12-2023

Ensuring Non-Repudiation in Long-Distance Constrained Devices

Ethan Blum

ENSURING NON-REPUDIATION IN LONG-DISTANCE CONSTRAINED DEVICES

By

Ethan Blum

A thesis submitted in partial fulfillment of the requirements of the Honors College at University of South Alabama and the Bachelor of Science in the Computer Science Department

University of South Alabama

Mobile

December 2023

Approved by:

_____	_____
Mentor: Dr. Michael Black	Committee Member: Mr. Ricky Green
_____	_____
Committee Member Dr. Todd McDonald	Douglas Marshall, Dean, Honors College

ACKOWLEDGMENTS

I would like to thank my Mentor, Dr. Michael Black, for all his hard work and dedication in helping me get to where I am. I cannot thank Dr. Black enough for how much he has helped me as my mentor and for extracurricular activities like the University of South Alabama DayZero Cybersecurity Club training. I would like to thank Mr. Ricky Green for allowing me to perform my testing with his devices and for his support as my committee member. I would also like to thank Dr. Todd McDonald for everything he has done for me. Dr. McDonald is one of the primary reasons for my appreciation for cybersecurity. He has recommended me for countless positions at prominent companies and opened many doors for me by being the sponsor of the DayZero Cybersecurity Club. I would also like to thank my friends at the DayZero Cybersecurity Club for attending competitions nationwide and helping me through my college career. I would also like to thank any other faculty and staff at the University of South Alabama School of Computing for the fantastic friends I have made while working there.

TABLE OF CONTENTS

Page

LIST OF TABLES v

LIST OF FIGURES vi

LIST OF ABBREVIATIONS vii

ABSTRACT viii

CHAPTER I INTRODUCTION 1

CHAPTER II RELATED WORK 5

CHAPTER III METHODOLOGY 10

3.1: Creation of The Message 11
3.2: Checking The Message on The Satellite 13
3.3: Testing Background 15

CHAPTER IV RESULTS 19

CHAPTER V CONCLUSION AND FUTURE RESEARCH 22

REFERENCES 24

APPENDIX 27

Appendix A: Bash Code 27

LIST OF TABLES

Table Page

TABLE 1: RESULTS OF TESTING: AGS VS UGS 20

TABLE 2: RESULTS OF TESTING: WITHIN TIME WINDOW AND OUTSIDE TIME WINDOW.... 21

LIST OF FIGURES

Figure Page

FIGURE 1: GROUND STATION AS A SERVICE PROBLEM 2

FIGURE 2: CORRECT MESSAGE STRUCTURE.....12

FIGURE 3: MALICIOUS MESSAGE STRUCTURE.....15

FIGURE 4: ACCEPTED MESSAGE.....16

FIGURE 5: ERROR MESSAGE.....16

LIST OF ABBREVIATIONS

BER	Bit Error Rate
BPSK	Binary Phase Shift Keying
DHCPv4	Dynamic Host Configuration Protocol for IPv4
DoS	Denial of Service
ECIES	Elliptic Curve Integrated Encryption Scheme
GS	Ground Station
GSaaS	Ground Station as a Service
ISS	International Space Station
LEO	Low Earth Orbit
MAC	Media Access Control
MACs	Message Authentication Codes
MD5	Message Digest 5
OSNMA	Open Service Navigation Message Authentication
QPSK	Quadrature Phase Shift Keying
SHA	Secure Hash Algorithm
TCP	Transport Control Protocol
UDP	User Datagram Protocol

ABSTRACT

Satellite communication is essential for the exploration and study of space. Satellites allow communications with many devices and systems residing in space and on the surface of celestial bodies from ground stations on Earth. However, with the rise of Ground Station as a Service (GsaaS), the ability to efficiently send action commands to distant satellites must ensure non-repudiation such that an attacker is unable to send malicious commands to distant satellites. Distant satellites are also constrained devices and rely on limited power, meaning security on these devices is minimal. Therefore, this study attempted to propose a novel algorithm to allow authenticating communications from ground stations to long-distance satellite communications while also ensuring the time delay on constrained devices remains low.

CHAPTER I

INTRODUCTION

Satellite communication is essential for the exploration and study of distant worlds. With countries and organizations across the globe set on venturing across space, there must be an effective way to ensure communication is protected. Distant worlds require long-distance satellite signals to enable communication from Earth across vast distances. Long-distance is defined as distances greater than lunar orbit. These distances range from 56 million km to 378 million km for the purposes of this study. Satellite communications introduce various problems such as high bit-error rates (BER) and increased time delays that slow down or prevent various security techniques used on the ground by devices with less BER and time delay (Shah et al., 2014). Because of these factors, securing satellite communications is a difficult task that must not increase the time delay.

In recent years there has been an interest in Ground Station as a Service (GSaaS) capabilities (Poole et al., 2021). These stations can be rented by organizations or entities for whatever they may need. They, however, introduce security challenges to satellite protection as anyone could attempt to contact a satellite (Poole et al. 2021). This security risk can be seen in Figure 1. Whether the adversary can communicate or send commands

to a satellite requires onboard cybersecurity assurances to prevent unauthorized access. In Figure 1, the instance of GSaaS is represented by the Unauthenticated Ground Station (UGS) and is for the adversarial testing and validation of the novel method proposed.

