt is correct and the worst is that the final one is the correct key so the attempt times is 2^n. As a result, the average attempt times is 2^{n-1}.

With the need to iterate and calculate hash values, the worst time complexity is $O(2^{256}+2^n)$ and best is $O(2^n+1)$, so average is $O(2^{255}+2^n)$

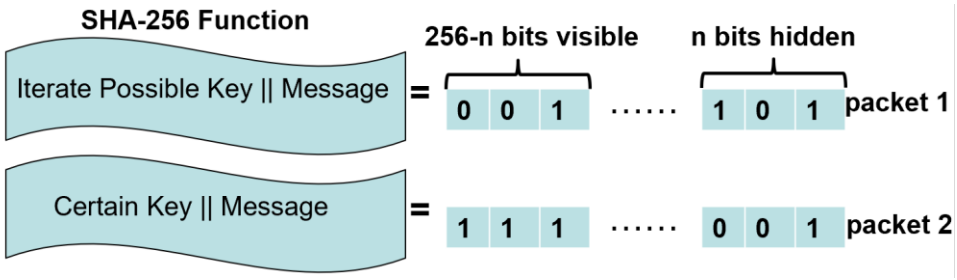

Fig. 4. Scenario 2 – n bits of signature hidden

3.2.4 Spoofing

Suppose that the attacker no longer needs a specific secret key and only want to make an illegal packet pass verification. We also assume that n bits of signature are hidden in this scenario.

The attacker can calculate all possible signatures because there only exist 2^{256-n} types of signatures. Then, the attacker can add each type of signature to the illegal packet and one packet can pass the verification.

As a result, what the attacker has to do is only to list all the signatures and modify the message packets. So, the time complexity is $O(2^{256-n})$ ($n \leq 255$).

Fig. 5. Scenario 3 – All signature combinations when n bits of signature hidden

3.3 Possible Solutions

3.3.1 For vulnerability 1: Use Proper Key Distribution Algorithm

When communicating in unsecured network, instead of sending packets with plaintext secret key to be shared with, it is more secured to use complex key distribution algorithms to avoid sending the key directly. It could help use Diffie-Hellman or advanced versions to fix this vulnerability.

3.3.2 For vulnerability 2: Change the key before a successful attack

Calculate the time complexity of different attacks for all scenarios, there exists a minimum time that the attacker has to spend for a successful attack. It could help to fix this vulnerability that the user changes the key before the recommended time.

3.4 Implementation

In order to perform our analysis, we prepared a testbed that reproduces the main components of a real UAS, made of the ArduPilot Mission Planner as the GCS and of a simulated drone generated by ArduPilot SITL [20]. Figure 6 below shows the architecture of the architecture of ArduPilot SITL. The SITL (software in the loop) simulator enables users to operate UAVs without using any hardware. It is a build of the autopilot code using a regular C++ compiler, which results in a native executable that allows the user to evaluate the code's behaviour without the usage of hardware.

