Covert Data Transmission using Secret-sharing and Network Steganography

by

Mohammad Ariful Islam
23241083

Shahrin Shafiq Simran
20101358

Mahir Aseef
20101338

Labiba Rahman
20141014

A thesis submitted to the Department of Computer Science and Engineering
in partial fulfillment of the requirements for the degree of
B.Sc. in Computer Science

Department of Computer Science and Engineering
School of Data and Sciences
Brac University
January 2024

Declaration

It is hereby declared that

1. The thesis submitted is our own original work while completing a degree at Brac University.
2. The thesis does not contain material previously published or written by a third party, except where this is appropriately cited through full and accurate referencing.
3. The thesis does not contain material that has been accepted or submitted for any other degree or diploma at a university or other institution.
4. We have acknowledged all the main sources of help.

Student's Full Name & Signature:

Mohammad Ariful Islam	Shahrin Shafiq Simran
23241083	20101358
Mahir Aseef	Labiba Rahman
20101338	20141014

Approval

The thesis titled "Covert Data Transmission using Secret-sharing and Network Steganography" submitted by

1. Mohammad Ariful Islam (23241083)
2. Shahrin Shafiq Simran (20101358)
3. Mahir Aseef (20101338)
4. Labiba Rahman (20141014)

Of Fall, 2023 has been accepted as satisfactory in partial fulfillment of the requirement for the degree of B.Sc. in Computer Science on January 18, 2024.

Examining Committee:

Supervisor:
(Member)

Arif Shakil

Lecturer
Department of Computer Science and Engineering
Brac University

Co Supervisor:
(Member)

Dr. Muhammad Iqbal Hossain

Associate Professor
Department of Computer Science and Engineering
Brac University

Program Coordinator:
(Member)

Dr. Md. Golam Rabiul Alam

Professor
Department of Computer Science and Engineering
Brac University

Head of Department:
(Chair)

Dr. Sadia Hamid Kazi

Chairperson
Department of Computer Science and Engineering
Brac University

Abstract

The present has made us more dependent on technology than ever before. As technology develops and new fields emerge, it has become more complicated to maintain a safe, secure, and covert environment for transmitting valuable information. Although cryptographic techniques are pretty strong in communication systems, they cannot be used as a standalone tool to communicate covertly. Covert transmission is vital for those who require extreme privacy and security, such as national defense organizations. Over the years, much research has been done based on covert data sent over a network. Our research has produced a new model to provide secure and hidden data transmission in a LAN. Our system model will be divided into secret sharing, network steganography, and hashing for integrity checks. The model will split the secret message into shares using Shamir's secret sharing scheme to add redundancy to our model so that even if some shares are lost during transmission, the message can still be reconstructed. Each share is hashed, and selected bits from the hash are appended to the corresponding share to provide integrity. Finally, we have used an ARP steganography technique and an IP steganography algorithm to send the shares, where each share is sent through one of the covert channels but not both. The steganographic algorithms provide a covert transmission channel and confidentiality for the transmission. A comprehensive security analysis of the overall model has been provided, highlighting how it provides security and covertness, potential vulnerabilities and weaknesses, and possible solutions.

Keywords: Covert Transmission, Secret-Sharing, Network Steganography, ARP Steganography, IP Steganography

Acknowledgement

Firstly, all praise to the Great Allah for whom our thesis has been completed without any major interruption.

Secondly, to our supervisor Arif Shakil sir and co-supervisor Dr. Muhammad Iqbal Hossain sir for their kind support and guidance in our work.

And finally to our parents without their support, it may not be possible. With their kind support and prayer, we are now on the verge of our graduation.

Table of Contents

List of Figures

List of Tables

Nomenclature

The next list describes several symbols & abbreviation that will be later used within the body of the document

ACK Acknowledgement Flag

ARP Address Resolution Protocol

DNS Domain Name System

FIN Finish Flag

IP Internet Protocol

LAN Local Area Network

MAC Media Access Control

$MTNS$ Moving Target Network Steganography

PSH Push Flag

$SCTP$ Stream Control Transmission Protocol

SHA Secure Hash Algorithm

TCP Transmission Control Protocol

UDP User Datagram Protocol

VMs Virtual Machines

Chapter 1

Introduction

1.1 Introduction

In today's digital age, the world has become more connected. With the increase in consumption of internet-connected devices, the security and integrity of the large amount of information passed across the globe have become a concern. Cyberattacks have become common, compromising the security, integrity, and privacy of the transmitted data. Cybercrime trends and attacks globally from 2012 to 2015 and proposed countermeasures to these attacks are discussed in [1]. The surge in cyber attacks during the COVID-19 pandemic is discussed in [2]. The impact that cybercrime has financially, politically, and on a military scale and their preventive methods are explored in [3]. Much research has been done to improve data security while in transmission [4]–[6]. However, moving away from traditional communication methods, it is often necessary to communicate covertly to counteract cyber threats, communicate in government intelligence operations, or the individual preferences for personal privacy and autonomy.

The application of steganography is a viable solution in these cases. Steganography is the technique of hiding data inside a text file, image, video, network packet, audio file, and so on [7]. It can provide invisibility and concealment of the data, be flexible across multiple forms of media without damaging or extensively changing the media, and be complementary with other forms of cryptographic techniques, adding additional layers of security.

Secret sharing protocols split confidential information into multiple shares, requiring a threshold number of shares to reconstruct the data. Hence, it offers resiliency against data compromise or loss by introducing redundancy. However, if these shares are sent across an open channel, attackers can intercept traffic, extract these shares, and reconstruct the data. These shares must be distributed among regular network traffic using steganography techniques to hide them. If multiple channels of covert data transmission are used, it is even more difficult for attackers to collect the shares to get the data.

The main goal of our research is to build such a model that can transfer confidential data through a network in a hidden and robust manner with the help of network steganography and secret sharing. In this paper, we transfer sensitive data on a LAN

using a combination of a secret sharing scheme, hashing, and network steganography, ensuring the confidentiality, integrity, availability, and secrecy of the data. The secret message is split into multiple shares by a threshold secret-sharing scheme. Each share is hashed, and the hash bits are appended to the shares, which are then sent to the recipient through multiple channels, employing network steganography to hide these shares.

1.2 Research Problem

Ensuring secure and covert communication in the era of increasing cyber threats has become a critical concern. Covert communication is pivotal in various scenarios where maintaining secrecy is imperative, such as in military operations, intelligence activities, and confidential corporate communications. It is vital to have secure and discreet communication, as unauthorized access or detection could jeopardize the success of the operation.

Traditional steganography techniques that hide data in the medium of transmission do not ensure high security if the data is not encrypted or reliability and integrity if the medium is discovered or altered.

If the data is sent through multiple channels of steganography, the data has to be ordered, and error detection mechanisms have to be used to check the validity. Secret-sharing schemes can be used to distribute the data and reconstruct the data at the receiver. However, an attacker can collect the shares and read the data if used in regular communication without encryption. As a result, there is a possibility of sensitive information being compromised.

Our investigation shows no existing research has explored secret-sharing combined with network steganography. This research intends to contribute to developing a more reliable solution for covert data transmission by integrating secret sharing, network steganography, and hashing techniques that will enhance cybersecurity measures.

1.3 Research Objectives

The primary objective of this research is to develop a novel framework that merges secret sharing and network steganography to send data covertly, which is also robust and secure. The objectives to achieve this goal are outlined as follows:

- Develop a hybrid model combining secret sharing and two network steganographic algorithms using ARP and TCP/IP.
- Enhance the security mechanisms of the steganographic algorithms to fit our model better.
- Combine the use of hashing and provide an integrity checking mechanism.
- Address possible vulnerabilities of the model and their solutions.

The rest of the paper is structured as follows: In Section 2, we examine related research and definitions related to secret-sharing schemes, network steganography, and recent work in other covert systems. Section 3 outlines our approach to methodically accomplishing our research and provides the flowcharts for our algorithms. In Section 4, we provide a security analysis of our model, details about the environment and implementation of our work, and the evaluation results of our model's algorithm.

Chapter 2

Literature Review

Before we describe our proposed model, a comprehensive literature review on secret-sharing and network steganography has been provided, and an overview of the algorithms of two network steganography techniques which are used in our work, is also included.

2.1 Secret Sharing

In general, secret-sharing is a data distribution technique where the data is defined as a secret and this secret is divided into shares, which are distributed among multiple participants. The important concept here is that the secret can only be reconstructed if a sufficient number of shares have been combined. This is set by the term threshold k as shown in Fig 2.1. There have been numerous research studies on this technique for various uses where there is a need for secure multi-party computation (some applications discussed in [8]), in cloud computing [9]–[11] or key management [12].

Shamir's Secret Sharing Scheme, also known as the k-n threshold sharing scheme, was developed by Shamir in 1979 [13]. This scheme works as follows:

Step 1: Choose a prime number P, which must be greater than the secret S.
Step 2: Choose a threshold value, k, the minimum number of shares required to reconstruct the secret S. The value of k must be less than or equal to n, the total number of shares to be generated.
Step 3: A random polynomial of degree $k-1$ is generated over the finite field of integers modulo P

$$f(x) = a_0 + a_1 x + a_2 x^2 + \ldots + a_{k-1} x^{k-1} \tag{2.1}$$

Here, a_0 is the secret S we want to share.

Step 4: Calculate n pairs of points on the polynomial at different values of x, where x is generally from 1 to n.

$$(x_1, f(x_1)), (x_2, f(x_2)), \ldots, (x_n, f(x_n)) \tag{2.2}$$

Each participant receives one of these shares as $(x_i, f(x_i))$

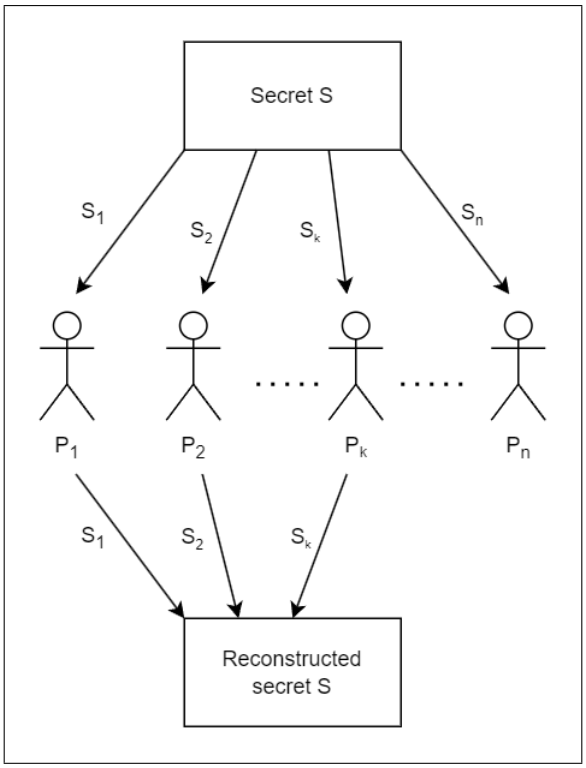

Figure 2.1: Visual representation of secret-sharing

For reconstruction, the Lagrange interpolation is used with minimum k shares

$$a_0 = \sum_{i=1}^{k} f(x_i) \cdot L_i(x) \tag{2.3}$$

Where:

a_0 : The secret to be reconstructed.
x_i : The x-values from the k shares.
$f(x_i)$: The corresponding y-values (shares).
$L_i(x)$: Lagrange basis polynomials for each share.

$L_i(x)$ is calculated by:

$$L_i(x) = \prod_{j=1}^{k, j \neq i} \frac{x - x_j}{x_i - x_j} \tag{2.4}$$

Numerous studies have been conducted on the application of secret sharing for information security. Krawczyk(1994) modified Shamir's secret sharing scheme to propose a new scheme that is computationally secure [14]. His scheme used a pseudo-random function, which allows for the efficient generation and distribution of secret shares among a group of participants. Beimel, A. (2011) has compiled more secret sharing schemes in [8], among which are Ito, Saito and Nishizeki's Constructions, Monotone Formulae Construction, and Multi-Linear Secret Sharing Scheme are noteworthy. Harn et al. (2020) proposed a threshold secret-sharing scheme where the secret remains safe from malicious actors in the reconstruction phase [15]. In traditional secret-sharing schemes, an attacker can reconstruct the secret if k

shares can be intercepted. Their proposed scheme requires the attacker to intercept all the shares to reconstruct the secret. Lee (2018) proposes a method for secret communication that utilizes Shamir's secret sharing scheme [16]. He divided the secret message into shares and used a public website to hide the shares among numeric data. The proposed method also includes a self-authenticable scheme on the receiver side to ensure the integrity of the secret message.

