0.1 INTRODUCTION

Data exfiltration, data extrusion or data exportation is the unauthorized, illegal and unapproved removal of data from a computer or a network. We have two types of data exfiltration, it could be manual where the attacker has physical access to the computer or it could be automated by a malicious program running over the network. As it requires the transfer of data inside and outside a company's network, it frequently mimics typical network traffic, allowing valuable data loss events to go unnoticed until data exfiltration has already been completed. And when your company's most precious information is in the hands of hackers, the damage is limitless.

There are more data exfiltration ways than there are roads to Rome. but still, the most sufficient way to exfiltrate data is by using covert channels.

Lampson first described covert channels [**?**] as "Channels not intended for information transfer at all". Since then the definition of covert channels has been developing. In 1993 Virgil Gligor defined them [**?**] as "A communication channel that allows a process to transfer information in a manner that violates the systems security policy". While the most recent definition was given in 2010 by Eric Couture [**?**], he described a covert channel as "A mechanism for sending information without the knowledge of the network administrator or other users". To sum all this up we can describe covert channels as "A Communication technique that is used to transfer information in a secretive and unauthorized manner. Hence, it's simply an extra way for information to leave a network".

Data Exfiltration using a covert channel can be expressed as a bag with a secret section that a spy is using to slip a weapon past security guards into or out of a guarded building. An attacker can use covert channels to transmit sensitive documents unobserved. In our case, rather than bypassing security guards we need to bypass network security standards to implement a covert channel. And just as a spy can use that same secret section to sneak a weapon from security guards when entering a guarded building, an attacker can use a covert channel to conceal a cyber weapon. For example, download a malware from an external server into the victim machine within an organization's private network.

0.2 Background

Network security is one of the most crucial and frequently neglected aspect in a computer network. A network security breach could impact all the actors in that network, from consumers, companies, and as well as governments. For consumers, networks could include their online identities leaving them victims of attackers who are interested in stealing information such as, credit card numbers, passwords, and other data that can cause a tremendous amount of damage if it was in the wrong hands. For companies and corporations, the damage could be much worse from losing sensitive data to another competitor. Furthermore, a weak network security could influence governments, as attackers could gain access to confidential military documents as well as to the financial records which could be used against that government itself.
Network attacks are expected and are more common than you think. And it is obvious now that a new course of action must be taken to prevent them.

Most of the cybersecurity research that has been, or is being done direct its attention on preventing attackers from breaching into the network. And the most commonly practiced method for that is the Intrusion Detection System (IDS) [**?**]. The goal of IDS is to give attackers several security layers to beat before

breaching the network. Of course, this defensive tool is quite necessary but it is even more essential to look at other lines of defence as well. Since these protective methods cannot examine all network traffic without costly hardware or high overhead that will result in network slowdowns. Plus, in reality, these defensive tools can never absolutely protect the network as the most eager and enthusiastic hackers will not stop trying at the sign of a powerful defence especially if they are trying to breach a military or governmental network that will give them a great advantage in the times of war and political activities. Hence, eventually, they will find a way around these tools to breach the network. Based on the above, it is not only important to look into the incoming traffic for abnormalities that could be an evidence of network breach but also to examine the outgoing traffic for the effect of a network breach. Promptly, in order to earn from a network breach, a hacker must be able to find a way to transfer data from the attacked machine to his server and this is known as Data Exfiltration. Some tools do exist to block sensitive data from leaving the network. These tools are referred to as Data Leak Prevention (DLP) tools. However, these tools are targeted and are an expression or keyword−based algorithms, which an intruder can easily overcome.

0.3 Security Model

Public Key Encryption (PKE) and Public Key Message Authentication (PKMA) schemes are two of the most significant and well studied cryptographic primitives. Traditionally, both notions are defined as non-interactive (i.e., used for single messages). Both the sender and receiver need not keep any state, such non-interactive "one−and−done" philosophy is also crucial in many applications. Joined with the fact that we now have many efficient candidates for both signature and encryption schemes, it might appear that there is limited benefit in prolonging the syntax of signature/encryption schemes to allow for possibly interactive realizations.
The biggest disadvantage of non-interactive signatures is the fact that there is no interaction between the sender and the receiver. Hence, this works for a single message only which is very limited. To avoid this, a new scheme was proposed by [**?**] that assumes the presence of interaction between the sender and the receiver which is logical especially in the case where there is a conversation. This new scheme is "Interactive Encryption and Message Authentication" [**?**]

This interactive scheme offers some advanced security properties which are impossible to be achieved with the non-interactive schemes, Deniability and Forward Security.

- ***Deniability:*** A Public Key Message Authentication protocol is deniable if the sender S can authenticate a message m to the receiver R in such a way that R cannot use the transcript of their conversation as evidence to later convince third parties about the fact that S took part in the protocol and authenticated m.

- ***Forward Security:*** Forward security guarantees that any leak of secret information at some time t should not affect the security of protocol runs that occurred in the past, i.e., at any time $t' < t$. Forward security is a desirable property that is not known to be achieved by standard non-interactive public key encryption.

We consider a problem that consists of an attacker "Alice" who already has breached a network and gained access to a machine controlled by a victim "Bob". Alice intends to exfiltrate data from Bob's machine to her machine via multiple n channels. We assume that there is a watcher "Eve " who can read all the n channels and block b out of n channels, where $b \leq (n-1)$.

To define our interactive security Model, we start by describing a ***Measured Matrix*** consisting of two matrices.