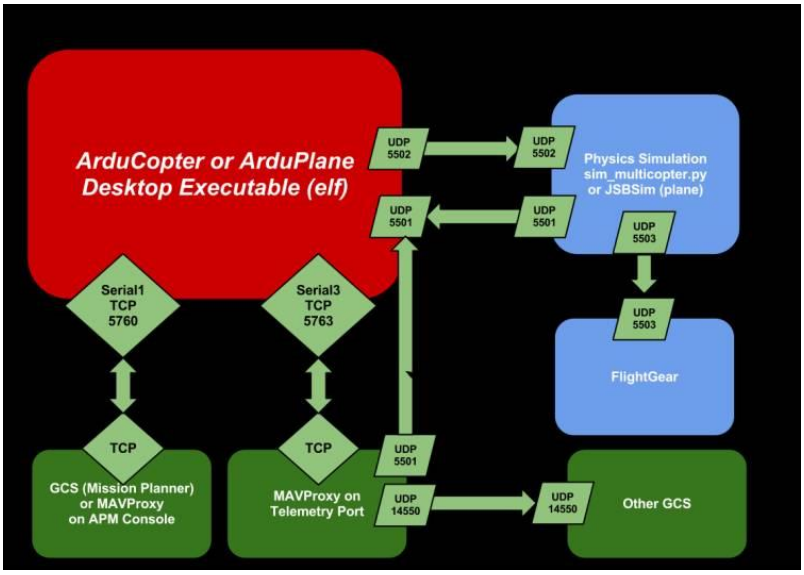

Fig. 6. ArduPilot SITL Architecture

Figure 7 below summarises the major components of our testbed. The ArduPilot SITL is deployed on the Ubuntu 22.04 virtual machine which owns IP as 172.27.116.7 as UAV and the Mission Planner software is deployed on the local host Windows 10 which owns IP 172.27.116.1 as GCS. In addition, an attacker is also deployed on the local host using Wireshark with the MAVLink plugin to analyse intercepted messages. The plugin needs to be added following the guidance from the official website. Both systems were connected via a local network, with network traffic monitored and manipulated using Wireshark equipped with a MAVLink plugin. This setup was crucial for capturing and analysing MAVLink messages exchanged between the UAV and GCS.

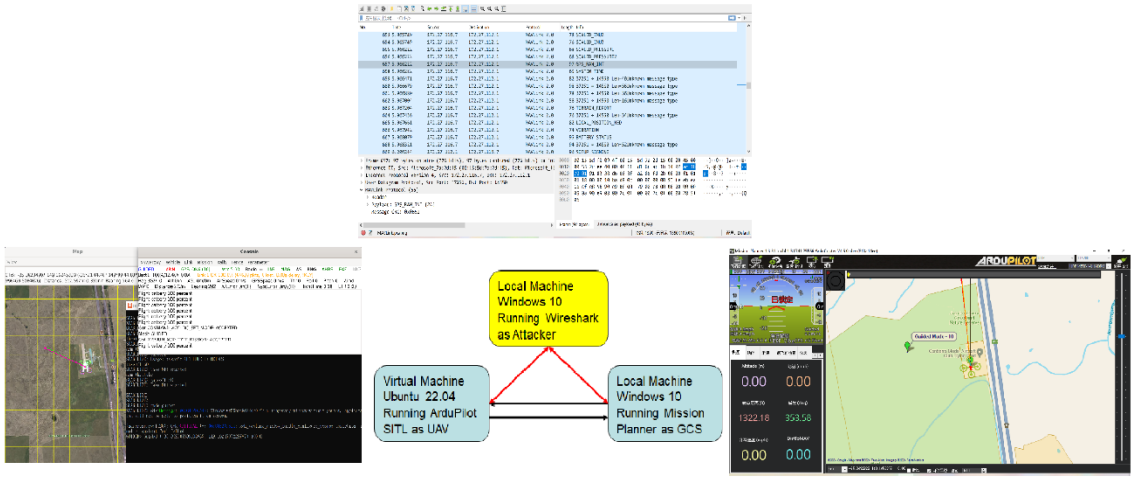

Fig. 7. MAVLink Analysis Testbed

To address the vulnerabilities identified in MAVLink's key distribution process, we implemented an Elliptic-curve Diffie-Hellman (ECDH) algorithm for secure key exchange. This modification was intended to enhance the security of the communication channel by preventing unauthorized eavesdropping and interception of the keys. Modifications were made to the MAVLink protocol's source code to integrate ECDH for key exchange. This involved altering the common.xml MAVLink definition to include new message types that facilitate the exchange of ECDH public keys. Python scripts were developed to automate the generation and exchange of keys. These scripts handled the encryption and decryption processes and were essential for validating the efficacy of the ECDH implementation.

Chapter 4

Results and Discussion

4.1 Results

4.1.1 Eavesdropping

By implementing the Wireshark with plugin, the secret key sent in plaintext is successfully captured by the attacker like shown in Fig. 8.