2.2 Prisoners' Problem

The prisoners' problem relates to a situation where two people need to communicate over a public network so that even though anyone can access the messages, the hidden message can only be understood by the sender and receiver [17]. In the Fif 2.2, Alice and Bob are prisoners, and the warden watches all their activities. The warden allows them to use computers to communicate and disclose their escape plan. The warden checks all the messages passed between them, called the passive warden, and can even modify them to trap them, called the active warden. Alice and Bob know this and communicate with each other without any use of cryptographic measures; thus, the warden is very knowledgeable about the contents of the messages. In such a situation, Alice and Bob have to devise a technique to hide their plan in a plain message and ensure that only they can find and understand it.

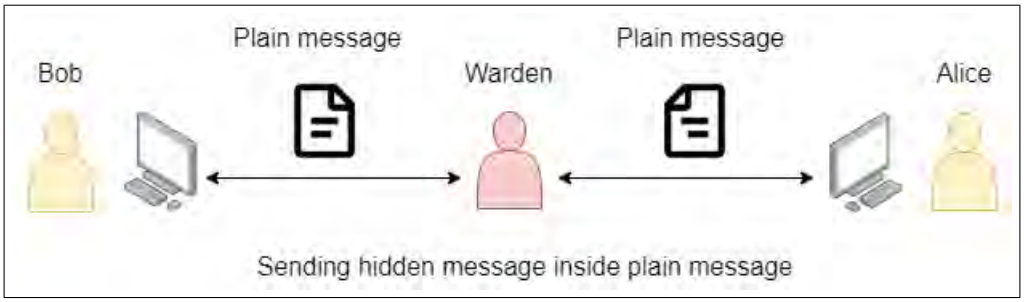

Figure 2.2: Prisoner's Problem

The solution to this problem is to apply a method where their plan is hidden from plain view and the messages passed between them seem innocuous. Steganography helps to achieve the objective here. Steganography hides secret data in plain text, audio or video files, images, or network packets. Network steganography is the technique where a covert channel or medium hides secret messages in the regular network traffic. Bob and Alice can use network steganography in different ways to communicate with each other secretly. For example, they can hide their plan in the TCP or IP header section or other network packets. They can also use the time delay between subsequent packets to convey their plan. Distinguishing network steganography-implemented network traffic from regular traffic can be challenging, and the usual detection method cannot distinguish them. Further study on network steganography has been detailed below.

2.3 Network Steganography

Information has become our most valued asset in the current age, where technologies are becoming our needs. Scientists and researchers worldwide are working day and night to find a way to keep this valuable asset from falling into the wrong hands. Here, steganography plays an important role in ensuring that security. Steganography is the technique of hiding data inside a text file, image, video, network packet, or audio file. There are five ways to implement steganography. Network steganography is one of them. It represents a covert communication method that secretly incorporates regular traffic to send data over an unreliable network. It uses different protocols to implement covert communication. There has been ongoing research about ways to hide messages using the protocols used for data transmission.

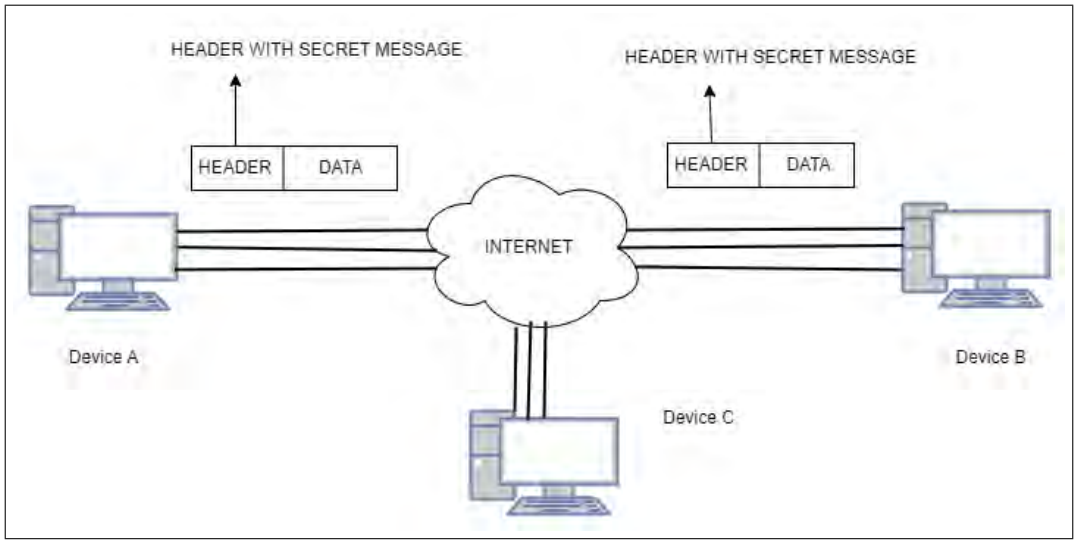

Figure 2.3: Visual representation of network steganography

In Fig 2.3, three devices are connected through the internet, and device C is a malicious device that wants to eavesdrop on the messages between them. Now, device A and device B want to communicate in such a way that the malicious device C does not know that they are secretly communicating. So, device A embedded regular network packets with hidden messages and sent them to device B. During this data transmission, even if device C followed network traffic, it would not be able to find that those network packets contained data. As a result, receiver B receives the packets containing secret data without anyone knowing them.

Data can be hidden in the network packet using a method called StegBlocks which is described in [18]. Two types of StegBlocks have been explained: StegBlocks TCP and StegBlocks SCTP. The TCP or SCTP header packet contains the hidden information. They developed a method to integrate the data in the packets such that the packets with hidden data and the normal packets seemingly blend in. Another paper [19] used the DNS protocol to hide data. They proposed using the header and answer fields of DNS message packets. Later on, other protocols like ARP in [20], and IPv4 in [21] were used with network steganography to transfer information undetected. The idea in [20] was to use the ARP reply packet. This method was

developed for a LAN network where the sender and receiver can execute the technique simultaneously. The idea of using the overflow field in the timestamp option of the IPv4 protocol, which can carry 4 bits, was described in [21]. They then sent the packet with hidden data using the UDP protocol, a faster transmission protocol, though TCP can also be used to send the packet.

2.3.1 ARPNetSteg

ARP is a protocol used in local area networks to map IP addresses to MAC addresses. The sender inserts the covert message in ARP packets in fields like the sender's hardware (MAC) address, padding bits, or other non-critical sections of the ARP packet, ensuring the packet's outward appearance remains primarily unchanged. This steganography technique is used in Local Area Networks (LAN). The Mac address of the sender is used to hide the covert message in [20]. First, the receiver sends an ARP broadcast request. When the sender gets the request, it sends an ARP reply with the message hidden in it. The sender divides the message into several partitions of 44-bit length, and each 44-bit covert message portion is used to make the Mac address. A total of 44 bits is hidden here, and the last 4 bits of the MAC address work as the control flag that checks whether there is any other ARP reply with a covert message portion. Upon receiving the reply packet, the receiver retrieves the message using the same algorithm used to hide the message.

Below is an overview of the sender-side algorithm:

Step 1: A covert message is taken as input as a string, and this string is converted to hexadecimal code.

Step 2: An ARP request is broadcast to discover all the allocated and unallocated IP addresses on the LAN. Based on the replies, an allocated list is created. An unallocated list is created for the IP addresses for which no reply was received.

Step 3: A random number between 1 and 255 is generated using a seed value, and a local IP address is created using this value. This IP address is checked to see if it is on the unallocated list. If not present, then step 3 is repeated.

Step 4: Wait for an ARP request from the covert data receiver device. Upon receiving the request, an ARP reply is crafted from the sender side by embedding the hexadecimal covert data into the first 44 bits of the sender hardware address and the last 4 bits for the control quad of the ARP reply. Also, the source IP, which is the IP of the ARP request receiver, will be the IP created using the randomly generated number.

Step 5: The ARP reply will be sent by repeating steps 3 and 4 until the receiver has received all the covert data.

Below is an overview of the receiver-side algorithm:

Step 1: On the receiver side, first, they initialized a covert message to NULL.

Step 2: Like the sender-side device, it discovers all the allocated and unallocated IP addresses on the LAN. Two lists are created to separately store the allocated and unallocated device IP addresses.

Step 3: A random number between 1 and 255 was generated using the same seed value used by the sender device, and using this value, a local IP address was created. This IP address was checked to see if it was present in the unallocated list. If not present, then step 3 was repeated.

Step 4: An ARP request is made using the local IP address made in Step 3 as the Target IP address and waits for an ARP reply.

Step 5: Upon receiving the ARP reply, the receiver side will start to process the covert data embedded in the sender hardware address field of the ARP reply. The control quad in the last 4 bits of the sender hardware address is checked, and if it is '0' the first 44 bits of the source MAC address are extracted and appended to a string variable, and an ARP request is broadcasted. If the control quad is 'f', the first 44 bits are appended; otherwise, the control quad is used to find the padding bits, where the padding bits are discarded and the remaining bits are appended. No more ARP requests are broadcasted after these two cases, so the process will continue until all the covert data is received. It will be converted from a hexadecimal value to a string upon receiving complete covert data.

2.3.2 Moving IP steganography

Instead of hiding the secret data in network packet fields, [22] proposed an approach where, instead of inserting any data, the data is extracted using a deterministic algorithm that utilizes hashing and the concept of permutation. The algorithm also requires user inputs: evaluation bits, a secret key, and a retry count. Evaluation bits define the number of bits to be hidden in each packet. The secret key generates the permutation array of random integers ranging from 0 to 255 if SHA256 hashing is used. The retry count specifies how many times the sender would send the secret message if the receiver could not extract the exact message.

The sender-side algorithm in is described below:

Step 1: The secret data is combined with a counter, the first four characters, the last four characters of the hash of the secret data, and a dash '-' at the end. The hashed bits are included for integrity verification on the receiver's side. The formatted data is converted to binary for transmission.

Step 2: A permutation array generator, seeded with the secret key, generates an array the same size as the number of evaluation bits, consisting of random integers from 0 to 255, inclusive. For every marked packet, a new permutation array is produced.

Step 3: A TCP/IP packet stream is generated for covert communication if there is no existing communication channel.

Step 4: For each packet, a hash is calculated with the following parameters: source IP address, destination IP address, source port number, destination port number, sequence number, and the data. This hash value is converted to binary.

Step 5: Using the permutation array values as indices, values in binary packet hash are compared with the binary formatted data, shown in Fig 2.4. Only if all the values match will the packet be marked in step 6.

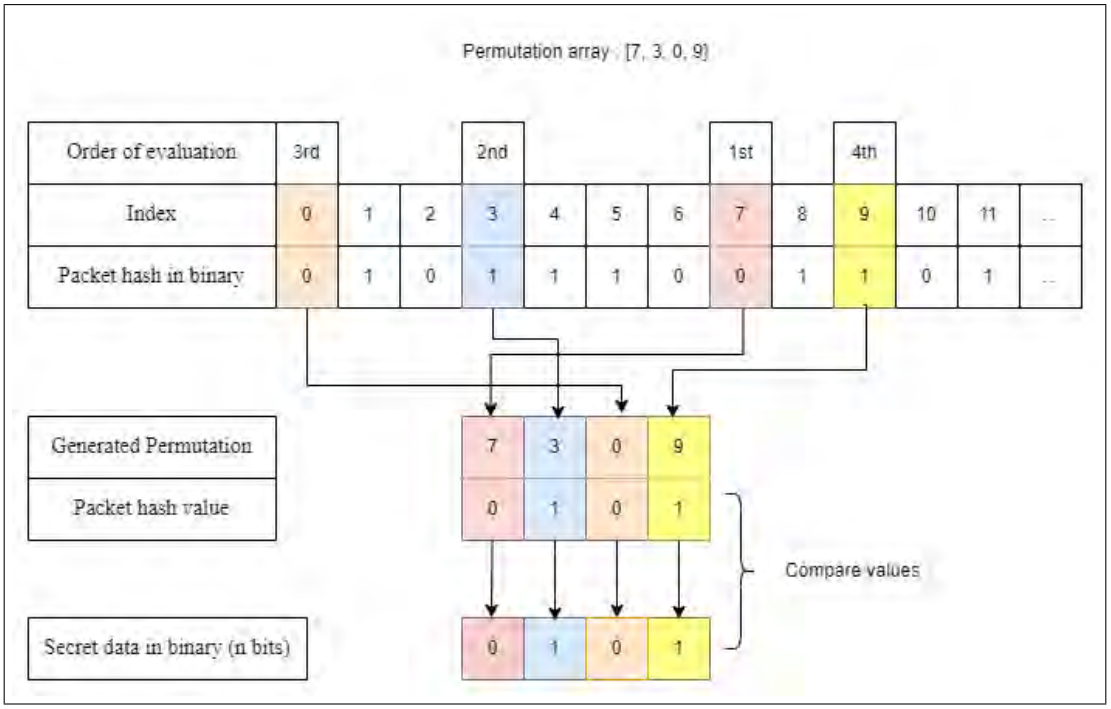

Figure 2.4: Packet hash evaluation to check if it can hold covert data

Step 6: The packets that need to be marked are done by activating the packet's TCP PSH (push) flag.