- ***Round Complexity:*** Which is the number of iterations that we should do to send p packets over n channels.
- ***Communication Complexity:*** Which is how many bits we transmit with respect to the size of the file that is intended to be transferred.

0.3.1 Security Properties

We define the security properties, ***completeness***, ***unforgeability*** and ***authenticity***.

- ***Completeness:*** Is the property which guarantees that Alice will receive the complete file that she is trying to exfiltrate from Bob's machine.
- ***Unforgeability:*** Is the property which ensures that it's not possible for the adversary to forge a packet and send it to Alice as if it were coming from Bob. In other words, ***Unforgeability*** guarantees that Bob will not be fooled by the adversary.
- ***Authenticity:*** Is the property which guarantees that Alice will receive messages from Bob only.

The watcher Eve changes the content of the communication channels with various objectives that include:

1. Deleting a packet so that Bob never receives it. Thus, preventing communication between Alice and Bob, censorship;
2. Inserting a new packet invented by Eve and send it to Alice wishing she will consider it authentic, forgeability;
3. Changing the order of packets sent through the channels.

0.3.2 Protocol Proposal

The proposed protocol is an innovative exfiltration technique that uses n covert channels to exfiltrate files from a controlled machine to a controlled server. The aim of this protocol is to guarantee that the attacker will receive complete and unforgeable data that he is trying to exfiltrate even in the presence of a watcher who is trying to stop the flow of data.

0.4 Protocols

We define two protocols that calculate the number of rounds and the time it takes Alice to exfiltrate a file from Bob's machine that she illegally controls, using n covert channels while we assume that Eve can read all the n channels but can block b out of these n channels where $b \leq (n-1)$.

From here on we will refer to Alice as the Server, Bob as Client and Eve as watcher.

Round: Each iteration over the available n channels is defined as a round.

Through out this protocol, to guarantee data authentication and integrity we used HMAC, it is a particular type of "Message Authentication Code" that uses a "Cryptographic Hash Function" with a "Secrete Key". With our protocol, to calculate the HMAC for the entire file we used the cryptographic function SHA-512 with a key K while to calculate the HMAC for the chunks of the file we used the cryptographic function SHA-1 with a key K'.

Assumption: Since the client and the server are controlled by the same person we assume the hashing keys and the cryptographic functions that are used to calculate the $MAC(key, Message)$ can be safely applied to both the client and the server.

Summary of Protocol #1:
We compute the HMAC $T = MAC(K, M)$ of the entire file. After that, we divide the file into chunks and compute the HMAC $t_i = MAC'(K', m_i)$ for each chunk of the file. First, we create a packet of type #1 containing the calculated HMAC $T' = MAC(K, M')$ where M' consists of all the attributes inside the packet, the form of type#1 packet is $(T', andOtherAttributes)$, it will broadcast over all the available n channels. Then, we construct packets of type #2 that are formed of $(ti, andOtherAttributes)$ and send each packet in parallel over the n available channels. Repeat until all packets are sent. Meanwhile, at the beginning, the server will try to verify one of the type #1 packets, by computing the HMAC of the extracted attributes of each packet and then compare the calculated HMAC with the received HMAC T'. After that, for each type #2 packet received, it will compute $t_m = MAC'(K', m_i)$ and compare it with t_i. If they are equal, the server will accept the packet otherwise it will drop it. And when it receives the entire packets, it will compute $T_M = MAC(K, M_{received})$ of the entire file and compare it with T. If they are equal it will accept the entire file otherwise it will drop it.

Summary of Protocol #2:
We compute the HMAC $T = MAC(K, M)$ of the entire file. After that, we divide the file into chunks and compute the HMAC $t_i = MAC'(K', m_i)$ for each chunk of the file. First, we create a packet of type #1 containing the calculated HMAC $T' = MAC(K, M')$ where M' consists of all the attributes inside the packet, the form of type#1 packet is $(T', andOtherAttributes)$, it will broadcast over all the available n channels. Then, for each chunk of the file, we construct a packet of type #2 that contains $(ti, andOther$ $Attributes)$. The client picks the first n packets and send a different packet over each of the n channels, then the client waits for an ACK from the server that contains the packets which were delivered correctly, the server will deduct the correctly received packets and will retransmit the remaining packets, this will be repeated until all the n packets are received correctly and then the client will move to the next n packets and so on. Meanwhile, the server at the beginning will try to verify one of the type #1 packets, by computing the HMAC of the attributes inside each packet and then compare the calculated HMAC with the received HMAC T'. After that for each packet of type #2 received the server will compute the HMAC $t_m = MAC'(K', m_i)$ and compare it with its corresponding t_i. If they are equal, the client will accept the packet and add it to the list of correctly received packets otherwise the packet will be dropped, when the server is done from verifying the n packets it will broadcast an ACK over all the n channels to the client containing the list of sequence numbers of the packets that were correctly received asking it to resend the rest of the packets and repeat until it correctly receives the entire n packets. So, when the server receives the complete file it will compute $T_M = MAC(K, M_{received})$ of the entire file and compare it with T that was sent in a packet of type #1. If they are equal it will accept the file otherwise it will drop it.

For Both Protocols: T protects the file as a whole, whereas each t_i protects a chunk of the file and tells us which chunk was corrupted. Tagging the index along with the chunk takes care of an adversary permuting the chunks. This protocol will work even by assuming that the adversary can corrupt the channels adaptively. i.e, at each run, the corrupted channels are chosen before Bob sends stuff. In other words, even if the watcher decides to corrupt always the same channels the protocol will converge since, after each run, some packets are duplicated or sent over a different channel.