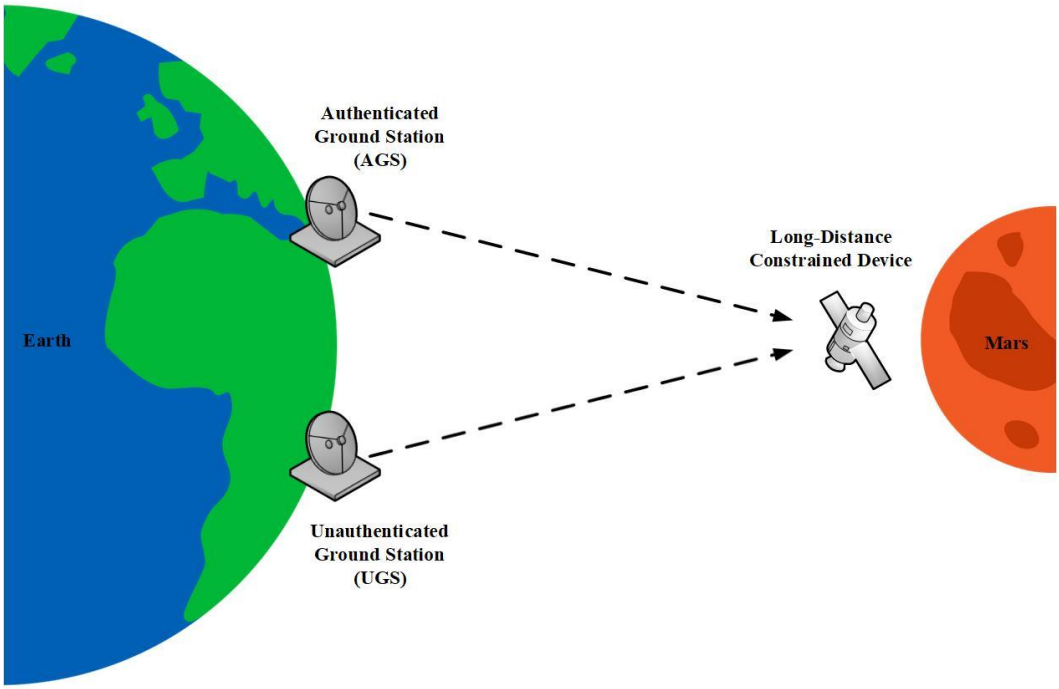

Figure 1: Ground Station as a Service Problem

One way to secure messages is the use of encryption. Encryption is effective at ensuring messages are sent in a secure way, but this also runs into the issue of onboard power consumption (Maple et al., 2022). Encryption is also limited because of the vast distance the communication travels, leading to time delays (Shah et al., 2014). These time delays make various encryption methods infeasible for extensive use across long-distances (Shah et al., 2014).

Problems such as high bit-error rates and increased time delays are compounded by the constrained power draw of satellite communications. Cybersecurity in space equipment has often been constrained by the limited resource nature of satellites and other space equipment (Maple et al., 2022). For this reason, protection methods in space must ensure little resources are used for communication.

With the advent of increased space traffic and mega-constellations in Low Earth Orbit (LEO) and beyond, cybersecurity in space has become a major issue. Along with that, there are an ever-increasing number of for-rent ground stations that an adversary could use for malicious purposes (Poole et al., 2021). Because an attacker could use these ground stations to send false requests to orbiting or long-distance satellites, there must be an authentication method to ensure non-repudiation in these space devices.

Access to long distance space devices should be limited to only authenticated devices. An attacker could use a weak authentication method to gain access and send malicious instructions to a space device. Some encryption algorithms can prevent this from happening, but they also require higher power in a resource constrained device. Encryption algorithms might ensure integrity and privacy, but some fail to solve the non-repudiation and access control aspects of satellite communication security. This limits the usability of such algorithms to ensure integrity and non-repudiation. To limit computing power consumption, a modified whitelist was introduced. This method ensured that the data being sent to the satellite was, first, correct. It then employed a ground station specific key to verify the sender. This computation was limited because the ground station was doing much of the computing involved in the creation of the message. This limited the on-board satellite computation because the message was sent in a format that

was easily read and understood. The satellite only needed to receive the data and verify the message that was sent by the authenticated ground station.

The research question for this study, “Can an algorithm be developed that ensures non-repudiation in long distance communications of power constrained devices?” The approach involved using three hashes to authenticate and validate the message sent to the long-distance constrained device. In this question, a successful condition was when the satellite could check the message and ensure it was coming from an authenticated ground station within the time delay plus two minutes. The failure condition was if the message that is received was not accepted by satellite or was received outside of the time delay window. This method was tested to ensure it can reliably prevent an unauthorized party from sending commands to a long distance, constrained device. It was also tested to ensure that outdated messages are not able to be used to harm satellite functions and processes.

The paper will be organized in the following manner. Chapter II includes the related works and will set up the foundation for the program background. Chapter III will go over the setup of the program and how tests will be conducted. Chapter IV will discuss the results of the testing and how they relate to the research question. Chapter V will conclude the paper and discuss any future topic that needs to be explored in this research.

CHAPTER II

RELATED WORK

Various research has been proposed regarding security in satellite communications. These studies form the foundation for the tools that are employed in this research. These studies will be compared to other studies and the applications of these research papers to this study will also be discussed.

UDP, or User Datagram Protocol, is a connection-less protocol (Postel, 1980). This means it does not need to establish a connection with an external device before it sends information. UDP sends data in groups called datagrams. A datagram contains the source port, destination port, length, and checksum sections in the header, along with the payload (Postel, 1980). UDP is the protocol to use when time is the constraint of communication, such as communication satellites (Criscuolo et al., 2001). As UDP is connection-less, it does not need to establish a connection before transmitting information. Establishing a connection takes valuable time that would otherwise be used to begin sending data. UDP is used because of the time-delay that long-distance satellite communications must deal with.

Time delay is a major issue when communicating long distances such as the distance between Earth and Mars, 378 million km, at the furthest point (Attwood, 2018).

At this distance the time delay is around 21 minutes one-way (Koktas & Basar, 2022; Attwood, 2018). This means that each message sent to and from a satellite orbiting mars would take 21 minutes to be received and another 21 minutes to be acknowledged totaling 42 minutes (Attwood, 2018). As the Earth and Mars get closer, the distance lessens as well as the time delay. At the closest point, the distance from Earth to Mars is 56 million kilometers and communications take around 3.2 minutes (Attwood, 2018). Another factor for time delay is the processing requirements for more stringent securing methods such as various encryption algorithms. Maple et al., (2022), discusses the time delay used for encrypting packets sent to and from the International Space Station (ISS). During these tests the Elliptic Curve Integrated Encryption Scheme (ECIES) encryption took at most 35 ms for 500 bytes of data and decryption took 25 ms for 500 bytes of data (Maple et al., 2022). These numbers in themselves are very good compared to the 21 minutes the message would take to be sent to mars, but this leaves out the power requirement of encrypting and decrypting a large volume of messages across a distance such as Mars to Earth. While on the surface, encryption is a good idea because it restricts the ability of an adversary to read messages sent in plain-text. However, encryption takes up valuable computing resources such as energy that is scarce in constrained devices such as satellites (Maple et al., 2022).