Step 7: Both marked and unmarked packets are sent to the receiver. If all the data is sent, the sender waits for an acknowledgement from the receiver. Otherwise, go back to step 2. If the integrity check fails at the receiver end, the receiver sends a negative number as acknowledgement data; the process is repeated from step 2 based on the retry count.

The author mentioned using a more covert approach to mark the data packets as future work. The algorithm is named Moving Target Network Steganography because the target bits where the data lies change for every marked packet, making it difficult for the attacker to guess the bit positions and their order.

Following is the receiver-side algorithm:

Step 1: The receiver generates the same permutation array using the known secret key to seed the permutation array generator. The size of the array equals the number of evaluation bits.

Step 2: The receiver keeps sniffing for incoming network packets and checks if the packet has the push flag activated. If a marked packet is found, the algorithm moves to step 3.

Step 3: Using the same parameters as the sender, the receiver calculates the hash of the packet and converts the hash value to binary.

Step 4: Extract the secret data from the binary packet hash by using the integers in the permutation array as indices shown in Fig 2.5.

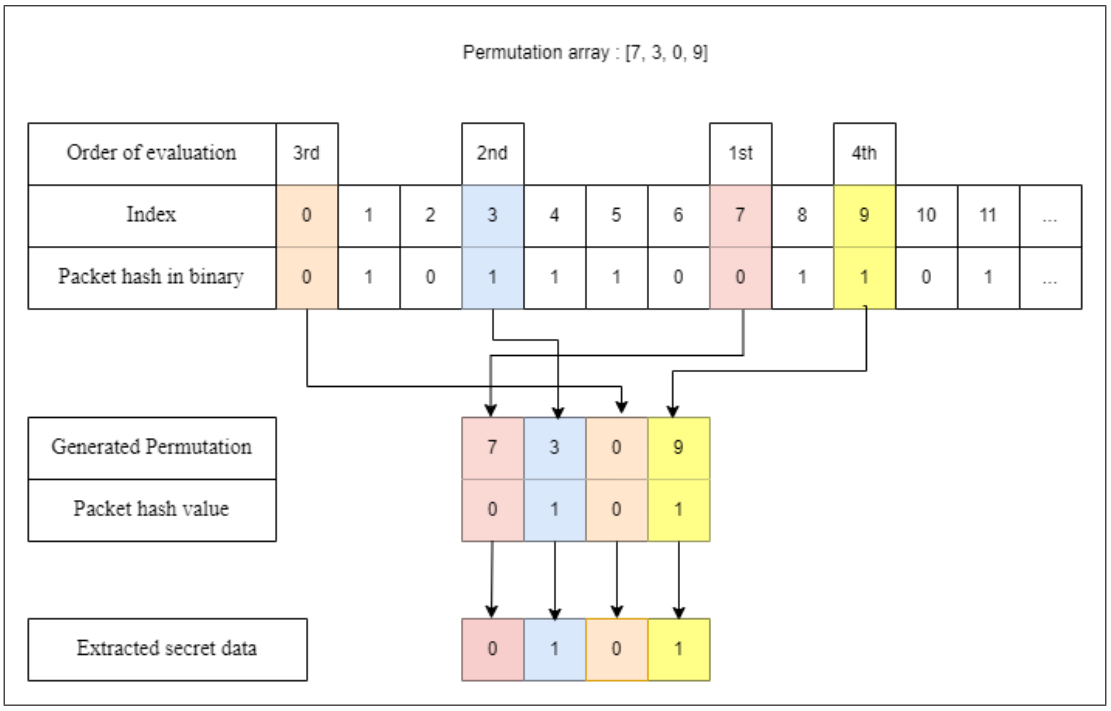

Figure 2.5: Packet hash evaluation to extract hidden bits

Step 5: The receiver reconstructs the complete binary string and checks if '-' is received, which means the end of transmission.

Step 6: The integrity of the data is checked. The hash of the message is produced, and the first four characters and the last four characters of the hash are compared with the provided hash. If the hash does not match, the receiver sends a negative number, which indicates the secret message was not received correctly. Otherwise, an acknowledgement is sent to the sender.

For each packet, the generated permutation array has no sequence pattern. Hence, brute force is not feasible for an attacker. Statistical analysis provides no result

since no data is hidden in the packet.

The time taken to send the data is proportional to the number of evaluation bits or the number of bits that are sent per marked packet. On the other hand, the data transfer rate is inversely proportional to the number of evaluation bits. A higher number of evaluation bits provides more security since an attacker must guess more hash positions. While choosing the value of evaluation bits, there is a trade-off between security and data transfer rate.

2.4 Covert Transmission Systems

This section covers existing research on other covert and secure data transmission models.

IoT-based Wireless Sensor Networks (WSNs) are lightweight, secure multi-hop data routing using a secret sharing scheme that is energy efficient [23]. In this method, sensor zones are first divided into inner and outer zones using the KNN algorithm. Then, the data is encrypted using a lightweight secret sharing scheme where an XOR operation is done on the data and the zone key at every zone. The data is decrypted in the same way. This method is energy efficient as a simple operation does the encryption method, and by using multi-hop routing, data is safe from data threats and malicious attacks.

A covert channel that is based on the concept of packet switching is presented in [24]. A modified TFTP (Trivial File Transfer Protocol) application is used for sending secret messages or data and demonstrates integrity improvement. However, the authors also highlighted the risk of leaking client data files without user notification. The paper also introduces a sliding entropy method that detects some cases of covert channels.

Distributed network covert channels have been analysed in [25]. They made a modified Distributed Network Covert Channel (DNCC) scheme where three different steganographic techniques are used for data hiding and secure transmission.

Data transmission using Dynamic Host Configuration Protocol (DHCP) is discussed in [26]. It incorporates three distinct covert channels using fields within the DHCP protocol, including xid, Sname and File, and Options, each providing different requirements regarding stealthiness and data capacity. The paper also provides the method's reliability and likelihood of detection.

A covert communication method based on Bitcoin transactions, hence blockchain steganography, is proposed in [27]. The experimental results demonstrated that the proposed method maintains the necessary security measures while being robust and resistant to detection.

Chapter 3

Methodology

3.1 Working Plan

The main goal of the proposed new system is to ensure that data is transmitted over the network covertly, robustly, and feasibly. We would use a secret-sharing scheme combined with hashing for integrity validation and create covert channels to ensure a covert method to send sensitive data over a LAN. To our knowledge, no such approach has been tried before.

Our proposed system is divided into three parts: Secret Sharing, Hashing, and Network Steganography.

From the sender side, as shown in Fig 3.1, the data to be transmitted is divided into shares using a secret-sharing scheme. This mechanism has two variables: the total number of shares that will be generated n and the threshold value k, which are known to both the sender and the receiver. At the receiver's end, the reconstruction phase only requires a minimum of k shares to reconstruct the message. Considerations must be taken when assigning the value of n and k because a high value for n and k means the computational overhead increases due to the computational resource required to create those n shares and also recreate the secret using k shares, while a small value for n and k means the secret-sharing scheme cannot provide sufficient security because an adversary might get the k shares very easily.

We hash the share and append the hash value to it for every share. The shares are then sent to the receiver through multiple covert channels, which are network steganographic algorithms in our implementation, where one channel should be used to send less than k shares to ensure that if one channel gets compromised, the attacker does not possess the threshold amount of shares to reconstruct the secret message.

On the receiver side in Fig 3.1, the shares and their hashes are extracted from the covert channels. The receiver generates the hash value of the shares and compares it with the hash value sent by the sender. A share is only valid if the hashes match. When k or more valid shares are received, the reconstruction algorithm of the secret-sharing scheme is used to produce the secret message.

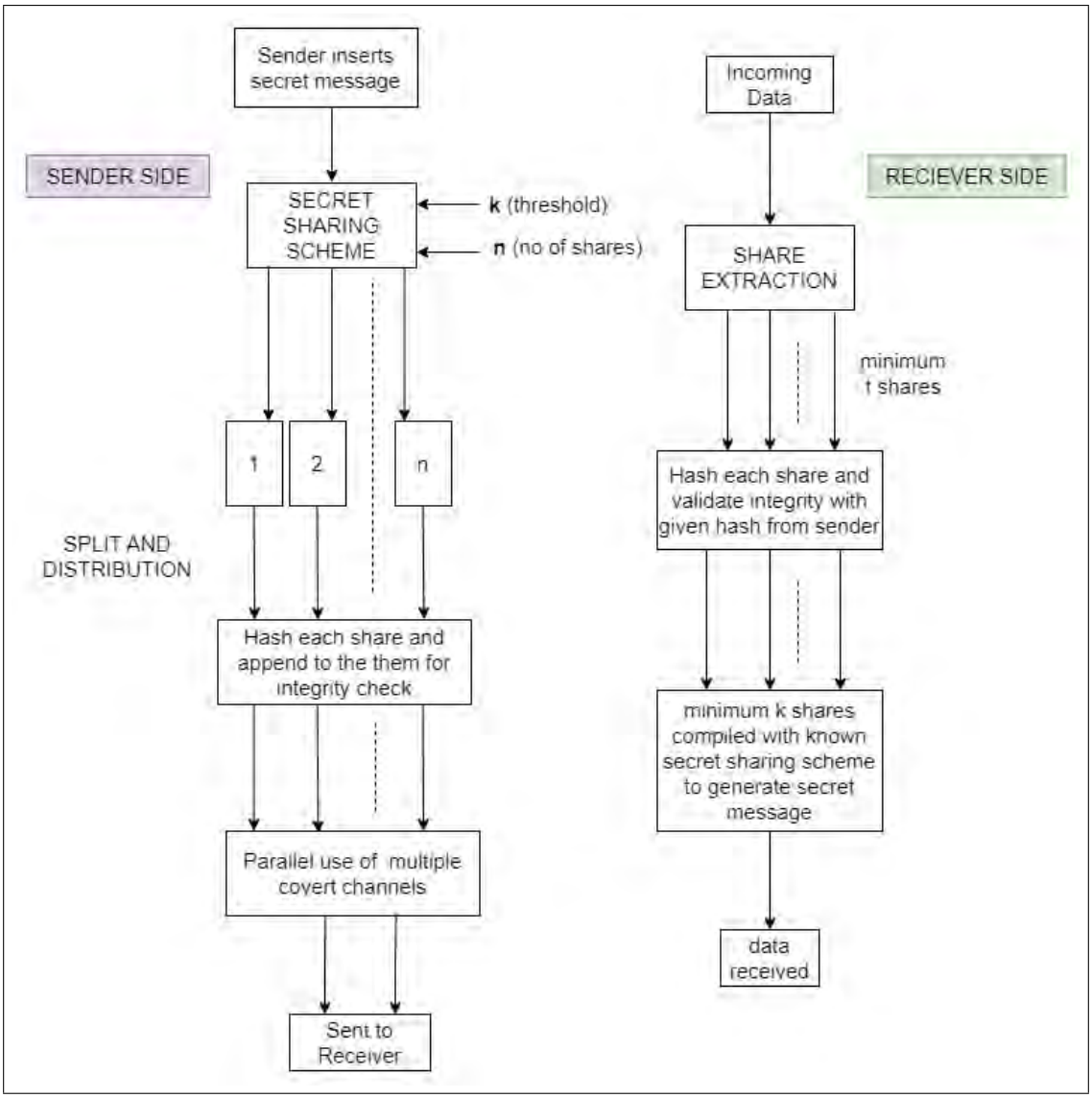

Figure 3.1: Proposed model for sender and receiver side

3.2 Algorithm

3.2.1 Modified ARPNetSteg algorithm

Using the algorithm provided in [20], we have modified it to integrate with our model. Initially, the sender and receiver must generate a list of allocated IPs and unallocated IPs in the LAN. After creating the lists, a previously defined seed value is used by both the sender and receiver to generate a list of random local IP addresses. The sequence of the address in the random IP list is the same on both sides due to the same seed value. The receiver selects the first IP in the random IP list and checks it is in the unallocated IP list, as shown in Fig 3.2. If the result is False, the process is repeated until the result is True. This IP address is used to create and send an ARP broadcast request from the receiver with the target IP address as the selected random IP. On the sender side in Fig 3.2, the first IP in the random IP list is chosen and checked to see if it is in the unallocated list. The process is repeated until the result is True. The sender now waits for an ARP request from the receiver.

The ARP request packet must have the destination IP as the selected random IP.