0.4.1 Protocol #1 :

Assumes that the adversary can block an unknown number of channels b out of n channels where $b < n$. s is the packet size used.

Suppose we have a file of size f that the server needs to exfiltrate from the client's machine. we divide this file into p packets where

$$p = \frac{f}{s}$$

Protocol #1 in general, will calculate the HMAC of the entire file and then form a packet containing:

- The HMAC of the complete packet calculated from the attributes::
    1. Packet type.
    2. Number of packets to be sent.
    3. Length of the file name.
    4. The file name that the client will send.
    5. HMAC of the entire file.
- Packet type.
- The file name that the client is going to send.
- The length of the file name.
- The number of packets to send.
- The HMAC of the entire file.

This packet will broadcast over the all n channels at the beginning of the exfiltration process to notify the server that a file will be sent now. Meanwhile, the server will start receiving these packets. For the first packet received it will, extract the attributes, calculate the HMAC of the extracted attributes and then compare it with the received HMAC of the complete packet if they verify it will create a file, save other attributes for later comparisons and drop all other packets of the same type. Otherwise, it will drop the packet and move on to the next one. Then the client divides the complete file into chunks. For each chunk, we form a packet p_i containing:

- The HMAC of the complete packet calculated from the attributes:
    1. Packet type
    2. Sequence number.
    3. Chunk of data from the file.
- Packet type.
- Sequence Number.
- Chunk of data from the file.

We will broadcast each p_i packet over all the n channels in parallel. E.g, packet 1 will be sent over n channels simultaneously, then packet 2 will be sent also over all the available channel and so on, until we send all the p packets. Meanwhile, the server is receiving these packets, it extracts the data inside them, then calculate the HMAC of the attributes:

- Packet type.
- Sequence Number.
- Chunk of data from the file.

The server will compare the calculated HMAC with the HMAC that was extracted from the packet, if they are equal it will drop all other packets that have the same sequence number otherwise it will drop it and then check the next packet and so on until it will have the complete file. After having the complete file it will calculate the HMAC of the entire file and compare it with the HMAC that was saved from the packet of type #1.

We define two types of packets in protocol #1 that will be explained later in this section. The first step is done when the client broadcasts a packet of type #1 (Table: 1)

Complete HMAC	Op Code = 1	Number of Packets	Length of the File Name	File Name	HMAC of the file

Table 1: Type #1 packet of protocol #1.

We call this packet the "File Transfer Request", it is broadcast over all the n channels at the beginning. Its job is to prepare the server to receive a new file. When the server receives this packet it will:

- Extract the attributes:
  1. OP Code (Type of the packet).
  2. Number of Packets.
  3. Length of the File Name.
  4. File Name.
  5. Hash of the File.
- Calculate the HMAC of the extracted attributes.
- Compare the calculated HMAC with the complete HMAC that is received within the packet.
- If they are equal it will:
  1. Create a file with the name "File Name".
  2. Save the complete HMAC of the file.
  3. Save the number of packets of the entire file.
  4. Drop all other packets of the same type.
- If they are not equal it will move to the next packet.

This packet is broadcast only one time at the beginning of the transmission of each file and the server will accept the first packet where the HMAC verifies and it will drop the rest.

Then we have packet of type 2 (Table: 2).

We call this packet the "Data packet", each data packet is also broadcast over all the n channels. Its job is to send the data (*chunks*) of the file. When the server receives this packet it will:

- Calculate the HMAC of the extracted attributes.
  1. OP Code (Type of the packet).
  2. Sequence Number.
  3. Chunk of Data from the File.
- Compare the calculated HMAC with the received HMAC.
- If the HMACs are equal it will accept the packet and drop all other packets with the same sequence number.
- If the HMACs are not equal it will drop the packet and move to the next one.

Note: The final "Data packet" will be padded with (char = 0) by the client in case it is not equal to the defined packet size, and then it will be stripped by the server when it is received.
The server will keep reading these "Data Packets" until the total number of packets accepted equals the number of packets attribute that was received in the "File Transfer Request" packet. Then it will calculate the HMAC of the entire file received and compare it with the HMAC that was received in the "File Transfer Request" packet, if they are equal it will accept the file otherwise, it will drop it. After that, it will get ready to receive the next file.

HMAC	Op Code = 2	sequence no.	Chunk of Data from the File

Table 2: Type #2 packet of protocol #1.

This protocol guarantees the delivery of the complete file even in the presence of a watcher who is trying to modify the packets between the client and the server. Although, this protocol does not use acknowledgment packets from the server to specify which packets were correctly delivered or no. It has high costs in terms of bandwidth and number of rounds as each packet is being broadcast over every channel each time.

0.4.2 Protocol #2 :

Suppose we have n channels out of which the watcher can corrupt at most an unknown number of channels b where $b \leq (n-1)$. Further, Suppose we have a file of size f that the server needs to exfiltrate from the client's machine. we divide this file into p packets where

$$p = \frac{f}{s}$$

Protocol #2 in general, will calculate the HMAC of the entire file and then form a packet containing:

- The HMAC of the complete packet calculated from the attributes:
  1. Packet type.
  2. Number of packets.
  3. Length of the file name.
  4. The file name that the client will send.
  5. HMAC of the entire file.
- Packet type
- The file name that the client is going to send.
- The length of the file name.
- The number of packets to send.
- The HMAC of the complete file.