Hashing algorithms have become popular for uses in data validation and host authentication (Pittalia, 2019). These algorithms take a variable sized input and convert it into a fixed sized output (Pittalia, 2019; Zniti & Ouazzani, 2023). The given output is always the same for the same input (Pittalia, 2019). Because of this, data validation can be performed using hashing algorithms (Zniti & Ouazzani, 2023). If data being used has

been altered or changed, the hash of that data will be different from the original. Another aspect of certain hashing algorithms is the one-way hash. A one-way hash is the inability to determine the input based on a given output (Pittalia, 2019). Some of the most common hashing algorithms today are Message Digest 5 (MD5) and Secure Hash Algorithm (SHA). MD5 is good for data validation and error checking while SHA is better for authentication uses (Pittalia, 2019). MD5 struggles with collision, which is a serious issue when using it for authentication instead of data validation (Pittalia, 2019). MD5, however, is faster to compute than SHA (Pittalia, 2019). For these reasons, SHA is used for Authentication, but data validation where MD5 is used because it requires less resources to compute.

Hashing for use in authentication of satellite communications can be seen in the Galileo Open Service system (Götzelmann et al., 2023). This system relies on the Open Service Navigation Message Authentication (OSNMA) protocol to protect users from malicious data and ensure users don't transmit malicious data (Götzelmann et al., 2023). This system provides Message Authentication Codes (MACs) generated using Keyed-Hash Message Authentication Codes (KMAC) (Götzelmann et al., 2023). KMACs use the data or payload being sent to the Galileo device on Earth and a cryptographic key to generate a MAC for the message (Götzelmann et al., 2023). These generated MACs can be used to find the next MAC key as the following key is generated from the existing MAC key (Götzelmann et al., 2023). This allows the user to verify previous or future keys from data that has already been received (Götzelmann et al., 2023). This method requires the use of a public key to be retrieved from the Galileo Service Centre during the test and preparation phase (Götzelmann et al., 2023). Various other studies also looked at

using public-private key pairs to ensure authentication (Khan et al., 2022; Liu et al., 2022). This might restrict usability in long-distance satellite communications as the time delay will be too great for a public key to be feasible (Shah et al., 2014). The reason for this has to do with the time delay associated with using such a key. If the satellite must query the ground station for its public key, the communication time of a message will increase as there is another message that needs to be sent to the ground station. On the other hand, if the ground station is attempting to send a message, it must either have the satellite's public key stored locally or must send a long message to retrieve it from the satellite. Another issue with using encryption during these communications is the power consumption for satellites is limited, making encryption costly (Maple et al., 2022).

Bit Error Rate (BER) is the communication system performance that depends on errors resulting from electronic-circuit related noise and turbulence (Mahdieh & Pournoury, 2010). There are three ways by which errors can result in communication such as scintillation, beam spreading, and beam wander (Mahdieh & Pournoury, 2010). Scintillation is due to small inconsistencies in the atmosphere as a beam passes through (Mahdieh & Pournoury, 2010). Beam spreading is the tendency for a beam to spread as it travels (Mahdieh & Pournoury, 2010). Beam wander is caused by much larger inconsistencies in the atmosphere as the beam is passing through (Mahdieh & Pournoury, 2010). Bohora and Bora, (2014), attempt to analyze the BER for different modulation schemes such as Binary Phase Shift Keying (BPSK) and Quadrature Phase Shift Keying (QPSK). Modulation techniques are used to improve BER performance (Bohora and Bora, 2014). Based on the results provided by Bohora and Bora, 2014, Binary Phase Shift

Keying is the more effective method to reduce BER in wireless communications (Bohora and Bora, 2014).

Shah et al., (2014), attempts to provide security and vulnerability information for commercial and military satellite communications. To ensure communication security, five features must be considered, confidentiality, authentication, integrity, access control, and key management (Shah et al., 2014). Confidentiality means that authenticated users are the only ones with access to the information (Shah et al., 2014). Authentication requires the user to verify his or her identity and can be done a variety of ways, each with their own pros and cons (Shah et al., 2014). Integrity ensures that the information being sent remains intact and hasn't been tampered with (Shah et al., 2014). Access Control ensures a system cannot be accessed without authorization via a compromise (Shah et al., 2014). Key management is the management of security keys used for things such as authentication (Shah et al., 2014). This research will handle four of these features, authentication, integrity, access control, and key management.

CHAPTER III

METHODOLOGY

To address the research question, “Can an algorithm be developed that ensures non-repudiation in long distance communications of power constrained devices?” The discussion of the environment, creation of the authentication method, validation of authentication, and testing setup is performed in this section. If the setup of the research is done in this fashion, it will produce a method of sending commands to a long-distance constrained device for applications in the space industry. The resulting testing proves that this method is valid and can prevent an attacker from sending malicious commands or replaying a previous message.

The testbed was set up with two-nodes. The two-node setup assumes the planet is facing the direction of the target. The reason for this is because, if the planet was facing away from the direction the radio dish is pointing, then there would need to be a relay to allow for constant communication. This is out of the scope of this research as this is proof of concept. A ground station was implied to be capable of transmitting high-energy focused radio signals. For this research it was assumed there was a satellite in Mars-orbit that captures batch command transmissions and sends individual commands from the

batch to a subservient device or internal process. The language chosen to produce and verify the message was Bash. This language was employed because of the versatility it provides. It comes with most distributions of Linux and can be easily implemented.