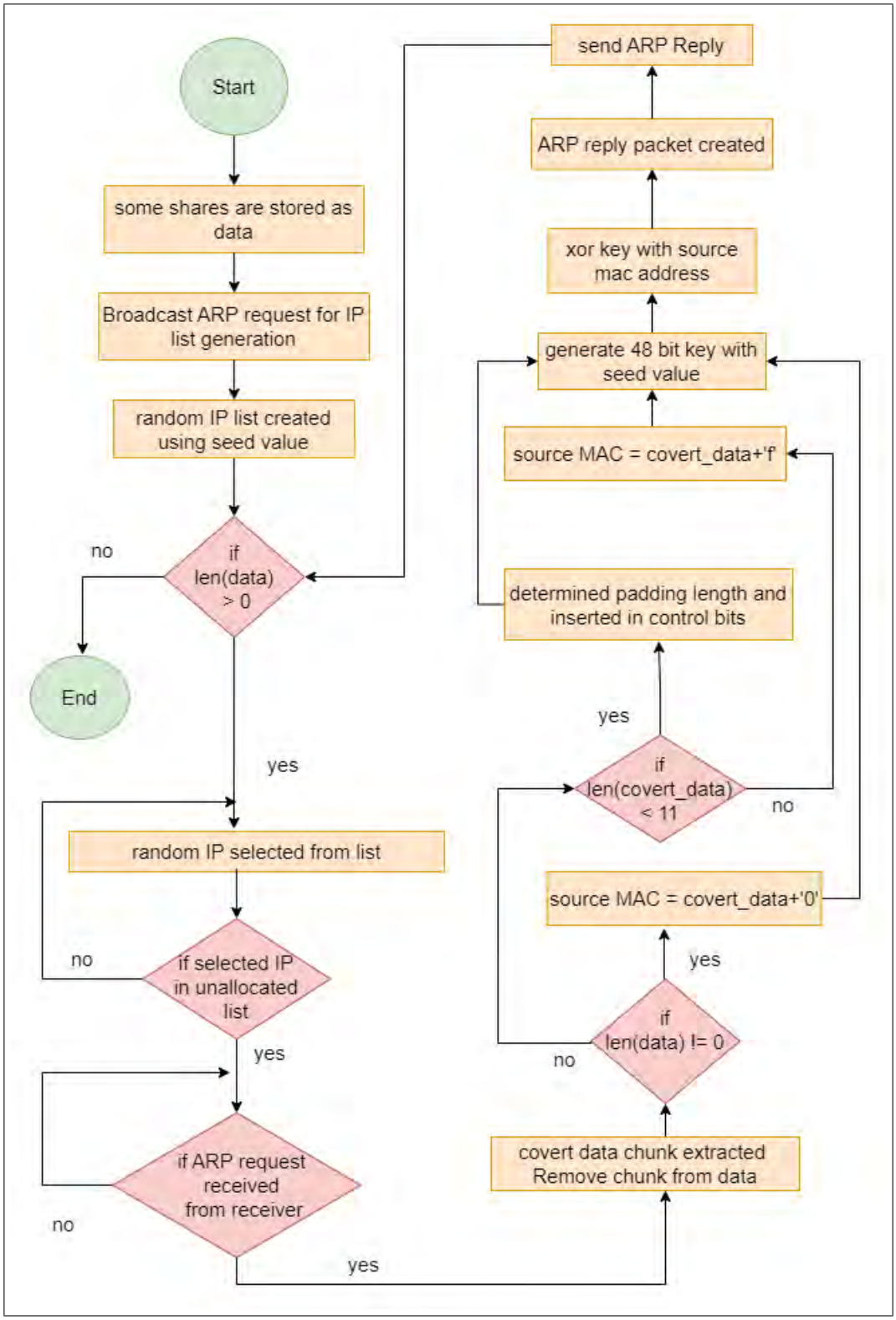

Figure 3.2: ARP steganography flowchart for sender side

As the receiver broadcasted a request for a MAC address using an unallocated IP in the LAN, the sender will be the only one to create a spoofed ARP reply for this request.

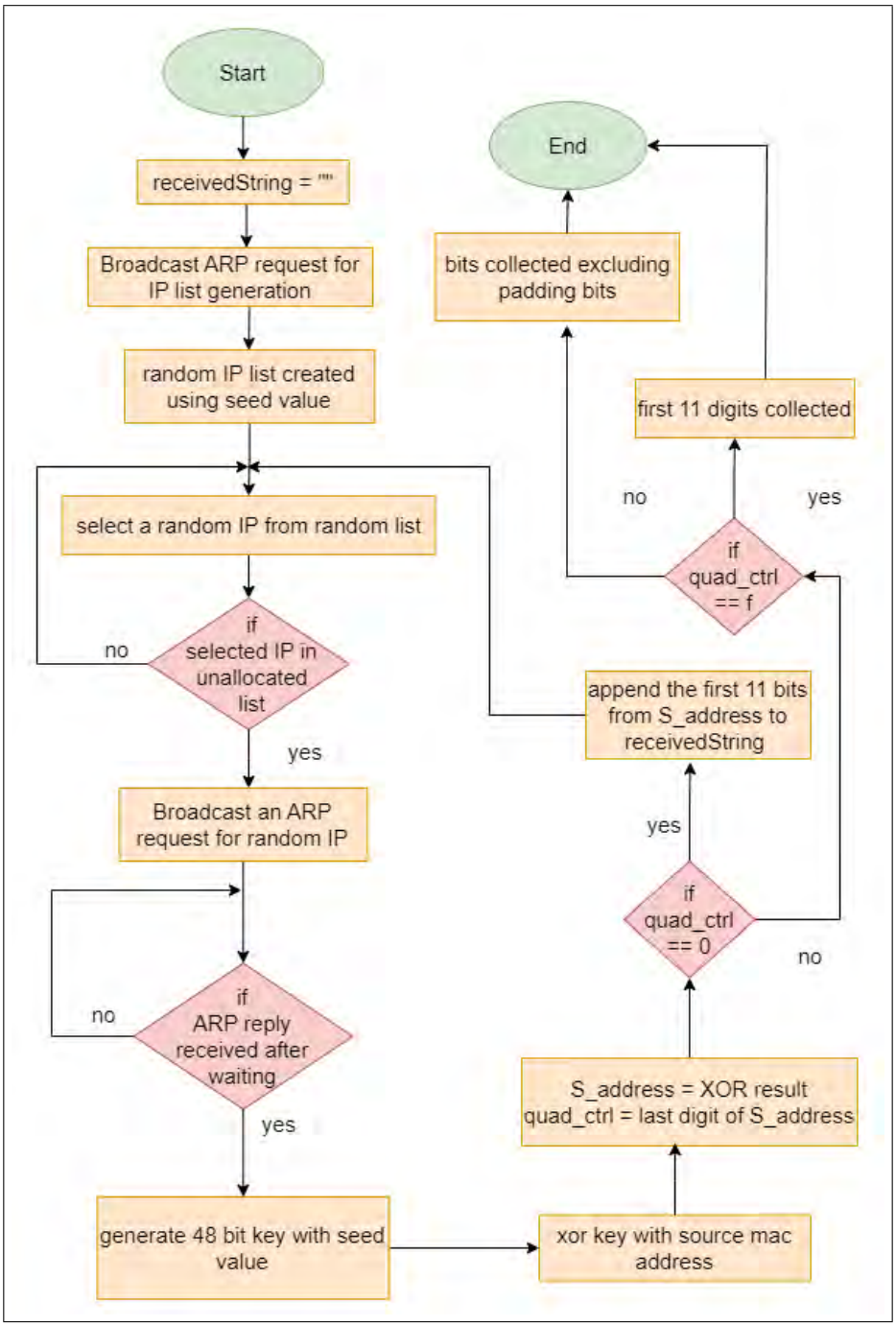

Figure 3.3: ARP steganography flowchart for receiver side

The sender sends the covert data by impeding it into the MAC address of the source hardware address field of the ARP Reply packet. The hardware address is 48 bits long. The covert data is stored in the first 44 bits of the Ethernet address. The last 4 bits are used to send a control quad. From the combined string, the first 11 digits (the shares were in hexadecimal) are chosen, and the control quad is set to '0', indicating more data to be sent. If only 11 digits need to be sent and there is no more data, the control quad is set to 'f', indicating that all data has been sent. If the last n digits are to be sent where n is less than 11, $(11-n)$ zeroes are padded

to the covert data to be sent, which is the control quad in this case. The source IP is set as the selected random IP, and the destination MAC and IP are set to that of the receiver extracted from the ARP request packet.

The seed value is used to generate a 48-bit key pseudorandomly to encrypt the 48-bit data by an XOR operation. This adds a layer of security if the steganographic medium is compromised. A new 48-bit key is generated for every packet to avoid any pattern in the resulting 48-bit ciphertext. Then, the 48-bit ciphertext is inserted in the ARP reply packet in the source MAC address field and sent to the receiver.

Once an ARP reply packet is received by the receiver in Fig 3.3, the seed value is used to generate a 48-bit key. The MAC address bits are extracted and XORed with the key. It returns the actual hexadecimal covert data along with the control quad that was sent. After that, the control quad is checked, and if it is '0', the first 44 bits of the source MAC address are extracted and appended to a string variable, and an ARP request is broadcasted. If the control quad is 'f', the first 44 bits are appended; otherwise, the control quad is used to find the padding bits, where the padding bits are discarded and the remaining bits are appended. No more ARP requests are broadcasted after these two cases.

3.2.2 Modified MTNS algorithm

The MTNS algorithm for the sender and receiver [22] has been configured to integrate with our model to send the shares seamlessly. Fig 3.4 shows the modified MTNS algorithm for the sender side. A detailed overview of the algorithm is described below.

Step 1: The data to be sent is passed to the IPSteg algorithm. An array of random integers of size $n + 1$, called the permutation array, is generated, where n is the number of evaluation bits.

Step 2: Random bytes of random size are generated, which will go into the data payload of the TCP/IP packet.

Step 3: A TCP/IP packet is created with the following parameters: sequence number, source port number and IP address, destination port number and IP address, and the data payload. This packet is then hashed using SHA-256 and converted to binary for bit comparison.

Step 4: The first n integers generated in the range 0 to 255 inclusive in the permutation array serve as indexes in the packet hash and are compared against the data in binary format. If all the index position values match against the data values, we try to mark the packet.

Scenario 1: If all the data bits match, the hash bit must be 1 at the index: the last integer of the array.

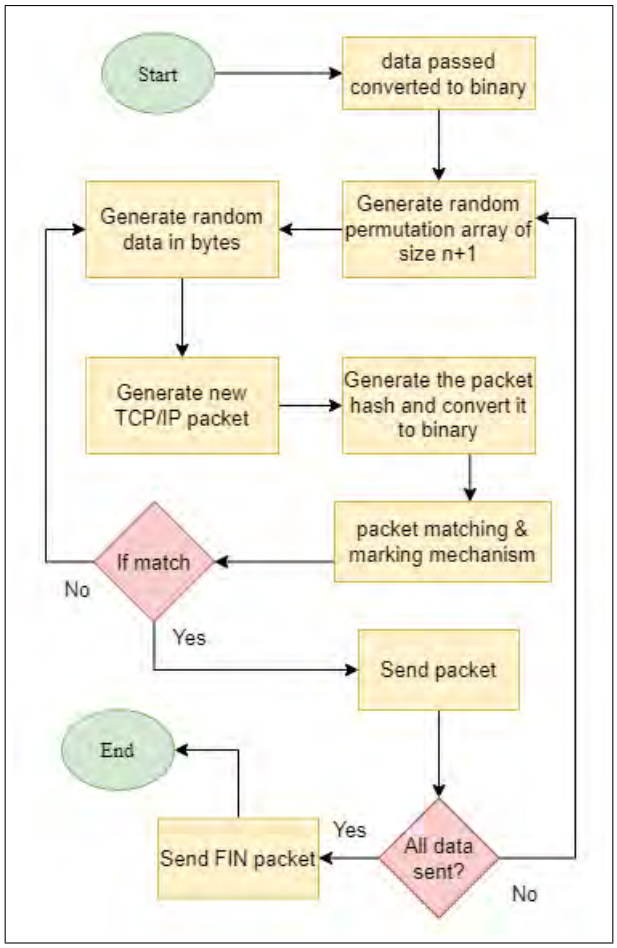

Figure 3.4: Modified MTNS flowchart for sender side

Scenario 2: If the data bits do not match, the hash bit must be 0 at the index: the last integer of the array.

Step 5: For scenario 1, if the bit is 1, or for scenario 2, if the bit is 0, only then the packet is sent. Otherwise, the process is repeated from step 2. This is why generating random data bytes is essential because this is the only parameter that can be varied, causing the hash value to change as well. If the hash value remained constant, all the bits could never be matched, and the algorithm would be stuck in a loop.

Step 6: If no more data is left to send, send a FIN packet to mark the end of transmission.

We have excluded the retry process from the original algorithm since our model considers invalid data received at the receiver end by adding redundancy using secret sharing, making it more robust. We have also suggested a new approach to marking the packets in Step 5 without activating the PSH flag of the TCP/IP packet. No flag has to be activated in this approach, making the transmission more covert.