This packet will broadcast over the all n channels at the beginning of the exfiltration process to notify the server that a file will be sent now. Meanwhile, the server will start receiving these packets. For the first packet received it will, extract the attributes, calculate the HMAC of the extracted attributes and then compare it with the received HMAC. If they verify it will create a file, save other attributes for later comparisons and drop all other packets of the same type. Otherwise, it will drop the packet and move on to the next one. After that, the client will divide the complete file into chunks and for each chunk, we form a packet p_i containing:

- The HMAC of each chunk of the file calculated from the attributes:
  1. Packet type
  2. Sequence number.
  3. chunk of data from the file.
- Packet type
- Sequence Number.
- Chunk of data from the file.

Protocol #2 will choose the first n packets and then send each packet p_i over a different channel. E.g, over channel #1, packet #1 will be sent, over channel #2, packet #2 will be sent and so on. The packets will be sent in parallel (At the same time). Meanwhile, the server is receiving the packets, it extracts the data inside each packet, then calculate the HMAC of the attributes:

- Packet type.
- Sequence Number.
- Data of the chunk.

Then it will compare the calculated HMAC with the HMAC that was received within the packet. If the HMACs verify it will accept the packet and add it to the list of accepted packets. Otherwise, it will drop it. After inspecting the n packets the server will use an ACK packet containing:

- The HMAC of the complete packet calculated from the attributes::
    1. Packet type
    2. Sequence numbers of the accepted packets.
- Packet type.
- Sequence Numbers of the accepted packets.

This Ack packet will be broadcast over all the n channels to inform the client about the accepted packets then the client will retransmit the packets that were not correctly delivered. This will be repeated until all the n packets are successfully received by the server and then the client will start sending the next n packets and the same process will be repeated till all the p packets are correctly received by the server. When all the p packets are received, the server will calculate HMAC of the entire file and then compare it with the HMAC that was received in the first broadcast packet, if they verify, it will accept the file. Otherwise, it will drop it.

We define three types of packets in protocol #2 that will be explained later in this section. The first step is done when the client broadcasts a packet of type 1 (Table: 3).

As with protocol #1, we call this packet the "File Transfer Request", it will broadcast over all the n channels in the beginning. Its job is to prepare the server to receive a new file. When the server receives this packet it will:

- Extract the attributes:
    1. OP Code (Type of packet).
    2. Number of Packets.
    3. Length of the File Name.
    4. File Name.
    5. HMAC of the Data.
- Calculate the HMAC of the extracted attributes.
- Compare the calculated HMAC with the complete HMAC that is received within the packet.
- If they are equal it will:
    1. Create a file with the name "File Name".
    2. Save the complete HMAC of the file.
    3. Save the number of packets of the entire file.
    4. Drop all other packets of the same type.
    5. Prepare to receive the chunks of the file.
- If they are not equal it will move to the next packet.

This packet is broadcast only one time at the beginning of the transmission of each file and the server will accept the first packet of these where the hashes verify and then it will drop the rest.

Then we have packet of type 2 (Table: 4).

We call this packet the "Data packet", each data packet is sent over one available channel. Its job is to send the data of the file. When the server receives this packet it will:

Complete HMAC	Op Code = 1	Number of Packets	Length of the File Name	File Name	HMAC of the entire file

Table 3: Type #1 packet of protocol #2.

Complete HMAC	Op Code = 2	sequence no.	Chunk of data from the file.

Table 4: Type #2 packet of protocol #2.

- Calculate the HMAC of the attributes:
  1. Packet type.
  2. Sequence Number.
  3. Chunk of data from the file.
- Compare the calculated HMAC with the received HMAC.
- If the HMACs are equal it will accept the packet and add the packet sequence number to a list of accepted packets.
- If the HMACs are not equal it will drop the packet.

Note: The final "Data packet" will be padded with (char = 0) by the client in case it is not equal to the defined packet size, and then it will be stripped by the server when it is received.
The server will keep reading these "Data Packets" until the total number of packets successfully received equals the number of packets received in the "File Transfer Request" packet. Then it will calculate the HMAC of the entire file received and compare it with the HMAC that was received in the "File Transfer Request" packet, if they are equal it will accept the file otherwise, it will drop it. Then it will get ready to receive the next file.

Then we have packet of type 3 (Table: 5).

We call this packet the "Data Packet Acknowledgment", This packet type is broadcast to the client from the server after inspecting the n packets that were received. Its job is to notify the client which packets were correctly received. When the client receives this packet type it will:

- Extract the attributes:
  1. Op Code (Packet type).
  2. sequence numbers.
- Calculate the HMAC of the extracted attributes.
- Compare the calculated HMAC with the received HMAC.
- If they are equal it will:
  1. Extract the sequence numbers of the received packets.
  2. Deduct these sequence numbers from the pool of the available packets to be sent.
  3. Drop the other packets of the same type.
  4. Send the remaining packets over the n channels again.
- If they are not equal it will:

HMAC	Op Code = 3	sequence no. of the received packets

Table 5: Type #3 packet of protocol #2.

1. drop the packet.
2. start analyzing the next packet.

This ACK packet is broadcast over all the channels, we can afford that since the size of these packets is relatively small.