3.1 Creation of The Message

The modified whitelist system was similar to the Dynamic Host Configuration Protocol for IPv4 (DHCPv4), which uses the MAC address to authenticate a whitelist (Xie et al., 2021). This protocol was used to model the design of the assigning of the unique client identity. This ensured a lightweight protocol was implemented for whitelist checking to ensure limited round trip time (RTT) delay. Each packet sent to the satellite had the MAC address of the sender attached to it to ensure that the packet will be accepted by the satellite. This project deviated from the DHCPv4 method as it instituted multiple one-way hash functions to authenticate the Authenticated Ground Station. Instead of using only the MAC address, it also instituted information about the message being sent into the hash. Included is a MD5 hash of the payload or the commands sent from the ground station, the MAC address of the ground station, and the time the message was sent in Base64.

The message being sent to the satellite was set up in a way that allows for easy reading by the destination satellite as seen in *Figure 2*. The first line of the message contained an MD5 hash of the payload of instructions sent to the satellite. This hash was created by computing the MD5 hash of lines 3 – 14, or the payload. An MD5 hash was chosen because it is good for integrity checking of files and strings. The second line contained a SHA256 hash of the payload hash from line 1 and the MAC address of the

authenticated ground station, and the Base64 timestamp. A Base64 encoded timestamp was added to the end of the SHA256 hash to validate time and to protect against certain replay attacks. Base64 was chosen because it is not a one-way hash and can be undone for comparison upon arrival to the satellite. Lines 3 – 14 contained instructions or the payload of the message. These instructions were general terms for any predefined commands to be run on the satellite. They were the instructions set to be executed by the

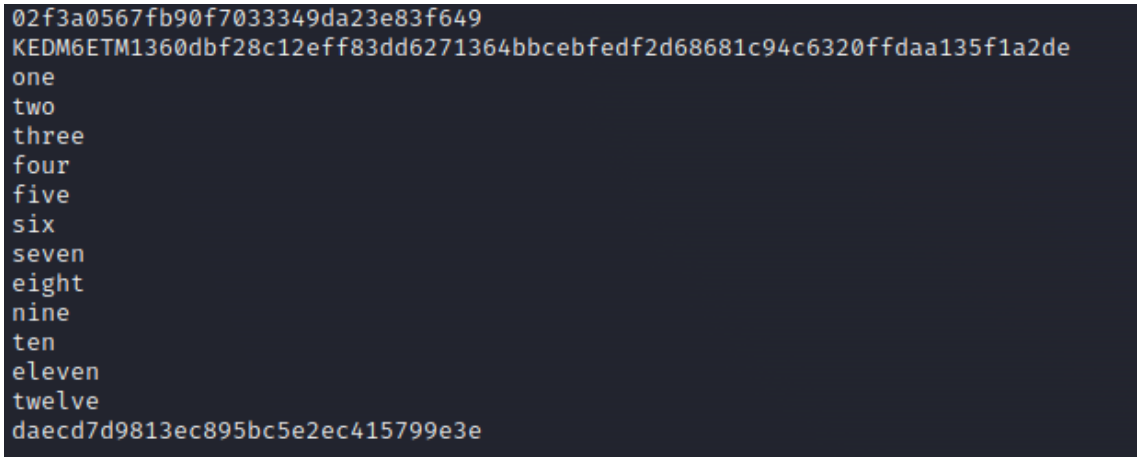

Figure 2: Correct Message Structure

satellite upon the message's arrival. If any of the instructions aren't used, they were considered a NULL command, and nothing was done. These NULL commands were still needed to fill the line, as the message format is constant and needs to be maintained. To maintain the format of the message the word "NULL" was placed in any un-used command sent to the satellite. The message finished with line 15 where there was another MD5 hash of the entire message with the other two hashes included. Lines 1-14 were

hashed with MD5. The contents of these lines were the MD5 of the payload in line 1, the SHA256 for authentication on line 2, the Base64 of the timestamp from line 2, and the payload of commands from lines 3-14. All of these were hashed using MD5 and placed on line 15. This hash ensured file integrity and that it hadn't been altered during transit.

Obfuscation of the hashes was used to ensure the hashes couldn't be reproduced or brute forced. This was done through an obfuscation mechanism used during the creation of the hashes. A string was first split in half. The second half was put in the front, and the first half was put in the back. The resulting string was then reversed per character. This was a simple way to increase the complexity of brute-forcing these hashes. Hashes on line 1 and line 15 used this method without any modifications. The SHA256 hash on line 2 used this method, but before the hash was reversed per character, the base64 encoded timestamp was added to the back of the hash. It was then reversed per character as in the other two hashes. The message that was created can then be sent to the satellite.

3.2: Checking The Message on The Satellite

Once the message arrived, the first action was reversing the SHA256 hash on line 2. The Base64 timestamp was pulled out for future comparison with the arrival time. Because the Base64 in Line 2 hadn't been through the full obfuscation routine, a reversal was all that is needed to pull it out. The SHA256 hash was then de-obfuscated and saved for future comparison. The de-obfuscation process involved reversing the hash per character, splitting the hash in half, and putting the first half in the back and the second half in the front. Thus, reversing the previous obfuscation technique performed during the

creation of the message. Line 1, or the MD5 hash of the payload, was then pulled out and went through the de-obfuscation process as well. It was then saved for future comparison. Line 15 then went through the de-obfuscation process and was saved for future comparison.

The satellite then generated these hashes using the information provided. Hash 1, or the MD5 of the payload of commands, was the computed MD5 hash of the payload inside the message that was received. Hash 2, or the SHA256 hash, computed a SHA256 hash using Hash 1 and the MAC address of the authenticated ground station (AGS) stored locally on the satellite. Hash 3, or the MD5 of the entire message, computed the MD5 hash of Hash 1, Hash 2, the Base64 timestamp from the message, and the payload from the message. A current timestamp was then taken and stored for future use during comparison.