The modified MTNS algorithm for the receiver shown in Fig 3.5 is described below.

Step 1: Generate the permutation array and sniff for incoming packets.

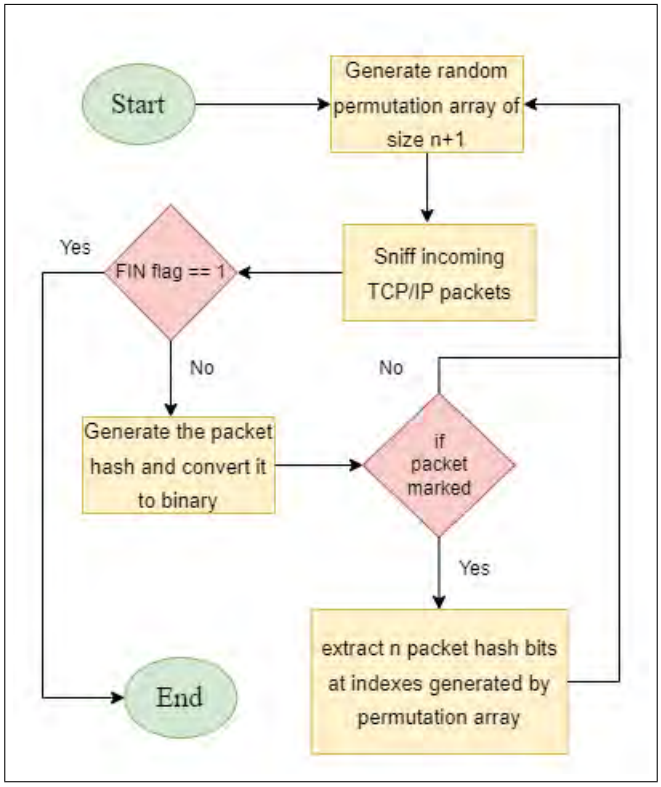

Figure 3.5: Algorithm flowchart for receiver side

Step 2: Check whether the FIN flag is activated once a packet is received. If yes, exit the algorithm. Otherwise, proceed to step 3.

Step 3: Hash the packet using the same parameters as the sender and convert it to binary format.

Step 4: Check if the bit is 1 at the index given by the last integer of the array to extract n bits using the values from the permutation array as indexes. Otherwise, proceed to step 1.

Unlike in the original algorithm, the data received is not validated inside the modified MTNS algorithm. Validating the data, i.e., the shares, is left to the main program instead. Only the data is extracted and passed on to the main program.

3.2.3 Model Algorithm

As outlined in Algorithm 1, the model requires pre-determined variables known to the sender and the receiver: a secret key or seed value, the number of evaluation bits for the IP steganography algorithm, and the threshold value k for the secret-sharing scheme. The sender must also provide n, the total shares produced in the

secret-sharing scheme.

The sender inputs a message, which is split into n shares with a threshold value of k using Shamir's secret sharing scheme. The original message can only be reconstructed using a minimum of k shares. A higher value of k improves security because an attacker must collect more shares to reconstruct the secret. A higher value of n adds redundancy, which provides fault tolerance, meaning that if some shares are lost or modified, the secret can still be reconstructed by brute force if possible if there are k valid shares by trying all combinations possible, but this may increase computational needs, making the defined scheme impractical.

Algorithm 1: Algorithm for Sender Side

Data: Original message to be transmitted
Input: Secret key or seed value, Number of evaluation bits i for modified MTNS, Threshold value k, Total number of shares n
Output: Shares sent through covert channels

Initialization:
userSeed = seed value or secret key;
evalBits = i ;
threshold = k ;
totalShares = n ;
Create Shares using Shamir's Secret Sharing:
Split the message into n shares with a threshold value k;
Note: A higher k value enhances security, and a higher n value adds redundancy;
Hash and Append to Each Share:
foreach *share* **do**
    Compute SHA-256 hash of the share;
    Append the first and last four hexadecimal digits of the hash to the share;
end
Send Shares Through Steganography Channels in Parallel:
Divide shares equally into two groups for parallel transmission;
Parallel Process 1: Transmit the first group of shares through modified MTNS channel;
Parallel Process 2: Transmit the second group of shares through modified ARPNetSteg channel;
Note: Both groups are transmitted simultaneously using multiprocessing;
Handle ACK Packet:
if *ACK packet is received* **then**
    Stop sending further data if there is any;
    Transmission is successful;
else
    Consider the transmission as failed;
end

To solve this problem, each share is hashed using a cryptographic hash like SHA-256. The first and last four hexadecimal digits are appended to the end of each share. This approach provides an efficient way to validate the shares on the receiver end. These shares are then sent through multiple covert channels to the receiver, where less than k shares are sent through each channel. For our algorithm, we have used two network steganographic algorithms described in Subsections 3.2.1 and 3.2.2.

Algorithm 2: Algorithm for Receiver Side

Input: Secret key or seed value, threshold k number of shares, evaluation bits i for the modified MTNS

Output: Reconstructed message if successful

```
1  Receive Shares:
2  Collect shares sent through modified ARPNetSteg and modified MTNS
     channels;
3  Validate Each Share:
4  foreach received share do
5  |  Separate the share from its hash bits;
6  |  Compute SHA-256 hash of the share;
7  |  if hash matches with the appended hash bits then
8  |  |  Mark the share as valid;
9  |  else
10 |  |  Discard the share;
11 |  end
12 end
13 Reconstruct Message:
14 if at least k valid shares are collected then
15 |  Reconstruct the message using valid shares;
16 |  Send an ACK packet back to the sender;
17 |  return Reconstructed message;
18 end
19 Failure Check:
20 if less than k valid shares are collected then
21 |  Indicate failure in message reconstruction;
22 |  return Failure notification;
23 end
```

The shares and their corresponding hash bits are separated at the receiver end in Algorithm 2. Each share is hashed, and the hash bits are compared with the first and last 16 bits of the hash value. Once k valid shares are extracted and data is reconstructed with the secret-sharing scheme successfully, the receiver sends an ACK packet back to the sender. If the sender receives the ACK packet, it stops sending more data. However, the secret transmission has failed if no ACK packet is received.

Chapter 4

Analysis and implementation

4.1 Security analysis

4.1.1 Secret-Sharing

Shamir's secret-sharing scheme in the model splits the secret message into multiple shares. Each individual share holds no semantic meaning or readable data. The message can only be reconstructed if the user has the threshold number of shares or more.

k = 4 , n = 7	k = 5 , n = 7	
1-390a1400234	1-58ae8d5673f	1-53ff47566f9
2-2dae573130d	2-3724380ce5a	2-7de692747bf
3-17b3a6065e7	3-8964f03bd1a	3-5d314b91287
4-2b98d8c94a	4-2a6b9fa2d6b	4-5b42e4c0d45
5-1359d6aa541	5-3b89b69bee8	5-5ba8f9ec6d6
6-37577f31f4	6-5ad2946b030	6-469e41eaa3d
7-306a94ed44b	7-42a45a8989d	7-90a8e7fea3

Table 4.1: Generated shares using Shamir's secret sharing scheme

Table 4.1 demonstrates the creation of the shares from the message "Attack" with a specific value for total shares n and threshold k. Shamir's secret sharing has its randomness in share generation, where the shares change every time the same secret is split, even if the parameters for the scheme are kept constant.

The message is not reconstructed correctly if less than the threshold number of shares is used. The scheme provides security with its randomness in share generation as well as with the threshold value. Secret-sharing also introduces fault tolerance in the model by adding redundancy in the data transmission, meaning even if some

shares are corrupted or lost, the secret can still be reconstructed if k valid shares are present. Users can customize the security level by adjusting the values of k and n. A higher value of k increases security, while a higher value of n adds redundancy.

However, there are two weaknesses to this scheme:

1. An attacker might acquire all the shares and be able to reconstruct and read the message if the shares are sent through only one communication channel without encrypting or hiding these shares.

2. A lower k compromises security, while a higher n might lead to computational inefficiency. The model might only be robust if $n - k$ is substantial.

Our model solves the first problem by using multiple covert channels. The shares are distributed and hidden in the communication protocol. Now, the attacker has to figure out how the shares are hidden and collect a minimum of k valid shares by compromising multiple channels, making the task even more difficult.

For the second problem, a tradeoff between redundancy and security has to be evaluated to find the optimal value for n and k.

4.1.2 Hashing and Integrity Verification

Cryptographic hash functions have various applications in the security domain, including message authentication, integrity verification, and digital signatures [28]. As such, these hash functions must have the following properties: collision resistance, pre-image resistance, and second pre-image resistance. SHA-256 is a cryptographic function that satisfies all requirements and is widely used in security systems.

We have used SHA-256 in the modified MTNS algorithm for integrity validation. We have hashed each share, appended the first and last 16 bits (32 bits in total) to each share, and then sent the shares through network steganographic channels to validate the shares on the receiver end. The choice to use only 32 bits instead of the whole hash value is to make a compromise between transmission efficiency and security. The data to be transmitted becomes too big if the whole hash is added, which makes the model inefficient. Relying on the collision resistance property and the avalanche effect of SHA256, we use only 32 bits.

SHA-256 produces a 256-bit hash value, which means 2^{256} possible hash outputs. This is a huge number, making the possibility of two matching hashes of two inputs scarce. Considering the 32 bits of the hash, the probability for a specific bit of two hashes is 0.5 or $\frac{1}{2}$. The probability that 32 bits of two hashes will match is $\left(\frac{1}{2}\right)^{32}$ or $2.328306437 \times 10^{-10}$. This is an astronomically small number, sufficient to reason the use of 32 bits from the 256-bit hash.

4.1.3 Modified MTNS

A probabilistic analysis of the MTNS algorithm on the possible combinations for evaluation bit size and of the attacker figuring out the correct permutation of each

packet has been provided in [22]. The permutation array is of length n, consisting of n integers in random order in the range of 0 to 255, including. Possible combinations are calculated using:

$$P(n, r) = \frac{n!}{(n - r)!} \tag{4.1}$$

Bits to match	Number of possible combinations
4	4,195,023,360
5	1,057,145,886,720
8	16,517,640,193,528,320,000
9	4,096,374,767,995,023,360,000
16	210,875,602,102,456,269,086,537,616,669,081,600,000

Table 4.2: Number of possible combinations for number of bits to match

Even if the attacker is entirely knowledgeable about the implementation and algorithm, they will still be unable to recreate the exact permutation that was used to evaluate $n + 1$ bits of secret data without the shared secret key because the permutations are generated using a random number generator that is seeded with the shared secret key. Furthermore, because the permutation changes every n bits of data, the attacker has limited time to predict and generate the permutation array before it changes.

When the PSH flag is activated, the attacker can identify which packet contains data in the original algorithm. Our model covertly hides the marking by the hashing technique to match the data. Hence, there is no existing pattern for the attacker to realize that data could be covertly transmitted here.

The algorithm's security critically lies on the shared secret key and the structure of the permutation array generator. The covert channel will also be compromised if the key is not secure. The permutation generator should produce pseudorandom values with no specific pattern; otherwise, the attacker may be able to guess the indexes where the data lies, making the algorithm less secure.

4.1.4 Modified ARPNetSteg

The original ARPNetSteg algorithm has a significant weakness. Anyone monitoring the network can see a pattern in each reply packet's sender MAC address field, shown in Table 4.3, because of the 'quad control' bit, which indicates the continuation or the end of transmission. Assuming that the attacker knows about the applied algorithm, he can readily get the covert data by extracting the MAC address bits from the field.

The first three reply packets have '0' in the last quad bit, indicating the continuation of transmission, and the last packet has 'a', informing the receiver that the last

ARPNetSteg	Modified ARPNetSteg
12:90:5f:65:ea:60	e5:76:8d:26:cb:50
6b:7f:ff:42:4d:e0	e4:02:49:3c:e9:2b
45:13:df:81:0f:f0	84:ff:19:c5:ed:da
f0:00:00:00:00:0a	73:05:ee:c9:a9:81

Table 4.3: Sender MAC address in original and modified ARPNetSteg

11 quad bits are empty and there is no more data to send. This pattern is easily noticeable to the naked eye. Our modified algorithm creates a 48-bit key with the shared secret key (also used in the modified MTNS algorithm) and is used to XOR the data to be sent, including the control quad bit. Table 4.3 shows what the MAC address looks like after the modification. The mechanism acts like one-time pad encryption, where the ciphertext is sent as the MAC address of the sender. There are $(281474976710656 - 1)$ possible combinations for the key for the attacker to go through to get the key by brute force. We are ignoring the key "000000000000". The key is randomly generated using the shared key as a seed, which is different for every packet. Now, the pattern given by the quad control bit is dissolved, and the data cannot be retrieved efficiently even if the attacker knows the algorithm without the shared secret key.