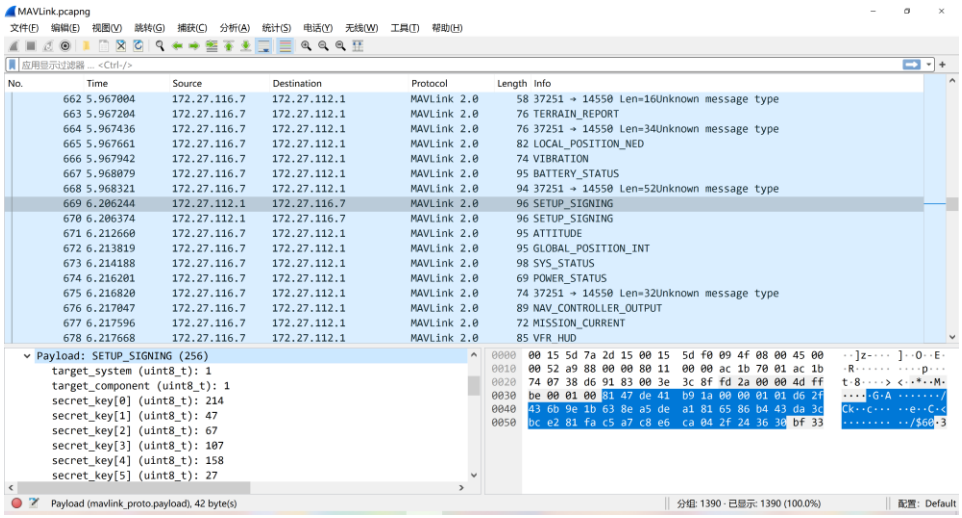

Fig. 8. Plaintext secret key in hexadecimal expression

4.1.2 Attack to Vulnerability 2 Results

Shown in Figure 9, when n = 1, the time consuming for Scenario 2 and 3 are similar. When n > 1, it is clear that Scenario 3 works better than Scenario 2.

As a result, the spoofing attack is mostly valued compared with brute force and forgery and collision attack.

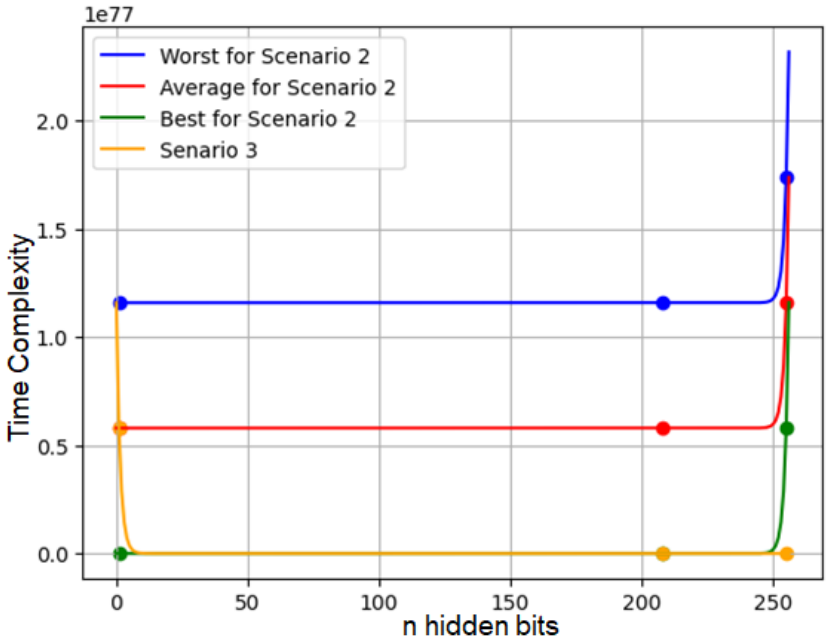

Fig 9. Comparison between Scenario 2 and Scenario 3

As the computation changes based on different equipment, the time cost for the attack varied. In the Table 2-4 below shows three types of facilities that performed differently doing the attacks. Based on the normal computation, even the fastest attack in reality would spend thousands of years. Such kind of time occupation can be regarded as failed attack. Therefore, it is necessary to use faster computing methods like GPU and parallel computing to speed up.