The satellite then decoded the Base64 timestamp from the ground station and computed the difference of the current timestamp. If the difference was greater than two minutes from the expected time delay from the ground station, the satellite errors and sent a message to the AGS saying the message failed. The time inside the file couldn't be altered because this changed the integrity hashes within the file as well. The first hash to be checked was the MD5 hash located on line 15, or the MD5 of the whole message. This made sure the message being sent had not been altered. If the hash matched Hash 3 computed by the satellite, the program continued to the next hash, otherwise, another failed message was sent to the AGS, and the process ended. The next hash checked was the SHA256 hash, or the authentication hash on line 2. This ensured the message was coming from the correct place and wasn't reproduced or replayed. This hash was checked

against Hash 2 computed by the satellite. If they were not the same, another failed message was sent, and the process stopped. The last hash to be checked was the MD5 hash on line 1. This was more of a confirmation to validate the integrity of specifically the payload. This hash was checked against Hash 1 that was computed by the satellite. If the message was faulty, lost integrity, or the authentication hash didn't match, the program transmitted a small message back to the AGS saying the message had an error. If all the tests passed, a small message was sent stating that the message was received successfully. These messages sent back to the ground station were not a form of handshake, they were more to ensure the process is working. The reason this was used instead of a handshake-based protocol was because the use of long-distance communications is expensive, especially for resource constrained devices. Using a handshake-based protocol requires extra communications.

3.3: Testing Background

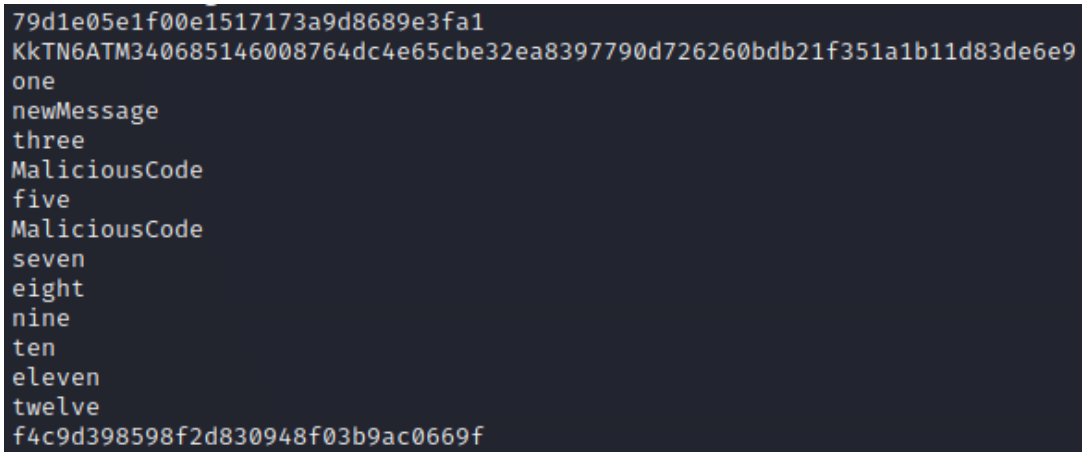

Figure 3: Malicious Message Structure

The security of this implementation was tested by an unauthorized third party. This third party sent a message to the satellite that was slightly altered, and the message was tested to see if the third party was able to gain access to the satellite. The unauthorized individual sent a batch of commands to the satellite and attempted to run false commands. These commands were different from the ones sent by the AGS and were used to test the integrity checks put in place in the message. If the device sent an error message back to the AGS, this test was considered a success. If an accepted message was sent from the satellite to the AGS, this was seen as a fail.

To test the timeout function used on the satellite, a second evaluation of the message was done. This test attempted to simulate sending the exact same message sent by both the AGS and the unauthenticated ground station (UGS). The UGS sent the message after the timeout setting of 2 minutes was passed. The time the message was sent was documented in a Base64 timestamp included in Line 2. This was tested by sending the same message twice. If the second message was not executed and an error was transmitted back to the ground station instead, this was considered a success.

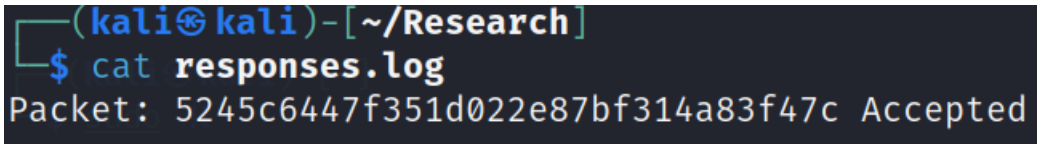

Figure 4: Accepted Message

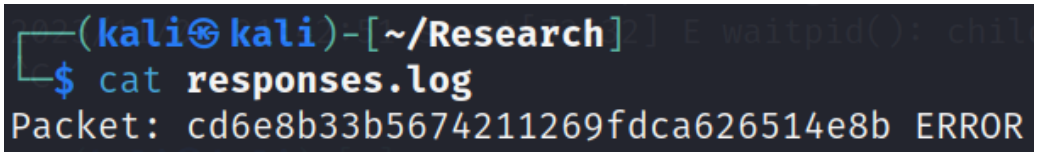

Figure 5: Error Message

One of the issues with using a hash function is the ability to replay the communication and attempt to gain unauthorized access to the device. This issue was negated by using the authenticated hash for every transmission. A malicious party would also have to be positioned directly above the ground station to have the ability to capture and retransmit packets. This is because the use of high-energy focused radio signals necessitates the adversary being directly above the AGS. To further prevent a party from submitting outdated credentials, the instructions sent and the MAC address of the AGS was used to create a SHA256 hash that changed with each message sent.

Testing was performed using a laptop to simulate the AGS and a Raspberry Pi Zero to simulate the constrained device or satellite. These devices were connected via ethernet using a network switch. Because of this, there was a limited time delay between devices. This time delay was simulated via a wait call that pauses the sending of the message until the desired time delay has been reached. Three separate time delays were tested 5 minutes, 14 minutes, and 21 minutes for three distances in Mars' orbit.