The ARPNetSteg algorithm provides an advantage with its high bit transfer, with 44 bits per packet at maximum, significantly higher than most network steganographic algorithms we have studied. However, this algorithm has some disadvantages, especially when the size of covert data (combined shares) is large. The number of ARP request/reply packets will increase significantly and can be flagged as unusual by network monitoring mechanisms for ARP spoofing. Moreover, it must be ensured that the device with the IP address in the unallocated list is not active during transmission, resulting in multiple replies for one ARP request, flagging the transmission as suspicious and losing the channel's covertness. Another issue with this algorithm is that the number of unallocated IPs should be significant for covert data transfer. Otherwise, multiple ARP requests will be made for the same IP, and the replies will contain different MAC addresses, which can be detected easily.

In the scenario where ARPNetSteg is unsuitable for network use, other network steganographic solutions, such as RSTEG, PadSteg, StegTorrent and others [29], can be considered. Exploration of further covert channels with dynamic mechanisms is left for further research.

4.1.5 Further possible algorithm vulnerabilities

Vulnerabilities that might arise that are not addressed by the algorithm have been mentioned below:

1. Random data manipulation

   MTNS relies on generating random bytes of data in the data payload to change the hash value per iteration and attempt to match the packet. If the attacker manages to change this data, it would lead to the receiver getting false positives in marking packets, compromising the integrity of the covert data.

   This problem can be solved by implementing a secure channel using SSL/TSL encryption protocols or an integrity check mechanism using hashing to validate the data payload.

2. Insufficient encryption for the covert data

   Our model does not use conventional encryption schemes to encrypt/decrypt data. The model will compromise confidentiality if the covert channel is not secure or the shared secret key is compromised. However, in our model, each covert data chunk is encrypted and decrypted with a key generated using the secret shared key in modified ARPNetSteg. No data is inserted in the packets using modified MTNS.

3. Single point of failure

   Both the modified network steganography algorithms use the same secret shared key to introduce randomness in key generation and generate permutation array. Thus, if this key is compromised, the model will be insecure, and the attacker can extract all the shares to reconstruct the data. Thus, the shared secret key should be protected since its compromise becomes the single point of failure.

4.2 Environment

The implementation of our proposed model required two virtual machines. Both machines are Ubuntu 20.04, allocated 4GB RAM and 15GB ROM. The host machine has 16GB RAM and Ryzen 5600G. Both the VMs were deployed using Oracle VM VirtualBox. They had two network interfaces enabled, one interface named 'enp0s3', which is used as NAT (Network Address Translation) and does not require any configuration on the host network or guest system. This is required to access the internet to download files and packages. The second interface, 'enp0s9', is used as an internal network, which creates a virtual LAN (Local Area Network) between the two machines. Within this network, the virtual machines communicate with each other in a separate environment and are used to carry out our steganographic algorithm. We have to manually set the IP addresses for this interface on both the sender and receiver machines. In our implementation, we have set the IP for the sender machine to 192.168.100.3 and the receiver machine to 192.168.100.6.

Python was used to code the entire model, which used cryptography and scapy libraries. Cryptography library is used to hash the shares. Scapy is used to craft packets like ARP requests, ARP replies, and TCP/IP packets. Wireshark is used to monitor network traffic and check the contents of the packets.

4.3 Implementation

First, the secret data 'Attack at dawn' is taken as input from the sender side in Fig 4.1. The secret data is converted into shares using the secret sharing scheme in the sender-side algorithm. The scheme produces different shares for the same input whenever used.

```
Secret message: Attack at dawn
Shares:
1-6700c5ead625fb9c843b93d81
2-c1edf7491e326829e2a5602d1
3-4f2f91e1fe1c435b924d15f72
4-936cdcafa9fd9499e55fc394e
5-9fe90b7edbc547c89b05aa7b
...
...
Message sent.
```

Figure 4.1: Sending the secret message

Next, for each share, a hash value is generated. With each share, their corresponding hash value is concatenated. The first 4 hexadecimal digits and the last four digits of the hash are concatenated to every share. For example:

Share: 1-1c34c41ac45
Hash: 3e11ac9f2ef9f2f5d4cbc9dc74efb4ef57e4ac4f88d65de3421e8846ec6deb93
Hashed share: 1-1c34c41ac453e11deb93

With the hashed shares generated, they are split into a ratio where x shares are sent through ARP and y shares are sent through IP, where $x<y<k$. The shares are combined into a single string with a separator string 'fff' to distinguish where one share ends and another begins. Now, they are ready to be sent parallelly through the steganography methods.

```
▸ Frame 2534: 42 bytes on wire (336 bits), 42 bytes captured (336 bits) on interface enp0s8, id 0
▾ Ethernet II, Src: PcsCompu_02:a6:82 (08:00:27:02:a6:82), Dst: PcsCompu_d2:aa:ba (08:00:27:d2:aa:ba)
  ▸ Destination: PcsCompu_d2:aa:ba (08:00:27:d2:aa:ba)
  ▸ Source: PcsCompu_02:a6:82 (08:00:27:02:a6:82)
    Type: ARP (0x0806)
▾ Address Resolution Protocol (reply)
    Hardware type: Ethernet (1)
    Protocol type: IPv4 (0x0800)
    Hardware size: 6
    Protocol size: 4
    Opcode: reply (2)
    Sender MAC address: 96:a3:2c:68:b5:4a (96:a3:2c:68:b5:4a)
    Sender IP address: 192.168.100.119
    Target MAC address: PcsCompu_d2:aa:ba (08:00:27:d2:aa:ba)
    Target IP address: 192.168.100.6

0000  08 00 27 d2 aa ba 08 00  27 02 a6 82 08 06 00 01   ··'····· '······ 
0010  08 00 06 04 00 02 96 a3  2c 68 b5 4a c0 a8 64 77   ········ ,h·J··dw
0020  08 00 27 d2 aa ba c0 a8  64 06                     ··'····· d·
```

Figure 4.2: Contents of an ARP reply packet

Using the revised ARPNetSteg method, a broadcast request is sent continuously from the receiver side until the sender receives it and sends a reply. Fig 4.2 of ARP reply packet content shows that the sender's MAC address and IP address

can be seen. Unallocated IP is the sender's IP address, and encrypted covert data are passed through the MAC address. The MAC address looks similar to an actual MAC address.

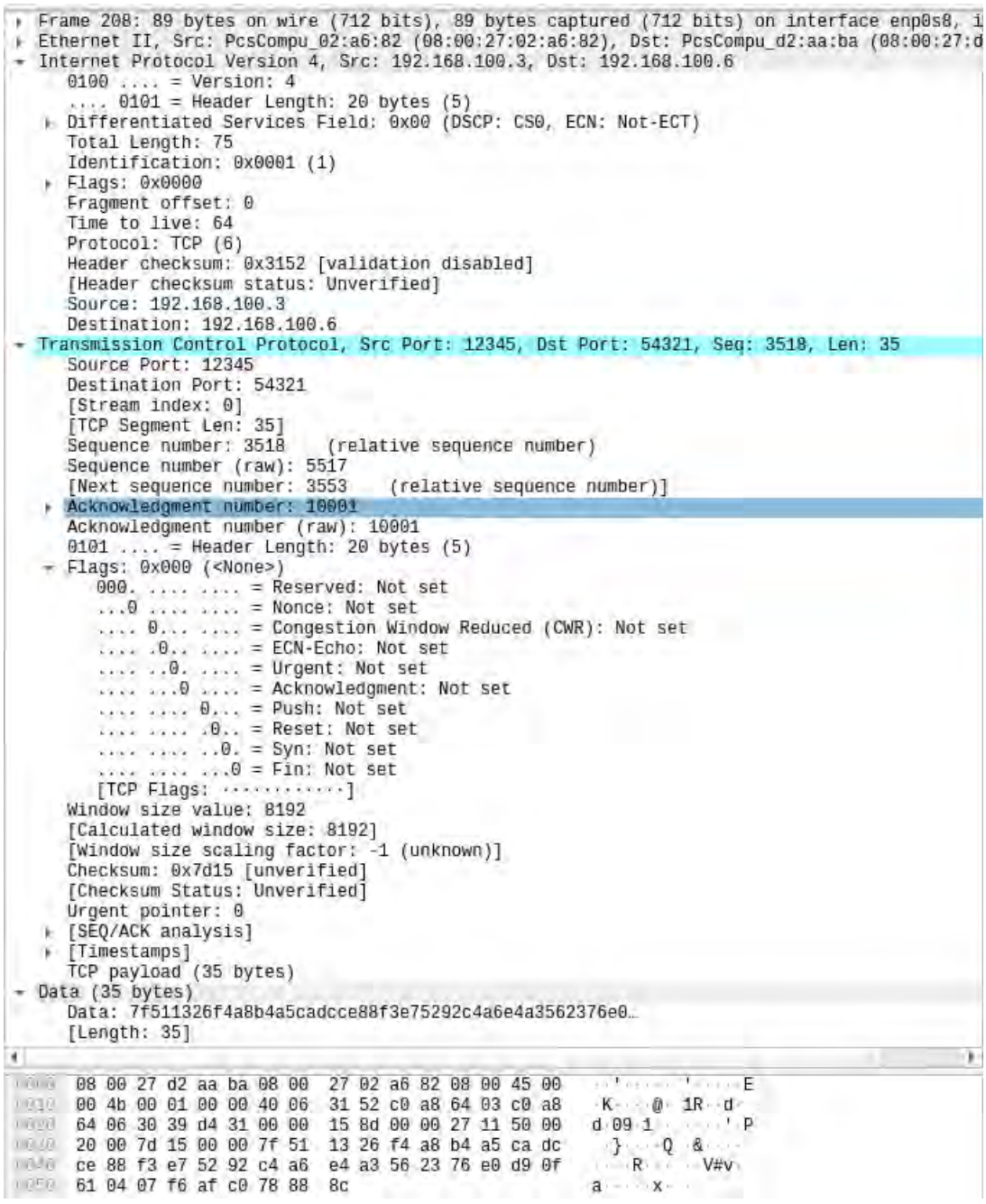

Figure 4.3: Contents of a marked TCP/IP packet

Using the modified MTNS method, we send 4 bits of data per marked packet. We are using 1 bit to mark the packet anonymously. So, 5 bits of data are being used for sending data per packet. On the receiver side, it is checked whether the FIN flag is active. If active, the last packet has been sent, and the covert data transmission is terminated. The contents of an unmarked and marked TCP/IP packet captured using Wireshark are shown in Fig 4.4 and 4.3.

Figure 4.4: Contents of an unmarked TCP/IP packet

In the IP packets, the payload carries random data, where the data does not have any special meaning. A flag is not used in our proposed way of marking an IP packet. Instead, marking is done by matching the marking bit, which enables us to create marked packets that are not distinguishable from the unmarked packets, as seen in Fig 4.3 and 4.4 of the contents of a market and an unmarked packet.

The two groups of combined shares from the two steganography methods are received on the receiver side. Then the shares are separated from the combined shares using the 'fff' separator string. To validate the shares, appended hash bits of each share are separated into shares and their corresponding hashes, matched with the hash generated on the receiver side.

```
Started receiving.....
...
...
Shares:
1-6700c5ead625fb9c843b93d81
4-936cdcafa9fd9499e55fc394e
2-c1edf7491e326829e2a5602d1
3-4f2f91e1fe1c435b924d15f72
5-9fe90b7edbc547c89b05aa7b
Secret message received: Attack at dawn
Time taken: 3.615704298019409
```

Figure 4.5: Message received Successful

After validating the shares, Shamir's secret sharing reconstruction scheme regenerates our secret data, shown in Fig 4.5.