The recommended key change frequency should be the minimum time for the attacker to attack based on the improved experiment.

attampts	time	computation (times per sec)
1000	0.025606871	39052.01903
1000	0.026183128	38192.53324
1000	0.02620697	38157.78748
1000	0.024331093	41099.67468
1000	0.02366209	42261.69317
10000	0.293772936	34039.89537
10000	0.292436838	34195.41828
10000	0.320053101	31244.81526
10000	0.305357933	32748.45327
10000	0.329926968	30309.73816
100000	2.96291995	33750.48995
100000	2.938405991	34032.05695
100000	2.998769045	33347.01623
100000	2.976176977	33600.1524
100000	2.949410915	33905.07558

Table 2. iMAC-6Core-i5-8500@3.0GHz

attampts	time	computation (times per sec)
1000	0.430999517	2320.188213
1000	0.437009096	2288.281889
1000	0.459000111	2178.648712
1000	0.459000111	2178.648712
1000	0.42399931	2358.494407
10000	4.363968134	2291.492443
10000	4.268000364	2343.017607
10000	4.325996637	2311.60605
10000	4.385997772	2279.982918
10000	4.26499939	2344.666221
100000	43.4945817	2299.136952
100000	43.32680917	2308.039801
100000	44.80906057	2231.691509
100000	44.6207521	2241.109692
100000	44.1892817	2262.992204

Table 3. Dell 5060-6Core-i5-8500@3.0GHz

attampts	time	computation (times per sec)
1000	0.601662397	1662.061655
1000	0.67192626	1488.258548
1000	0.612790585	1631.878859
1000	0.712426424	1403.65372
1000	0.782908678	1277.288179
10000	8.797591925	1136.674682
10000	8.990038395	1112.342302
10000	9.286863804	1076.789777
10000	9.622730017	1039.206128
10000	9.079746246	1101.352365
100000	102.7883003	972.8733692
100000	103.0754337	970.1632715
100000	107.3341348	931.6700616
100000	115.0022221	869.5484157
100000	121.4510829	823.3767668

Table 4. T14-8Core-i7-10510U@1.8GHz

Chapter 5

Conclusion and Future Work

5.1 Conclusion

The research conducted thoroughly investigates the vulnerabilities of the MAVLink protocol, particularly focusing on its application within both MAVLink 1.0 and 2.0 versions. Despite improvements in MAVLink 2.0, such as message signing functionalities intended to verify the origins of communications, significant security gaps were identified that could be potentially exploited by cyber attackers.

The study revealed critical vulnerabilities, primarily during the key distribution phases, where plaintext keys transmitted over unsecured channels are susceptible to eavesdropping. These findings underscore the inherent security risks that could compromise the integrity and confidentiality of UAV communications.

To mitigate these risks, the research proposed implementing Elliptic-curve Diffie-Hellman (ECDH) for key exchanges to enhance security during the key distribution phases. Additionally, it was recommended that cryptographic keys be updated regularly to limit the window of opportunity for attackers to exploit old or stale keys.

5.2 Future Work

Looking forward, the dissertation suggests further exploration into developing more robust key distribution mechanisms that are resistant to Man in the Middle (MITM) attacks. It also recommends investigating the integration of more computationally intensive security algorithms and assessing their feasibility within the MAVLink protocol using higher computational resources. This continuous assessment of security measures is essential as new vulnerabilities emerge, ensuring the ongoing safety and effectiveness of UAV operations across various sectors.

References

[1] M. Gharibi, R. Boutaba, and S. L. Waslander, “Internet of drones,” IEEE Access, vol. 4, pp. 1148-1162, Mar. 2016.

[2] S. Waharte and N. Trigoni, “Supporting search and rescue operations with UAVs,” in Proc. Int. Conf. Emerg. Secur. Technol., Canterbury, U.K., pp. 142-147, Sep. 2010.

[3] Y. M. Kwon, J. Yu, B. M. Cho, Y. Eun and K. J. Park, “Empirical Analysis of MAVLink Protocol Vulnerability for Attacking Unmanned Aerial Vehicles,” IEEE Access, vol. 6, pp. 43203-43212, Aug. 2018.