The laptop acted as the AGS compiled and sent the message to the long-distance constrained device or satellite. The AGS had an open port listening for sending messages from the constrained device. The constrained device had the MAC address of the AGS loaded onto it. The constrained device was listening for incoming messages on an open port. These messages were sent for checking before any of the commands are run. A third device was connected to the ethernet network to simulate the UGS attempting to recreate the message being sent to the constrained device.

To listen and send messages, Socat (Socket Cat) was used because it doesn't send extra messages like net cat. Net cat sent a TCP FIN message when it closed a session, and

this was extra overhead that is not viable in the long-distance satellite environment that was emulated. This didn't come natively installed on Linux, so it needed to be installed before deployment. The Earth is assumed to be in constant communication with the long-distance constrained device and it does not block communication by spinning away from the constrained device. The MAC address was pre-loaded onto the long-distance constrained device.

CHAPTER IV

RESULTS

The test conducted was the 5-minute test, which had the Authenticated Ground Station (AGS) sending a message to the constrained device with the wait setting at 5 minutes. The Unauthenticated Ground Station (UGS) then attempted to recreate the message by sending a message with identical authenticated and integrity hashes and a different payload. The first success condition was if the constrained device accepts the commands from the AGS and ran them then transmitted a received message back to the AGS. The second success was if the constrained device sends an error or a failed message to the AGS if the UGS sent a message to the constrained device with an altered payload. This proved the UGS message failed and was labeled as Failed. This test was done for the 14-minute and the 21-minute time delays as well. This test was not testing the timestamp mechanism because this mechanism was to prevent another type of replay attack that was tested in the next test.

As seen in Table 1, the AGS could send batch-command messages to the constrained device. The time delay settings successfully authenticated or denied the given ground station. These messages used the format 5 minutes to the constrained device and 5 minutes for accepted or failed requests back to the authenticated ground station. The UGS

attempted to send different commands to the constrained device. When these messages made it to the constrained device, they were calculated to be unauthenticated, and a failed message was sent to the AGS.

Table 1: Results of Testing: AGS vs UGS

Time Delay Setting	AGS Result	AGS Time	UGS Result	UGS Time
5 Minutes	Success	10:01.25 Min	Failed	10:00.58 Min
14 Minutes	Success	28:00.76 Min	Failed	28:02.24 Min
21 Minutes	Success	42:00.52 Min	Failed	42:00.65 Min

The second test attempted to resend the message a second time after a message had previously passed the accepted time-delay window. Each window allowed for 2 minutes more than the current time delay of a message being sent. For a 5-minute time delay, there was a 7-minute window from the time sent for the message to be received. If it was received after that, it was not accepted, and an error message was sent back to the ground station. In this test, the payload/instructions or the hashes was not be modified, the message was only retransmitted to attempt to overload the constrained device with instructions. This was tested for each of the three time-delay settings, 5 minutes, 14 minutes, and 21 minutes. The time window for these were 7 minutes, 16 minutes, and 23 minutes, respectively.

Table 2: Results of Testing: Within Time Window and Outside Time Window

Time Delay Setting	AGS Result	AGS Time	UGS Result	UGS Time
5 Minutes	Success	10:02.52 Min	Failed	18:17.58 Min
14 Minutes	Success	28:01.21 Min	Failed	35:23.15 Min
21 Minutes	Success	42:00.93 Min	Failed	49:42.12 Min

According to these tests, a freshly generated message could be sent to the constrained device. However, the same message sent outside the time window was not processed by the constrained device. This shows that the time window method did ensure that an adversarial replay attack was prevented. Each test was done with another message and each replay attack was performed with the message generated by the AGS.

CHAPTER V

CONCLUSION AND FUTURE RESEARCH

This research was successfully able to simulate authentication in a satellite environment. The authenticated ground station could send commands to the constrained device or satellite. Once these commands made it to the constrained device, it authenticated the sender and checked the validity of the message. If the sender was correct, an accepted message was sent back to the authenticated ground station. The unauthenticated ground station also attempted to send a seemingly authenticated message. This message was checked in the same manner and proven to be false. The constrained device then sent a failed message back to the authenticated ground station for logging purposes. These tests simulated the creation of a novel method of ensuring non-repudiation for long-distance communications on constrained devices without dramatically increasing time delay or increasing the resource requirements of the communication.

This research could be taken in many different directions. In this instance, there was no handling of error messages sent back to the authenticated ground station. This included retransmit handling or unauthenticated device access handling. Because no encryption was used in this research, employing a light-weight encryption algorithm that

doesn’t need a handshake or session might prove useful. This method also needs to be compared against other standard methods of ensuring non-repudiation for long-distance devices. Attacks such as more complex replay attacks, denial-of-service attacks, intentional format issues, command injections, etc. could be tested against this method in future research. The programming language used in this research was Bash. This language, while versatile and easily implementable, does increase the overhead of the algorithm. Future research should be done to reproduce this algorithm using a less computationally expensive language such as C or C++.

REFERENCES

Attwood, J. R. (2018, June). *JRASC2018JuneMarsAttwood.pdf*. Royal Astronomical Society of Canada. https://www.rasc.ca/sites/default/files/JRASC2018JuneMarsAttwood.pdf

Baqtian, H., & Ali Al-Aidroos, N. (2023). *Three Hash Functions Comparison on Digital Holy Quran Integrity Verification*. *11*, 1–7.

Bohra, D. D., & Bora, A. (2014). *Bit Error Rate Analysis in Simulation of Digital Communication Systems with Different Modulation Schemes*. *1*(3).