This protocol guarantees the delivery of the file even in the presence of a watcher who is trying to modify the packets between the client and the server. It has low costs in terms of bandwidth and number of rounds in comparison with protocol #1 as each packet is sent only one time over a channel and will only be retransmitted in case of a failed delivery. Although this protocol uses ACKs packets. These packets are relatively small in comparison of the data packets so in a way we conserved the low bandwidth.

0.5 Mathematical Logic

To understand better how the protocol is working, we derived two equations that will explain the ***security models***:

1. First equation will explain the ***round complexity*** of the best, average and the worst case scenarios.
2. Second equation will explain the ***communication complexity***.

Round Complexity: Is defined as the number of rounds it takes the protocol to send p packets over n channels.
Let n = total number of channels
Let b = number of bad/corrupted channels
Let p = number of packets to be sent
Let q = max number of rounds
Consider $p = n$ for simplicity
Assume always $b < n$
General form of b : $b = c \times n$ Where c is the fraction of corrupted channels out of n.

Examples of the form of b:

$$b = \frac{2}{3}n, \qquad b = \frac{3}{4}n, \qquad b = \frac{1}{2}n$$

As an example, assume the total number of channels $n = 10$, the total number of packets to send $p = 10$, the number of corrupted channels is $b = 5$. So, $b = \frac{1}{2}n$.

In the best case scenario:

- In round 1, we expect $\frac{1}{2}$ of the total packets to be delivered.
- In round 2, we expect the other $\frac{1}{2}$ of the packets to be correctly delivered.

According to that, we can derive the equation of the best case scenario:
#packets correctly delivered after q rounds =

$$\frac{p}{n-q} \tag{1}$$

In the worst case scenario:

- In round 1, we expect $\frac{1}{2}$ of the total packets to be delivered.
- In round 2, we expect $\frac{1}{2}$ of the remaining packets to be correctly delivered.
- In round 3, we expect $\frac{1}{2}$ of the remaining packets to be correctly delivered.

Based on this, we deduce the #packets that are correctly received in each round to be:

- ***Round*** #1: $(n-nc)$ packets, which translates to half of the packets are correctly delivered.

- ***Round*** #2: $(nc - nc^2)$ packets, which translates to half of the remaining packets will be correctly delivered.
- ***Round*** #3: $(nc^2 - nc^3)$ packets, which translates to half of the remaining packets will be correctly delivered.
- ***Round*** #q: $(nc^{(q-1)} - nc^q)$ packets, as this is the final round, all the remaining packets will be delivered in this round.

According to that, we can derive the equation:
#packets correctly delivered after q rounds =

$$\sum_{i=1}^{q}[(1-c) \times c^{(i-1)} \times n] \tag{2}$$

Now, in the case where $p > n$. Then, the general form of p will be: $p = m \times n + r$
Where m is the coefficient and r is the remainder.
Examples of the form of p:

$$p = 4n + 2, \qquad p = 3n + 5,$$
$$p = 17n + 2$$

In this case, the general form of the equation will become:
#packets correctly delivered after q rounds =

$$m[\sum_{i=1}^{q}[(1-c)\times c^{(i-1)}\times n]]+\sum_{i=\frac{n}{r}}^{q}[(1-c)\times c^{(i-1)}\times n] \tag{3}$$

In the case where $r = 0$. Then the equation will become:
#packets correctly delivered after q rounds =

$$m[\sum_{i=1}^{q}[(1-c) \times c^{(i-1)} \times n]] \tag{4}$$

Note: The above equation assumes that the watcher will always corrupt the maximum number of packets. In other words, this equation calculates the #packets of packets correctly delivered over each round in the worst case scenario.

The average case scenario:

$$average = \frac{best + worst}{2} \tag{5}$$

Communication Complexity: Is defined as the total number of bytes we're sending in order to deliver the complete file with respect to the number of bytes of the file.
Let n = The total number of channels.
Let s = The size of the packet.
Let r = The total number of rounds.
let f = The file size.

The Communication complexity equation is:

$$cc = n \times s \times r \tag{6}$$

Also, we define the efficiency of the protocol which is:

$$\epsilon = \frac{f}{cc} \times 100 \tag{7}$$

0.6 Experiments

As mentioned before, the provided equation calculates the maximum number of rounds needed to receive a file. To mimic this situation in the experiments, we randomly changed the position of the corrupted channels after every round.

To test the efficiency of these protocols, we implemented them simulating a real word scenario. Note that we implemented the protocols using normal TCP channels since it doesn't make any difference if they were covert channels or normal channels. Through out the experiments we consider our computer running ($macOS10.13.4$) positioned in Rome as the victim having these characteristics:

- CPU: 4.
- Processor: Intel Core i5.
- Clock Speed: 2,5 GHz.
- Memory: 8GB.

While the attacker server is a rented machine running ($ubuntu-xenial-16.04$) from amazon located in Sydney "Australia" with the characteristics below:

- vCPU: 1.
- Processor: Intel Xeon.
- Clock Speed: 2,5 GHz.
- Memory: 1GB.

To check how the number of corrupted channels with respect to the number of total available channels affects the efficiency and the number of rounds it takes the protocol to send a file. We conducted the following experiments with a file of size $6MB$ that will be divided into 93 packets:

We repeated the same experiments with a larger file $40MB$ that is divided into 230 packets in total.
The graphs below will show how the total number of rounds required to send the file change when increasing the number of corrupted channels using protocol #2 and the equation (Fig. #23). In addition in (Fig. #24) we compared them with protocol #1. By using a total 10 channels.