[4] Eddy Deligne, “ARDrone Corruption,” Journal in Computer Virology, vol.8, pp.15-27, 2012.

[5] R. Altawy and A. M. Youssef, “Security, Privacy, and Safety Aspects of Civilian Drones: A Survey,” ACM Transactions on Cyber-Physical Systems archive, vol. 1, no. 2, Feb. 2017.

[6] S. Vemi and C. Panchev, “Vulnerability Testing of Wireless Access Points Using Unmanned Aerial Vehicles (UAV),” the European Conference on e-Learning, 2015.

[7] N. M. Rodday, R. d. O. Schmidt and A. Pras, “Exploring Security Vulnerabilities of Unmanned Aerial Vehicles,” the IEEE/IFIP Network Operations and Management Symposium (NOMS), pp. 993-994, 2016.

[8] MAVLink protocol, 2009, [online] Available: https://mavlink.io/en/.

[9] H. Jiang, Z. Zhang, L. Wu and J. Dang, “Three-Dimensional Geometry-Based UAV-MIMO Channel Modelling for A2G Communication Environments,” IEEE Communications Letters, vol. 22, no. 7, pp. 1438-1441, Jul. 2018.

[10] Yu Su, Jian Zhou, and Zhinuan Guo, "A Trust-Based Security Scheme for 5G UAV Communication Systems," IEEE Int. Conf. on DASC/PiCom/CBDCom/CyberSciTech, Aug. 2020.

[11] K. Yoon, D. Park, Y. Yim, K. Kim, S. Kai Yang, and M. Robinson, “Security Authentication System Using Encrypted Channel on UAV Network,” the 1st IEEE Int. Conf. on Robotic Computing (IRC), pp. 393-398, 2017.

[12] T. Alladi, Naren, G. Bansal, V. Chamola, and M. Guizani, “SecAuthUAV: A Novel Authentication Scheme for UAV-Ground Station and UAV-UAV Communication,” IEEE Transactions on Vehicular Technology, vol. 69, no. 12, pp. 15068-15077, 2020.

[13] X. Xu, Y. Zeng, Y. L. Guan and R. Zhang, “Overcoming Endurance Issue: UAV-Enabled Communications with Proactive Caching,” IEEE Journal on Selected Areas in Communications, vol. 36, no. 6, pp. 1231-1244, June 2018.

[14] A. Abdallah, M. Z.Ali, J. Miˇsi´c, andV.B. Miˇsic, “Efficient security scheme for disaster surveillance UAV communication networks,” Information, vol. 10, no. 2, 2019.

[15] C. Esposito, M. Ficco, A. Castiglione, F. Palmieri, and A. De Santis, “Distributed group key management for event notification confidentiality among sensors,” IEEE Transactions on Dependable and Secure Computing, vol. 17, no. 3, pp. 566-580, Jun. 2020.

[16] G. K. Verma, B. Singh, N. Kumar, and D. He, “CB-PS: An efficient short-certificate-based proxy signature scheme for UAVs,” IEEE System Journal, vol. 14, no. 1, pp. 621–632, Mar. 2019.

[17] M. Wazid, A. K. Das, N. Kumar, A. V. Vasilakos, and J. J. Rodrigues, “Design and analysis of secure lightweight remote user authentication and key agreement scheme in Internet of Drones deployment,” IEEE Internet Things Journal, vol. 6, no. 2, pp. 3572–3584, Apr. 2018.

[18] S. Dahiya and M. Garg, “Unmanned Aerial Vehicles: Vulnerability to Cyber Attacks,” the Int. Conf. on Unmanned Aerial System in Geomatics, pp. 201-211, Apr. 2019.

[19] L. M. A. Tridgell and L. Meier, MAVLink 2.0 packet signing proposal, 2015, [online] Available: https://mavlink.io/en/.