Bonafini, S., Satriano, N., & Sacchi, C. (2022). Study on Relay Networks based on Lagrangian Points for Optical-based Mars-to-Earth Communications. *2022 IEEE 9th International Workshop on Metrology for AeroSpace (MetroAeroSpace)*, 152–157. https://doi.org/10.1109/MetroAeroSpace54187.2022.9856020

Eggert, L., Fairhurst, G., & Shepherd, G. (2017). *UDP Usage Guidelines* (Request for Comments RFC 8085). Internet Engineering Task Force. https://doi.org/10.17487/RFC8085

El-hajj, M., Mousawi, H., & Fadlallah, A. (2023). Analysis of Lightweight Cryptographic Algorithms on IoT Hardware Platform. *Future Internet*, *15*(2), Article 2. https://doi.org/10.3390/fi15020054

Götzelmann, M., Köller, E., Viciano-Semper, I., Oskam, D., Elias Gkougkas, & Simon, J. (2023). Galileo Open Service Navigation Message Authentication: Preparation

Phase and Drivers for Future Service Provision. *NAVIGATION: Journal of the Institute of Navigation*, *70*(3). https://doi.org/10.33012/navi.572

Khan, M. A., Alzahrani, B. A., Barnawi, A., Al-Barakati, A., Irshad, A., & Chaudhry, S. A. (2022). A resource friendly authentication scheme for space–air–ground–sea integrated Maritime Communication Network. *Ocean Engineering*, *250*, 110894. https://doi.org/10.1016/j.oceaneng.2022.110894

Koktas, E., & Basar, E. (2022). *Communications for the Planet Mars: Past, Present, and Future* (arXiv:2211.14245). arXiv. http://arxiv.org/abs/2211.14245

Liu, T., Peng, W., Zhu, K., & Zhao, B. (2022). A Secure Certificateless Signature Scheme for Space-Based Internet of Things. *Security and Communication Networks*, *2022*, 1–13. https://doi.org/10.1155/2022/5818879

Mahdieh, M. H., & Pournoury, M. (2010). Atmospheric turbulence and numerical evaluation of bit error rate (BER) in free-space communication. *Optics & Laser Technology*, *42*(1), 55–60. https://doi.org/10.1016/j.optlastec.2009.04.017

Maple, C., Epiphaniou, G., Hathal, W., Atmaca, U. I., Sheik, A. T., Cruickshank, H., & Falco, G. (2022). The Impact of Message Encryption on Teleoperation for Space Applications. *2022 IEEE Aerospace Conference (AERO)*, 1–10. https://doi.org/10.1109/AERO53065.2022.9843424

Neish, A., Walter, T., & Enge, P. (2019). Quantum-resistant authentication algorithms for satellite-based augmentation systems. *NAVIGATION*, *66*(1), 199–209. https://doi.org/10.1002/navi.287

Pittalia, P. (2019). A Comparative Study of Hash Algorithms in Cryptography. *International Journal of Computer Science and Mobile Computing*.

Poole, C., Bettinger, R., & Reith, M. (2021). Shifting Satellite Control Paradigms: Operational Cybersecurity in the Age of Megaconstellations. *Air & Space Power Journal*, *35*(3), 46–56.

Postel. (1980). *Information on RFC 0768 » RFC Editor*. https://www.rfc-editor.org/info/rfc0768

Postel. (1981). *Transmission Control Protocol*. https://www.ietf.org/rfc/rfc793.txt

Shah, S. M. J., Nasir, A., & Ahmed, H. (2014). *A Survey Paper on Security Issues in Satellite Communication Network infrastructure*. *2*(6).

Tedeschi, P., Sciancalepore, S., & Di Pietro, R. (2022). Satellite-based communications security: A survey of threats, solutions, and research challenges. *Computer Networks*, *216*, 109246. https://doi.org/10.1016/j.comnet.2022.109246

Xiao, X., You, L., Wang, J., Wang, W., & Gao, X. (2023). Multigroup Multicast Beamforming for High Throughput GEO Satellite Communications Under Power-Consumption Outage Constraints. *IEEE Communications Letters*, *27*(3), 941–945. https://doi.org/10.1109/LCOMM.2023.3241086

Xie, W., Yu, J., & Deng, G. (2021). A Secure DHCPv6 System Based on MAC Address Whitelist Authentication and DHCP Fingerprint Recognition. *2021 7th Annual International Conference on Network and Information Systems for Computers (ICNISC)*, 604–608. https://doi.org/10.1109/ICNISC54316.2021.00114

Zniti, A., & Ouazzani, N. E. (2023). Hash algorithm comparison through a PIC32 microcontroller. *Bulletin of Electrical Engineering and Informatics*, *12*(4), Article 4. https://doi.org/10.11591/eei.v12i4.4982

APPENDIX

Appendix A: Bash Code

This appendix includes all the code used to perform this research. They are listed by the file the code belongs to.

CreateMessage.sh

```
#!/bin/bash
source hashFile.sh
source makeKey.sh
file="./testFile.txt"


transformMessage() {
    local string="$1"

    local length=${#string}
    local half_length=$(($length/2))

    local first_half=$(echo "${string:0:$half_length}")
    local second_half=$(echo "${string:$half_length:$length}")

    echo "$second_half$first_half"
}



# Obfuscate Message Hash
revMD5=$(rev <<< $(transformMessage $md5_hash))

echo "Obfuscated Payload Hash: $revMD5"

# Generate Authentication SHA256
shaHash=$(./makeKey.sh)
currTime=$(date +%H:%M)
```

```
timeHash=$(base64 <<< $currTime)
echo "$timeHash"
# Add Time Hash to the back of SHA256 and reverse everything
result=$(rev <<< $(transformMessage $shaHash)$timeHash)
echo "Sha Hash: $shaHash"
echo "Obfuscated Authentication Hash: $result"

# generage hash of message
messageHash=$(md5sum <<< $md5_hash$result$payload | awk '{print $1}')

# obfuscate Message Hash
revMessHash=$(rev <<< $(transformMessage $messageHash))

echo "Obfuscated Message Hash: $revMessHash"

# Input all the data into the file
echo "$revMD5" >> temp.txt

echo "$result" >> temp.txt

cat $file >> temp.txt

echo "$revMessHash" >> temp.txt

# Pass data to message.txt
mv temp.txt message.txt
```