The graphs below will show how the total number of rounds required to send the file changes when increasing the number of corrupted channels using protocol #2 and the equation (Fig. #25). In addition in (Fig. #26) we compared them with protocol #1. By using a total of 20 channels.

The graphs below will show how the total number of rounds required to send the file changes when increasing the number of corrupted channels using protocol #2 and the equation (Fig. #27). In addition in (Fig. #28) we compared them with protocol #1. By using a total of 30 channels.

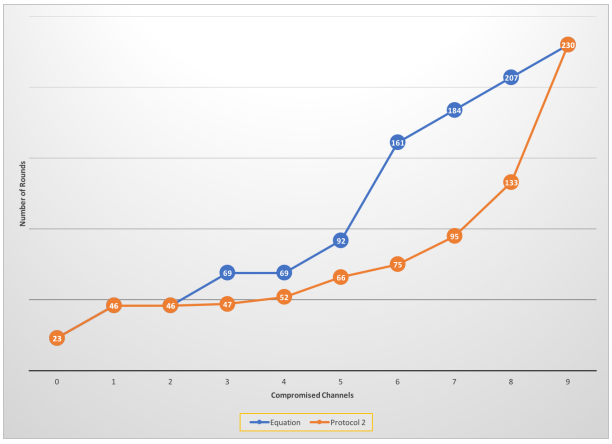

Figure 1: The difference in number of rounds taken protocol #2, and the equation

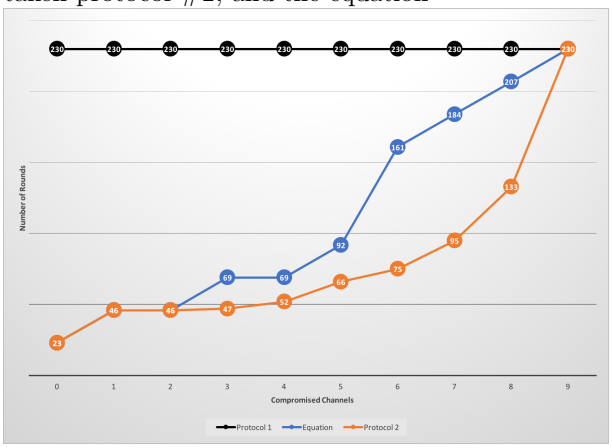

Figure 2: The difference in number of rounds taken protocol #2, #1 and the equation

The graphs below will show how the efficiency of the protocols is changing with respect to the number of corrupted channels.

- With a total number of 10 channels. (Fig. #29).
- With a total number of 20 channels. (Fig. #30).
- With a total number of 30 channels. (Fig. #31).

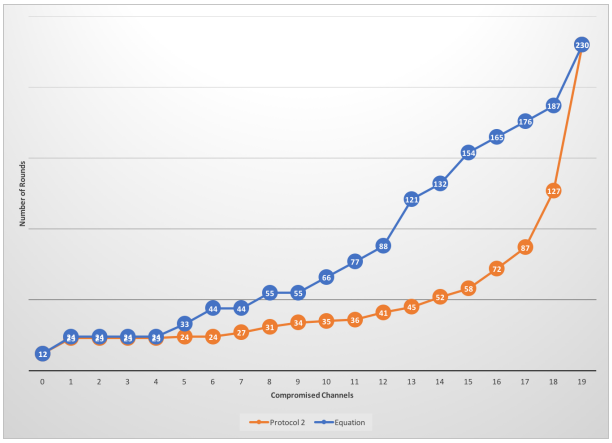

Figure 3: The difference in number of rounds taken protocol #2, and the equation

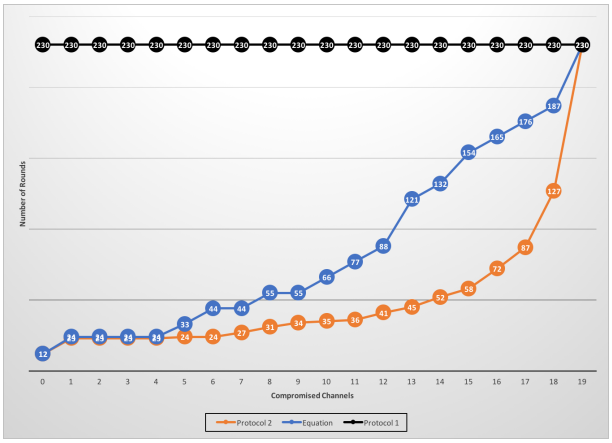

Figure 4: The difference in number of rounds taken protocol #2, #1 and the equation