[20] ArduPilot SITL, 2024, [online] Available: https://ardupilot.org/dev/docs/sitl-simulator-software-in-the-loop.html

Appendix A. Attack Code

Normal brute force and collision for scenario 2.

```
import hashlib
import time
import sys
import itertools
import string
import base64

def brute_force_sha256(signature48bit, header, payload, crc, linkid,
timestamp):
    start = time.time()
    attempts = 0
    for key in itertools.product(string.ascii_lowercase + string.digits,
repeat=64):
        key = ''.join(key)
        hashed_key = hashlib.sha256(key.encode('utf-8')).hexdigest()
        complete_signature = hashed_key + header + payload + crc + linkid
+ timestamp
        if
(hashlib.sha256(bytes.fromhex(complete_signature)).hexdigest())[:12] ==
signature48bit:
            attempts += 1
            print("Key found:", key)
            print("Time taken:", time.time() - start)
        else:
            # print('Attempt Signature: ',
hashlib.sha256(bytes.fromhex(complete_signature)).hexdigest()) # For
hardcode testing
            attempts += 1
        if attempts == 10000:
            print('10000 attempts made in', time.time() - start)
            break

if __name__ == '__main__':
    seed = "MAVLink"
    header = input("Enter Header: ")
    payload = input("Enter Payload: ")
    crc = input("Enter CRC: ")
    linkid = input("Enter Link ID: ")
```

```
    timestamp = input("Enter Timestamp: ")

    hashed_seed = hashlib.sha256(seed.encode('utf-8')).hexdigest()

    # hex_seed = bytes.fromhex(hashed_seed)
    # base64_bytes = base64.b64encode(hex_seed)
    # base64_seed = base64_bytes.decode('ascii')

    signaturecompleta = hashed_seed + header + payload + crc + linkid +
timestamp
    # signature =
hashlib.sha256(bytes.fromhex(signaturecompleta)).hexdigest()
    signature =
'bc320e47f5896bb25f00615916493d332065e93ef8d5b39a9b7b2e617241eaa1' #
hardcoded signature
    signature48bit = signature[:12]

    brute_force_sha256(signature48bit, header, payload, crc, linkid,
timestamp)
```

Parallel Computing to brute force and forgery and collision for scenario 2.

```
import hashlib
import time
import sys
import itertools
import string
from concurrent.futures import ThreadPoolExecutor

seed = "MAVLink"
secret_key = hashlib.sha256(seed.encode('utf-8')).hexdigest()
hashed_secret_key = hashlib.sha256(bytes.fromhex(secret_key)).hexdigest()
signature = hashed_secret_key[:12]

def check_key(key):
    hashed_key = hashlib.sha256(key.encode()).hexdigest()
    if hashed_key[:12] == signature:
        return key
    else:
        return None

def brute_force_parallel():
```

```
    start = time.time()
    attempts = 0
    with ThreadPoolExecutor() as executor:
        for key in itertools.product(string.ascii_lowercase +
string.digits, repeat=64):
            key = ''.join(key)
            future = executor.submit(check_key, key)
            if future.result():
                print("Found key:", future.result())
                print("Time taken:", time.time() - start)
                break
            attempts += 1
            if attempts % 100000 == 0:
                print("Attempts:", attempts)

if __name__ == '__main__':
    brute_force_parallel()
```

Appendix B. Changed source code for ECDH

Changes in common.xml

```
    <message id="66666" name="ECDH_KEY_EXCHANGE">
        <description>Message to exchange ECDH public keys for secure
communications.</description>
        <field type="uint8_t" name="target_system">system id of the
target</field>
        <field type="uint8_t" name="target_component">component ID of the
target</field>
        <field type="uint64_t" name="pub_key">Public key</field>
    </message>
```