2.070212667	192.168.100.3	192.168.100.6	TCP	94 12345 → 54321 [<None>] Seq=42610 Win=8192 Len=40
2.071404302	192.168.100.3	192.168.100.6	TCP	94 12345 → 54321 [<None>] Seq=42650 Win=8192 Len=40
2.074442865	192.168.100.3	192.168.100.6	TCP	89 12345 → 54321 [<None>] Seq=42690 Win=8192 Len=35
2.075692922	192.168.100.3	192.168.100.6	TCP	80 12345 → 54321 [<None>] Seq=42725 Win=8192 Len=26
2.079213456	192.168.100.3	192.168.100.6	TCP	90 12345 → 54321 [<None>] Seq=42751 Win=8192 Len=36
2.080925144	192.168.100.3	192.168.100.6	TCP	77 12345 → 54321 [<None>] Seq=42787 Win=8192 Len=23
2.083126227	192.168.100.3	192.168.100.6	TCP	89 12345 → 54321 [<None>] Seq=42810 Win=8192 Len=35
2.085653406	192.168.100.3	192.168.100.6	TCP	93 12345 → 54321 [<None>] Seq=42845 Win=8192 Len=39
2.086576539	192.168.100.3	192.168.100.6	TCP	101 12345 → 54321 [<None>] Seq=42884 Win=8192 Len=47
2.087991370	192.168.100.3	192.168.100.6	TCP	97 12345 → 54321 [<None>] Seq=42931 Win=8192 Len=43
2.088813771	192.168.100.3	192.168.100.6	TCP	87 12345 → 54321 [<None>] Seq=42974 Win=8192 Len=33
2.089809405	192.168.100.3	192.168.100.6	TCP	82 12345 → 54321 [<None>] Seq=43007 Win=8192 Len=28
2.090873377	192.168.100.3	192.168.100.6	TCP	74 12345 → 54321 [<None>] Seq=43035 Win=8192 Len=20
2.092771357	192.168.100.3	192.168.100.6	TCP	97 12345 → 54321 [<None>] Seq=43055 Win=8192 Len=43
2.094890925	192.168.100.3	192.168.100.6	TCP	79 12345 → 54321 [<None>] Seq=43098 Win=8192 Len=25
2.096499381	192.168.100.3	192.168.100.6	TCP	79 12345 → 54321 [<None>] Seq=43123 Win=8192 Len=25
2.097983383	192.168.100.3	192.168.100.6	TCP	101 12345 → 54321 [<None>] Seq=43148 Win=8192 Len=47
2.101548302	192.168.100.3	192.168.100.6	TCP	90 12345 → 54321 [<None>] Seq=43195 Win=8192 Len=36
2.103861589	192.168.100.3	192.168.100.6	TCP	96 12345 → 54321 [<None>] Seq=43231 Win=8192 Len=42
2.105105973	192.168.100.3	192.168.100.6	TCP	82 12345 → 54321 [<None>] Seq=43273 Win=8192 Len=28
2.106168301	192.168.100.3	192.168.100.6	TCP	104 12345 → 54321 [<None>] Seq=43301 Win=8192 Len=50
2.107198579	192.168.100.3	192.168.100.6	TCP	85 12345 → 54321 [<None>] Seq=43351 Win=8192 Len=31
2.108193138	192.168.100.3	192.168.100.6	TCP	93 12345 → 54321 [<None>] Seq=43382 Win=8192 Len=39
2.108540112	PcsCompu_02:a6:82	PcsCompu_d2:aa:ba	ARP	42 192.168.100.119 is at 96:a3:2c:68:b5:4a
2.109694046	192.168.100.3	192.168.100.6	TCP	83 12345 → 54321 [<None>] Seq=43421 Win=8192 Len=29
2.111504924	192.168.100.3	192.168.100.6	TCP	101 12345 → 54321 [<None>] Seq=43450 Win=8192 Len=47
2.112641955	192.168.100.3	192.168.100.6	TCP	104 12345 → 54321 [<None>] Seq=43497 Win=8192 Len=50
2.113829711	192.168.100.3	192.168.100.6	TCP	76 12345 → 54321 [<None>] Seq=43547 Win=8192 Len=22
2.118119206	192.168.100.3	192.168.100.6	TCP	100 12345 → 54321 [<None>] Seq=43569 Win=8192 Len=46

Figure 4.6: Capturing packets in wireshark

4.4 Practical Analysis

4.4.1 Modified MTNS Analysis to find Optimal Evaluation Bits

We have analyzed the modified MTNS algorithm to find the average execution time by varying the length of the data bits from 64 to 512 and using 4 and 8 as evaluation bits to find the optimal evaluation bit to use in our model. We calculated the execution time 20 times per case and determined the average execution time.

Our observations indicate that when the evaluation bit is 4, the execution time is significantly lower than when the evaluation bit is 8, as shown in Table 4.4. In the case of varying lengths of covert data, we see that for 4 evaluation bits, the differences in execution time do not vary significantly.

Number of data bits	Average Execution Time in seconds	
	Eval bits = 4	Eval bits = 8
64	0.8286901951	4.262881517
96	1.045309973	5.693948579
128	1.291510653	7.218351531
160	1.613841462	8.106755567
256	2.251874804	13.56775496
384	2.750419021	22.08677053
512	3.695364404	27.83321488

Table 4.4: Average Execution Time based on Evaluation and Data bits

On the contrary, for 8 evaluation bits, the differences in execution time make an impact. Analyzing the data from Table 4.4 and Fig 4.7 shows that the line for 4 evaluation bits is almost flat with a slight ascend. On the other hand, the line for 8 evaluation bits is increasing linearly. Thus, we can conclude from this experiment that the optimal number of evaluation bits is 4.

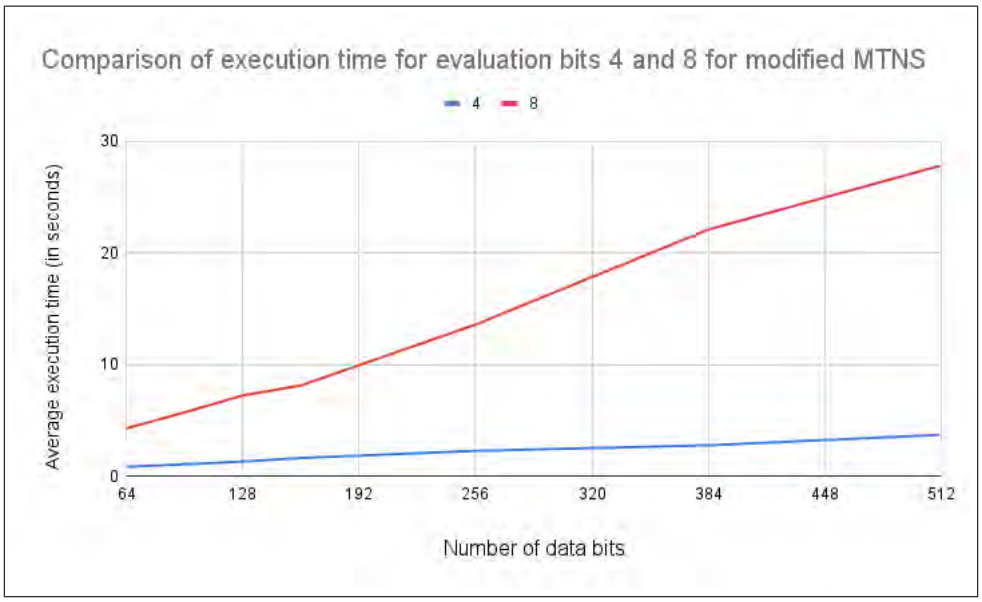

Figure 4.7: Graphical Comparison of Execution Time

4.4.2 Analysis on Model's Execution Time

To analyze how the execution time of the model increases per string length, the length of the secret message is varied from 1 to 28 and kept the following variables constant:

$$\text{Threshold} = 6$$
$$\text{Total shares} = 9$$
$$\text{Shares sent through modified ARPNetSteg} = 2$$
$$\text{Shares sent through modified MTNS} = 4$$

We have shown the results of ten successful attempts in Fig 4.8 to send the 6 shares and the corresponding execution time. The results indicate that the average execution time fluctuates but increases in general as the length of the secret message string is increased. This implies that transmitting a longer message takes longer to send and receive. However, since there are fluctuations, the relationship is not strictly linear.

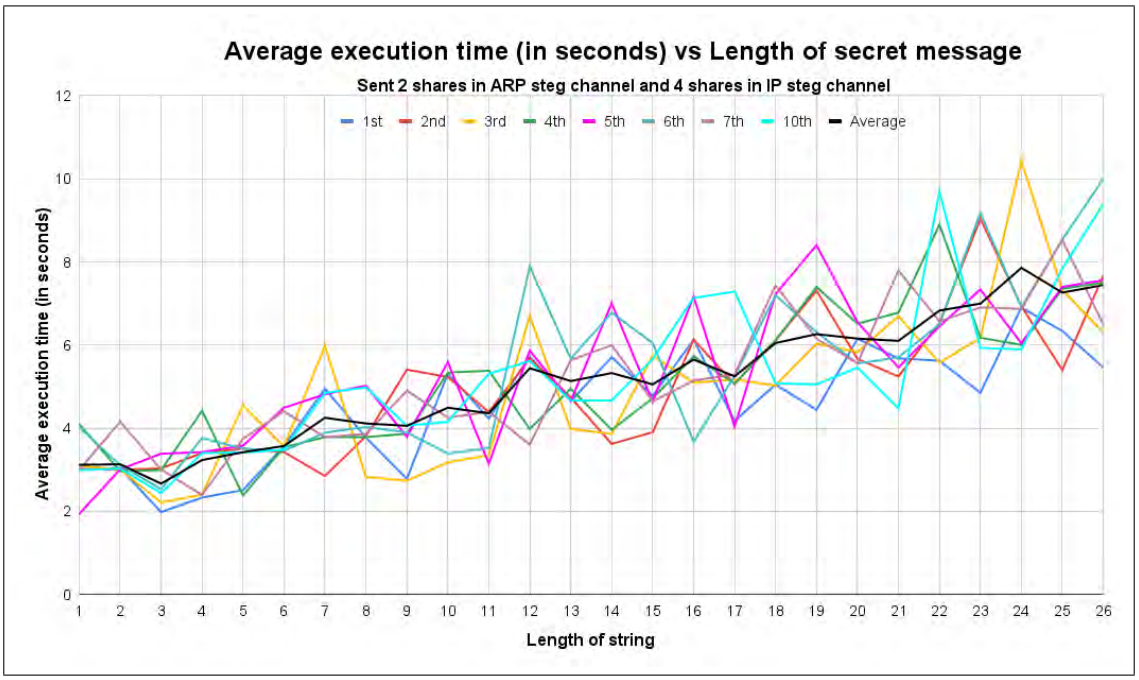

Figure 4.8: Detailed Comparison of Execution Time of the model

The fluctuations may arise due to two reasons:

1. The probabilistic nature of data bit matching and the marking bit. This means the total number of packets to send the same data varies. An analysis of the average number of packets transmitted to send the data in 100 iterations is shown in Fig 4.9

2. The receiver algorithm did not receive an ARP reply packet and had to broadcast the request to receive the packet again. This is a limitation of the environment of our implementation.

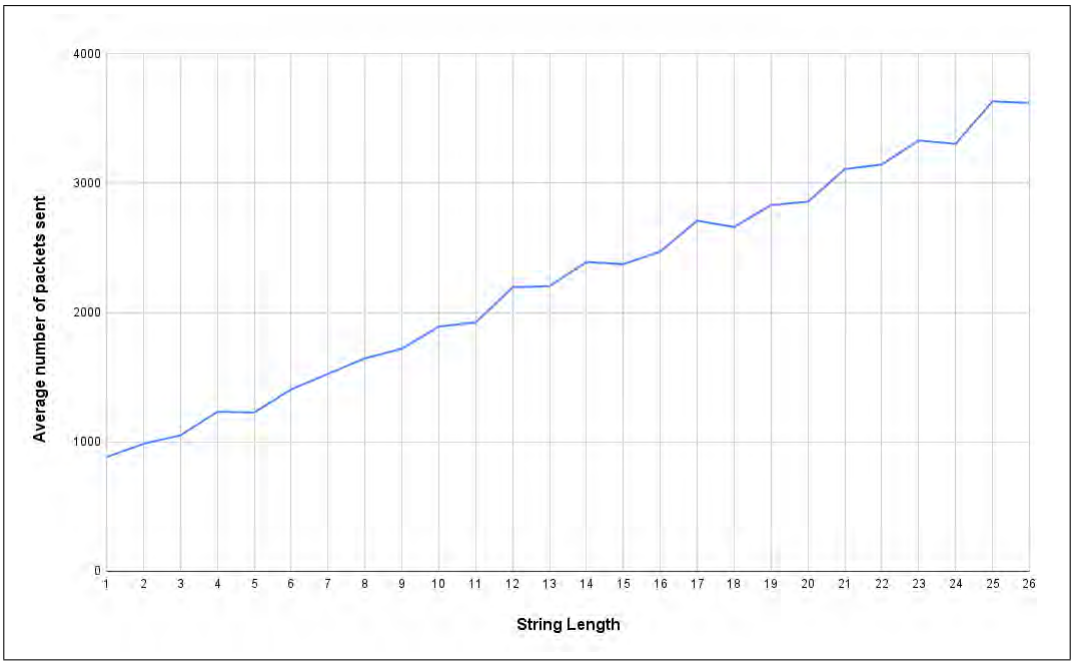

Figure 4.9: Average number of packets sent per message length

The graph in Fig 4.10 represents the time taken to transmit data of various lengths. The x-axis of the graph is the length of the covert data string, and the y-axis of the graph is the time taken in seconds. The values for ARP and IP shares being transferred are changed for each line.