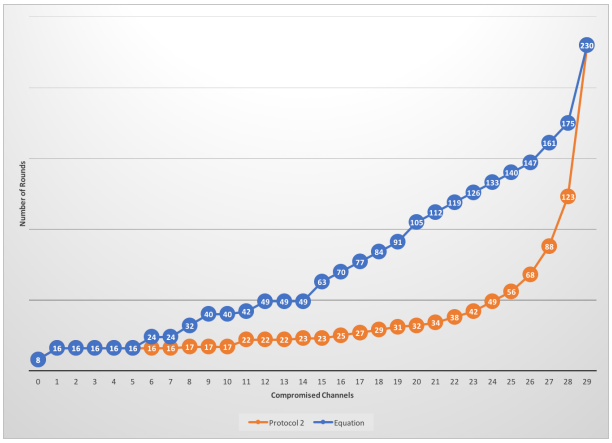

Figure 5: The difference in number of rounds taken protocol #2 and the equation

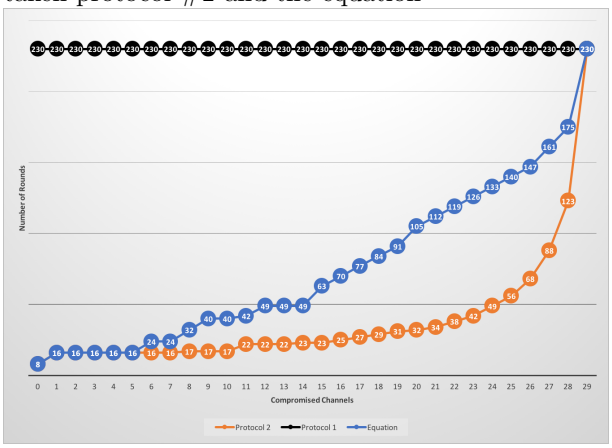

Figure 6: The difference in number of rounds taken protocol #2, #1 and the equation

0.6.1 Analysis

General Notes:

As before, we notice that Protocol #1 always needs a fixed number of rounds to send the file regardless of the number of available channels or the number of corrupted channels. And the number of rounds is the same as the number of packets to be sent which is expected from protocol #1 as it broadcasts every packet over all the channels at each round. So, by having for example, 10 packets the protocol will broadcast 1 packet in every round and therefore, to send 10 packets, it will take 10 rounds.

Now, by moving to protocol #2 we can notice that, with 0 compromised channels, when we increase the number of total available channels, the number of rounds needed to send the file decreases.

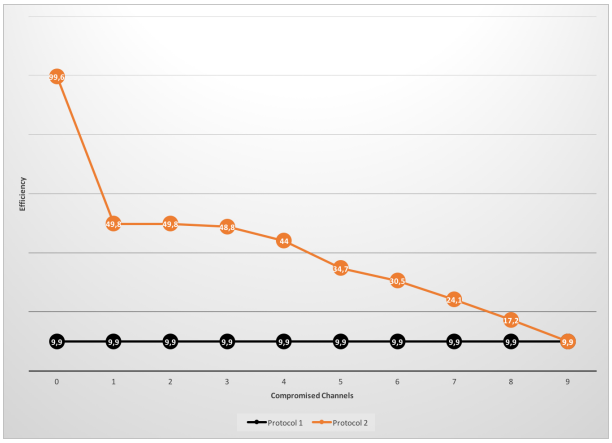

Figure 7: The efficiency of protocols #1 & #2 w.r.t the compromised channels.

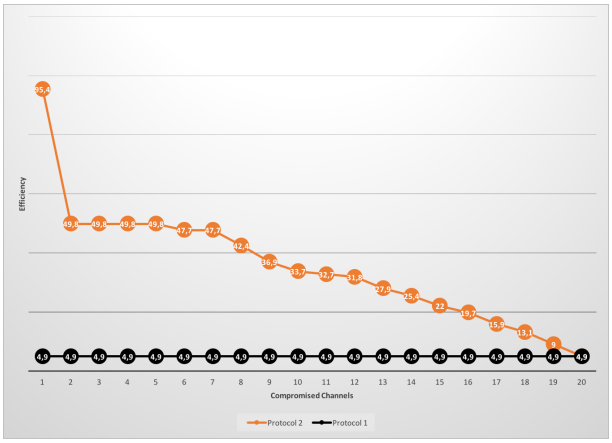

Figure 8: The efficiency of protocols #1 & #2 w.r.t the compromised channels.

With only 1 channel that is not compromised, we can notice that protocol #2 acts in the same way as protocol #1, since at every round only 1 packet is correctly received.

Case Study: Figure #23

- ***X-axis***: Shows the number of compromised/bad channels.
- ***Y-axis***: Shows the number of rounds taken to completely send the file.
- ***Blue line***: The number of rounds calculated by the equation.
- ***Orange line***: The number of rounds extracted from the experiments.

This graph visualises how many rounds have taken the equation and the experiments to send the file over the number of compromised channels.

Analyzing points

- 0 ***Compromised Channels***: Both the protocol and the equation takes the lowest number of rounds possible to send the file.

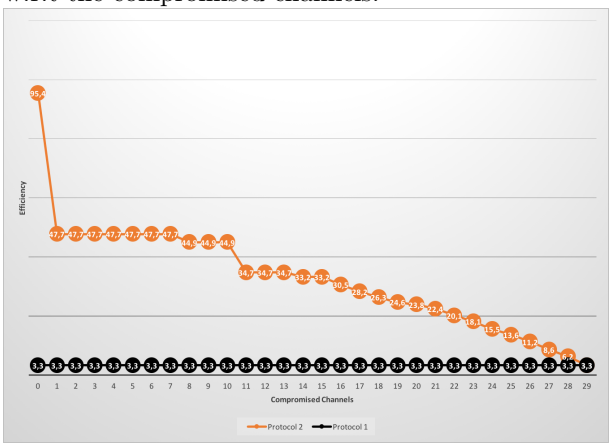

Figure 9: The efficiency of protocols #1 & #2 w.r.t the compromised channels.

- 3 ***Compromised Channels***: A turning point for both the equation line and the experiments.
- 9 ***Compromised Channels***: The peak of both the equation and the experiments lines.

comparing trends:

- $0 - 2$ ***Compromised Channels***: The lines of both the protocol and the equa-

tion slightly increase but still taking the same number of rounds to send the file.