ECDH-Client.py

```
import socket
from collections import OrderedDict
from ecc.elliptic import mul,add,neg
from ecc.Key import Key


DOMAINS = {
    # Bits : (p, order of E(GF(P)), parameter b, base point x, base point
y)
    256:
(0xffffffff00000001000000000000000000000000ffffffffffffffffffffffff,
          0xffffffff00000000ffffffffffffffffbce6faada7179e84f3b9cac2fc6325
51,
          0x5ac635d8aa3a93e7b3ebbd55769886bc651d06b0cc53b0f63bce3c3e27d260
4b,
          0x6b17d1f2e12c4247f8bce6e563a440f277037d812deb33a0f4a13945d898c2
96,
          0x4fe342e2fe1a7f9b8ee7eb4a7c0f9e162bce33576b315ececbb6406837bf51
f5)
}

if __name__ == '__main__':

    global p,n,b,x,y,c_p,c_q,c_n

    server_ip = "192.168.111.144"
```

```
server_port = 6633

# initialization
p, n, b, x, y = DOMAINS[256]
c_p = 3
c_n = p
c_q = p - b
G = (x,y)

token=0


# TCP connection to responder B
sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
sock.setblocking(1)
print('begin connection')
sock.connect((server_ip, server_port))

try:
    while (token==0):
        print('connection up')
        print ('connected')

        # 1. A side:
        #1.1) generate x_a=SKa, h1=PKa*G using Key.generate(bit)
        keypair = Key.generate(256)
        SKa = keypair._priv[1]
        PKax = keypair._pub[1][0]
        PKay = keypair._pub[1][1]
        PKa = (PKax,PKay)

        # 1.2) A->B: M1=(PKa)
        M1=str(PKa[0])+','+str(PKa[1])
        sock.send(M1.encode())

        # 3. A side: receive M2 from B
        M2 = sock.recv(1024).decode()
        PKbx = M2.split(',')[0]
        PKby = M2.split(',')[1]
        PKb = (int(PKbx),int(PKby))

        # 4 A side: compute shared secret k1=SKa*PKb
        k1=mul(c_p,c_q,c_n,PKb,SKa)
```

```
            print ('A side: the shared secret is', k1)

            token=1

    except KeyboardInterrupt:
        s.close()
        print("KeyboardInterrupt")
```

ECDH-Server.py

```
import socket
from collections import OrderedDict
from ecc.elliptic import mul,add,neg
from ecc.Key import Key


DOMAINS = {
    # Bits : (p, order of E(GF(P)), parameter b, base point x, base point
y)
    256:
(0xffffffff00000001000000000000000000000000ffffffffffffffffffffffff,
          0xffffffff00000000ffffffffffffffffbce6faada7179e84f3b9cac2fc6325
51,
          0x5ac635d8aa3a93e7b3ebbd55769886bc651d06b0cc53b0f63bce3c3e27d260
4b,
          0x6b17d1f2e12c4247f8bce6e563a440f277037d812deb33a0f4a13945d898c2
96,
          0x4fe342e2fe1a7f9b8ee7eb4a7c0f9e162bce33576b315ececbb6406837bf51
f5)
}

if __name__ == '__main__':

    global p,n,b,x,y,c_p,c_q,c_n

    HOST = '192.168.111.144'
    PORT = 6633

    # initialization
    p, n, b, x, y=DOMAINS[256]
    c_p=3
    c_n=p
```

```
c_q=p-b
G=(x,y)

token=0

print('Begin')

#TCP link
sock = socket.socket(socket.AF_INET,socket.SOCK_STREAM)
sock.bind((HOST,PORT))

print('Listen to the connection from client...')
sock.listen(5)
try:
    while (token==0):
        connection, address = sock.accept()
        print('Connected. Got connection from ', address)

        # 2. B side:
        # 2.1) receive M1=(PKa) from B
        M1=connection.recv(1024).decode()
        PKax=M1.split(',')[0]
        PKay=M1.split(',')[1]
        PKa=(int(PKax),int(PKay))

        # 2.2) generate y_b=SKb, h2=PKa=SKb*G using Key.generate(bit)
        keypair = Key.generate(256)
        SKb = keypair._priv[1]
        PKbx = keypair._pub[1][0]
        PKby = keypair._pub[1][1]
        PKb = (PKbx,PKby)

        # 2.3) B->A: M2=(PKb)
        M2=str(PKb[0])+','+str(PKb[1])
        connection.send(M2.encode())

        # 4. B side:compute shared key k2=SKb*PKa
        k2=mul(c_p,c_q,c_n,PKa,SKb)
        print('B side: the shared secrety is', k2)

        token=1

except KeyboardInterrupt:
```

```
        print('>>>quit')
    #sys.exit(0)
```