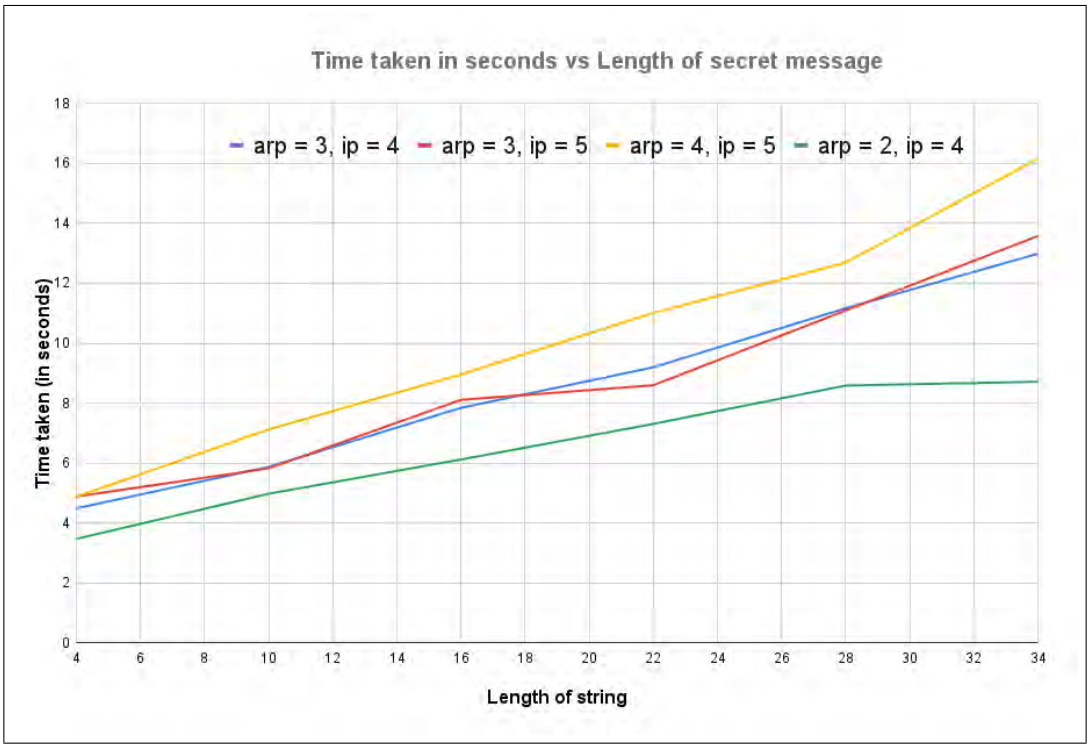

Figure 4.10: Comparison of the distribution of 6 to 9 shares

If we compare the red and blue lines, we can see that even though the number of IP packets is different, the lines are close together, which means a change in the number of IP shares passed has not significantly increased the execution time. Thus, increasing the shares passed through modified MTNS does not significantly affect the time cost.

However, if we compare the red and yellow lines or the green and blue lines where the number of IP shares was kept constant and ARP shares were increased, we can see a notable increase in the time taken to transfer data. This shows us the modified ARPNetSteg algorithm is time-consuming in our test environment and decreases the data transfer rate. Overall, the execution time increases when the number of shares sent through the covert channels increases or the secret message length is increased.

Chapter 5

Conclusion

5.1 Conclusion

This paper focuses on the covert transmission of data. The research problem implicates the need for a secure and robust method of data transmission through the network, where covert data transfer is the key. The main goal of our research is to build such a model that can transfer confidential data through a public network in a hidden manner with the help of network steganography and secret sharing. Our research has produced an improved version of ARPNetSteg where the covert data transmitted in each ARP reply packet is encrypted by generating a key with the shared secret key. We have also worked on a more hidden method to mark the packets instead of using the PSH flag. Practical implementation and analysis of the covert model for secure data transmission, combining Shamir's secret sharing scheme with modified MTNS and ARPNetSteg have been shown. The model is used to send the secret message split into multiple covert channels and can endure data loss in transmission. A detailed security analysis of the algorithm, addressing advantages and drawbacks, and practical analysis to find the optimal number of data bits and average execution times have been provided. This work not only contributes to the field of covert communication but also provides a new, robust approach to implementing hidden data transmission.

5.2 Future research work

Our current work has the potential for further research in the future. We can research more steganographic algorithms to add to our system that provides better performance than ARP. Other algorithms with much better reliability, a higher bit rate, or more indiscernibility would work better for our system. Moreover, the ratio of shares distributed between ARPNetSteg and modified MNTS is currently static. Making that ratio dynamic will bring more reliability to our model. Furthermore, the value of the threshold and the number of shares are static. Introducing those variables to be dynamic based on network traffic and network reliability would make the system more efficient. Another essential part would be to deploy and test the model on a real network and document its usability.

Bibliography

[1] A. Bendovschi, "Cyber-attacks – trends, patterns and security countermeasures," *Procedia Economics and Finance*, vol. 28, pp. 24–31, 2015, 7th INTERNATIONAL CONFERENCE ON FINANCIAL CRIMINOLOGY 2015, 7th ICFC 2015, 13-14 April 2015,Wadham College, Oxford University, United Kingdom, ISSN: 2212-5671. DOI: https://doi.org/10.1016/S2212-5671(15)01077-1. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S2212567115010771.

[2] H. S. Lallie, L. A. Shepherd, J. R. Nurse, *et al.*, "Cyber security in the age of covid-19: A timeline and analysis of cyber-crime and cyber-attacks during the pandemic," *Computers Security*, vol. 105, p. 102 248, 2021, ISSN: 0167-4048. DOI: https://doi.org/10.1016/j.cose.2021.102248. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0167404821000729.

[3] Y. Li and Q. Liu, "A comprehensive review study of cyber-attacks and cyber security; emerging trends and recent developments," *Energy Reports*, vol. 7, pp. 8176–8186, 2021, ISSN: 2352-4847. DOI: https://doi.org/10.1016/j.egyr.2021.08.126. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S2352484721007289.

[4] H. Hui, C. Zhou, S. Xu, and F. Lin, "A novel secure data transmission scheme in industrial internet of things," *China Communications*, vol. 17, no. 1, pp. 73–88, 2020. DOI: 10.23919/JCC.2020.01.006.

[5] S. Zhang, M. Lagutkina, K. O. Akpinar, and M. Akpinar, "Improving performance and data transmission security in vanets," *Computer Communications*, vol. 180, pp. 126–133, 2021, ISSN: 0140-3664. DOI: https://doi.org/10.1016/j.comcom.2021.09.005. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0140366421003340.

[6] F. Chen, "Data transmission security in computer network communication," *Journal of Physics: Conference Series*, vol. 1881, no. 4, p. 042 014, Apr. 2021. DOI: 10.1088/1742-6596/1881/4/042014. [Online]. Available: https://dx.doi.org/10.1088/1742-6596/1881/4/042014.

[7] S. Dhawan and R. Gupta, "Analysis of various data security techniques of steganography: A survey," *Information Security Journal: A Global Perspective*, vol. 30, no. 2, pp. 63–87, 2021. DOI: 10.1080/19393555.2020.1801911. eprint: https://doi.org/10.1080/19393555.2020.1801911. [Online]. Available: https://doi.org/10.1080/19393555.2020.1801911.

[8] A. Beimel, "Secret-sharing schemes: A survey," in *Coding and Cryptology*, Y. M. Chee, Z. Guo, S. Ling, *et al.*, Eds., Berlin, Heidelberg: Springer Berlin Heidelberg, 2011, pp. 11–46, ISBN: 978-3-642-20901-7.

[9] T. Ermakova and B. Fabian, "Secret sharing for health data in multi-provider clouds," in *2013 IEEE 15th Conference on Business Informatics*, 2013, pp. 93–100. DOI: 10.1109/CBI.2013.22.

[10] F. Alsolami and T. E. Boult, "Cloudstash: Using secret-sharing scheme to secure data, not keys, in multi-clouds," in *2014 11th International Conference on Information Technology: New Generations*, 2014, pp. 315–320. DOI: 10.1109/ITNG.2014.119.

[11] J. Li, X. Wang, Z. Huang, L. Wang, and Y. Xiang, "Multi-level multi-secret sharing scheme for decentralized e-voting in cloud computing," *Journal of Parallel and Distributed Computing*, vol. 130, pp. 91–97, 2019, ISSN: 0743-7315. DOI: https://doi.org/10.1016/j.jpdc.2019.04.003. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S074373151930262X.

[12] B. Wu, J. Wu, E. B. Fernandez, and S. Magliveras, "Secure and efficient key management in mobile ad hoc networks," in *Proceedings of the 19th IEEE International Parallel and Distributed Processing Symposium (IPDPS'05) - Workshop 17 - Volume 18*, ser. IPDPS '05, USA: IEEE Computer Society, 2005, p. 288.1, ISBN: 0769523129. DOI: 10.1109/IPDPS.2005.393. [Online]. Available: https://doi.org/10.1109/IPDPS.2005.393.

[13] A. Shamir, "How to share a secret," *Communications of the ACM*, vol. 22, pp. 612–613, Nov. 1979. DOI: 10.1145/359168.359176.

[14] H. Krawczyk, "Secret sharing made short," in *Advances in Cryptology — CRYPTO' 93*, D. R. Stinson, Ed., Berlin, Heidelberg: Springer Berlin Heidelberg, 1994, pp. 136–146, ISBN: 978-3-540-48329-8.

[15] L. Harn, Z. Xia, C. Hsu, and Y. Liu, "Secret sharing with secure secret reconstruction," *Information Sciences*, vol. 519, pp. 1–8, 2020, ISSN: 0020-0255. DOI: https://doi.org/10.1016/j.ins.2020.01.038. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0020025520300402.

[16] C.-W. Lee, "A secret transmission method via numeric data with a blind authentication capability," en, *Multimed. Tools Appl.*, vol. 77, no. 13, pp. 16 623–16 659, Jul. 2018.

[17] G. J. Simmons, "The prisoners' problem and the subliminal channel," in *Advances in Cryptology: Proceedings of Crypto 83*, D. Chaum, Ed. Boston, MA: Springer US, 1984, pp. 51–67, ISBN: 978-1-4684-4730-9. DOI: 10.1007/978-1-4684-4730-9_5. [Online]. Available: https://doi.org/10.1007/978-1-4684-4730-9_5.

[18] W. Frączek and K. Szczypiorski, "Perfect undetectability of network steganography," *Security and Communication Networks*, vol. 9, no. 15, pp. 2998–3010, 2016.

[19] M. Drzymała, K. Szczypiorski, and M. Ł. Urbański, "Network steganography in the dns protocol," *International Journal of Electronics and Telecommunications*, vol. 62, 2016.

[20] P. Bedi and A. Dua, "Arpnetsteg: Network steganography using address resolution protocol," *International Journal of Electronics and Telecommunications*, vol. 66, 2020.

[21] P. Bedi and A. Dua, "Network steganography using the overflow field of timestamp option in an ipv4 packet," *Procedia Computer Science*, vol. 171, pp. 1810–1818, 2020.

[22] T. Soni, "Moving target network steganography," Ph.D. dissertation, Rowan University, 2020.

[23] K. Haseeb, N. Islam, A. Almogren, I. Ud Din, H. N. Almajed, and N. Guizani, "Secret sharing-based energy-aware and multi-hop routing protocol for iot based wsns," *IEEE Access*, vol. 7, pp. 79 980–79 988, 2019. DOI: 10.1109/ACCESS.2019.2922971.

[24] B. Yuan and P. Lutz, "A covert channel in packet switching data networks," Jan. 2005.

[25] K. Cabaj, P. Żórawski, P. Nowakowski, M. Purski, and W. Mazurczyk, "Efficient distributed network covert channels for Internet of things environments†," *Journal of Cybersecurity*, vol. 6, no. 1, Dec. 2020, tyaa018, ISSN: 2057-2085. DOI: 10.1093/cybsec/tyaa018. eprint: https://academic.oup.com/cybersecurity/article-pdf/6/1/tyaa018/34893203/tyaa018.pdf. [Online]. Available: https://doi.org/10.1093/cybsec/tyaa018.

[26] R. Rios, J. A. Onieva, and J. Lopez, "Covert communications through network configuration messages," *Computers Security*, vol. 39, pp. 34–46, 2013, 27th IFIP International Information Security Conference, ISSN: 0167-4048. DOI: https://doi.org/10.1016/j.cose.2013.03.004. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0167404813000497.

[27] X. Luo, P. Zhang, M. Zhang, H. Li, and Q. Cheng, "A novel covert communication method based on bitcoin transaction," *IEEE Transactions on Industrial Informatics*, vol. 18, no. 4, pp. 2830–2839, 2022. DOI: 10.1109/TII.2021.3100480.

[28] S. Thomsen, "Cryptographic hash functions," English, Ph.D. dissertation, Feb. 2009.

[29] N. Singh, J. Bhardwaj, and G. Raghav, "Network steganography and its techniques: A survey," *International Journal of Computer Applications*, vol. 174, pp. 8–14, Sep. 2017. DOI: 10.5120/ijca2017915319.