hashFile.sh

```
#!/bin/bash

# provide file path
file="./testFile.txt"

num_lines=$(wc -l < "$file")
Hash1="
Hash2=""
payload=""
Hash3=""
# Loop through the file using a for loop
```

```
for ((line_number = 1; line_number <= num_lines; line_number++)); do
   # Read and process the line at the current line number
   line=$(sed -n "${line_number}p" "$file")

   payload+="$line"

done < "$file"

# Calculate the generic file hash
md5_hash=$(md5sum <<< $payload | cut -d ' ' -f 1)

# Return MD5 hash
#echo "$md5_hash"
```

makeKey.sh

```
#!/bin/bash

# Add source file
source findNetVector.sh
source hashFile.sh

# Get the host machines MAC address
interface_name=$sanitize_string

# Get and parse the mac address
mac_address=$(ip link show $interface_name | awk '{print $2}' | grep -o -E "([0-
9A-Fa-f]{2}[:-]){5}([0-9A-Fa-f]{2})")


# Get hash of file
fileHash=$md5_hash

# Create ground key Add time too
textToHash=$fileHash$mac_address

groundKey=$(sha256sum <<< $textToHash| awk '{print $1}')

#flip the hash so it is harder to compute
```

```
echo $groundKey
```

findNetVector.sh

```
#!/bin/bash

# This bash script is used to detect and output the current network device
# being used. This will be used to send a sentinel packet.

# using IP
ip_output=$(ifconfig)

# use grep to filter out only active interfaces with IP addresses
active_interfaces=$(echo $ip_output | awk '/<UP,BROADCAST/{print $1}')

sanitize_string=$(echo $active_interfaces | tr -d ':')

#echo $sanitize_string
```

sendMessage.sh

```
#!/bin/bash


#sources
source ./createMessage.sh

file="./message.txt"

sleep 0
socat - UDP:<IP Addr>:<Port> < $file

#rm $file
```

check.sh

```
#!/bin/bash

# define file paths
directory="./Research/check"
file="message.txt"
```

```
script="./Research/readFile.sh"

# Check if the file exists in the directory
if [ -e "$directory/$file" ]; then
    echo "File Found! Calling another script..."



    bash "$script"

else

    echo "Waiting on message..."

fi
```

readFile.sh

```
#!/bin/bash

file="./check/message.txt"


transformMessage() {
    local string="$1"

    local length=${#string}
    local half_length=$(($length/2))

    local first_half=$(echo "${string:0:$half_length}")
    local second_half=$(echo "${string:$half_length:$length}")

    echo "$second_half$first_half"
}


extractTime() {
    local string="$1"

    local length=${#string}
```

```
        local startIndex=$((length - 8))

        local timeHash=$(echo "$string" | rev | cut -c 1-8 | rev)

        echo "$timeHash"
}

# Get the number of lines in the file
num_lines=$(wc -l < "$file")
Hash1="
Hash2=""
payload=""
Hash3=""
macGS="<Ground Station Mac Address>"
prevTime=""
# Loop through the file using a for loop
for ((line_number = 1; line_number <= num_lines; line_number++)); do
   # Read and process the line at the current line number
   line=$(sed -n "${line_number}p" "$file")

   if [ $line_number -eq 1 ]; then
      Hash1=$(rev <<< $(transformMessage $line))
      echo "Hash 1: $Hash1"
   fi

   if [ $line_number -eq 2 ]; then
      prevTime=$(echo "$line" | cut -c 1-8 | rev)
      tempHash2=$(echo "$line" | cut -c 9-)
      Hash2=$(rev <<< $(transformMessage $tempHash2))
      echo "Hash 2: $Hash2"
   fi

   if [ $line_number -eq 15 ]; then
      Hash3=$(rev <<< $(transformMessage $line))
      echo "Hash 3: $Hash3"
   fi

   if [ $line_number -lt 15 ] && [ $line_number -gt 2 ]; then
      payload+="$line"
```

```
    fi

done < "$file"

currTime=$(date +%s)

echo "Message Sent Time: $prevTime"

Hash1Check=$(md5sum <<< $payload | cut -d ' ' -f 1)
echo "Hash 1 Check: $Hash1Check"
Hash2Check=$(sha256sum <<< $Hash1Check$macGS | cut -d ' ' -f 1)
echo "Hash 2 Check: $Hash2Check"
revHash2C=$(rev <<< $(transformMessage $Hash2Check)$prevTime)
Hash3Check=$(md5sum <<< $Hash1Check$revHash2C$payload | cut -d ' ' -f 1)
echo "Hash 3 Check: $Hash3Check"

decodepTime=$(date -d "$(echo "$prevTime" | base64 --decode)" "+%s")

timeDelay=0

decodepTime=$(echo $decodepTime | sed 's/^0*//')
currTime=$(echo $currTime | sed 's/^0*//')

echo "Time Sent: $decodepTime"
echo "Time Recieved: $currTime"

timeDiff=$((currTime - decodepTime))

echo "Time Difference: $timeDiff"

if [ "$timeDiff" -lt "$(($timeDelay + 120))" ]; then
        echo "Message is not expired"

        if [ $Hash3Check == $Hash3 ]; then
                echo "Hash 3 Matches"

                if [ $Hash2Check == $Hash2 ]; then
                        echo "Hash 2 Matches"
```

```
                if [ $Hash1Check == $Hash1 ]; then
                    echo "Hash 1 Matches"
                    echo "All Hashes Match! Valid message recieved"

                    sleep $timeDelay
                    echo "Packet: $Hash1Check Accepted" | socat –
                       UDP:<IP>:<Port>
                else
                sleep $timeDelay
                echo "Packet: $Hash1Check ERROR: Payload is
                corrupted/altered" | socat - UDP:<IP>:<Port>

                fi
            else
            sleep $timeDelay
            echo "Packet: $Hash1Check ERROR: Unauthorized" | socat –
             UDP:<IP>:<Port>

            fi

        else
        sleep $timeDelay
        echo "Packet: $Hash1Check ERROR: File is corrupted/altered" | socat –
           UDP:<IP>:<Port>

        fi

else
sleep $timeDelay
echo "Packet: $Hash1Check ERROR: Time Delay Exceeded" | socat -
UDP:<IP>:<Port>

fi
```