- $3 - 8$ ***Compromised Channels***: The lines of both the equation and the experiment increase where the line of the experiment is always higher.
- $8 - 9$ ***Compromised Channels***: The line of the equation slightly increases till it reaches its maximum point while the line of the experiments increases rapidly to reach its maximum point.

Case Study: Figure #25

- ***X-axis***: Shows the number of compromised/bad channels.
- ***Y-axis***: Shows the number of rounds taken to completely send the file.
- ***Blue line***: The number of rounds calculated by the equation.
- ***Orange line***: The number of rounds extracted from the experiments of protocol #2.

This graph visualises how many rounds taken the equation, and the experiments of protocol #2 to send the file with respect to the number of compromised channels.

Analyzing points

- 0 ***Compromised Channels***: Both the protocol and the equation takes the lowest number of rounds possible to send the file.
- 3 ***Compromised Channels***: A turning point for the equation line.
- 7 ***Compromised Channels***: A turning point for the experiments line.
- 12 ***Compromised Channels***: The turning of the equation line where it started to increase significantly.
- 17 ***Compromised Channels***: The turning of the experiments line where it started to increase rapidly.
- 19 ***Compromised Channels***: The peak of both the equation and the experiments lines.

comparing trends:

- $0 - 4$ ***Compromised Channels***: Both the protocol and the equation slightly increases but still taking the same number of rounds to send the file.
- $5 - 12$ ***Compromised Channels***: The equation line is steadily increasing.
- $7 - 17$ ***Compromised Channels***: The experiment line is steadily increasing.
- $12 - 19$ ***Compromised Channels***: The equation line is rapidly increasing till it reaches its maximum point.
- $17 - 19$ ***Compromised Channels***: The experiment line is rapidly increasing till it reaches its maximum point.

Case Study: Figure #27

- ***X-axis***: Shows the number of compromised/bad channels.
- ***Y-axis***: Shows the number of rounds taken to completely send the file.
- ***Blue line***: The number of rounds calculated by the equation.
- ***Orange line***: The number of rounds extracted from the experiments.

This graph visualises how many rounds taken the equation, and the experiments of protocol #2 to send the file over the number of compromised channels.

Analyzing points

- 0 ***Compromised Channels***: Both the protocol and the equation takes the lowest number of rounds possible to send the file.
- 5 ***Compromised Channels***: A turning point for the equation line.
- 27 ***Compromised Channels***: The turning of the experiments line where it started to increase rapidly.
- 29 ***Compromised Channels***: The peak of both the equation and the experiments lines.

comparing trends:

- $0-5$ ***Compromised Channels***: Both the lines of the protocol and the equation slightly increase but still taking the same number of rounds to send the file.
- $5-28$ ***Compromised Channels***: The equation line is steadily increasing.
- $11-28$ ***Compromised Channels***: The experiment line is increasing little by little.
- $28-29$ ***Compromised Channels***: The equation line is rapidly increasing till it reaches its maximum point.
- $28-29$ ***Compromised Channels***: The experiment line is rapidly increasing till it reaches its maximum point.

Now, we can confirm the proportional relation between the number of corrupted channels and the number of rounds it takes the protocol to send a file. In other words, as the number of corrupted channels increases, the number of rounds will also increases.

Case Study: Figures (#29, #30&#31)

- ***X-axis***: Shows the number of compromised/bad channels.
- ***Y-axis***: Shows the efficiency of each protocol.
- ***Black line***: The efficiency of protocol #1.
- ***Orange line***: The efficiency of protocol #2.

These graphs visualise the efficiency of protocols #1 and #2 with respect to the number of compromised channels.

By looking at the three figures, as we noticed from the first experiment the line of protocol #1 is always steady in each figure, and the efficiency is decreasing by increasing the number of total channels. Moreover, the efficiency and the number of compromised channels are independent. And that is because protocol #1 is broadcasting the same packet over all the channels at each round. Plus, by increasing the number of channels, we're sending more packets. And with that we're making it less efficient. Therefore, we can say that, the efficiency protocol #1 is inversely proportional to the total number of available channels.

Analyzing trends of protocol #2

1. ***For Figure*** #29:
   - $0-1$ ***Compromised Channels***: The efficiency is at its maximum but it starts decreasing quickly.
   - $1-2$ ***Compromised Channels***: The efficiency is stable.
   - $2-9$ ***Compromised Channels***: The efficiency is decreasing steadily till it reaches its minimum point.
2. ***For Figure*** #30:

- $0 - 1$ ***Compromised Channels***: The efficiency is at its maximum but it starts decreasing quickly.
- $1 - 5$ ***Compromised Channels***: The efficiency is stable.
- $5 - 19$ ***Compromised Channels***: The efficiency is decreasing steadily till it reaches its minimum point.

3. ***For Figure*** #31:

   - $0 - 1$ ***Compromised Channels***: The efficiency is at its maximum but it starts decreasing quickly.
   - $1 - 7$ ***Compromised Channels***: The efficiency is stable.
   - $7 - 10$ ***Compromised Channels***: The efficiency is slightly decreasing.
   - $10 - 11$ ***Compromised Channels***: The efficiency is quickly decreasing.
   - $11 - 29$ ***Compromised Channels***: The efficiency is decreasing steadily till it reaches its minimum point.

The same deduction with the first experiment, there is an inverse proportional relation between the number of corrupted channels and the efficiency of protocol #2. In other words, as the number of corrupted channels increases, the efficiency decreases.