5. EXPERIMENTS AND RESULT ANALYSIS

This chapter covers the details of experiments carried out. The chapter is divided in two sub-sections: one covers the experiments performed to initiate attacks like session hijacking through ADD_ADDR packet and another eavesdropper in initial handshake to gain the access of security keys which can be used to initiate other attacks. Moreover, the impact of the attacks on data loss is demonstrated with graph. Next subsection covers the experiments to demonstrate the Identity Based Encryption working and its comparision with Elliptic Curve Cryptography (ECC). The performance and security evaluation of proposed model is compared with existing solutions.

5.1 RESULTS AND DISCUSSION OF SESSION HIJACKING BY MAN-IN-THE-MIDDLE ATTACK AND EAVESDROPPER IN INITIAL HANDSHAKE

In an ADD_ADDR packet session hijacking attack, the attacker first impersonates a genuine user to add their IP address as a different address and then uses this address to construct a sub-flow over an existing connection to divert traffic [23] [24] [48]. The performed attack is an example of a non-directive active attack. The data needed to announce a new address to the other host in the MPTCP version (v0) are new IP address to be annouced and address ID. MPTCP session hijacking through a MitM attack is possible due to the lack of authentication when adding a new address via the ADD_ADDR packet in the initial version (v0) of the Linux Kernel Implementation of MPTCP. The connection can be attacked with just the source IP address and port or the destination IP address and port. In this case, the destination port can be inferred from the service operating on the server, as a specific port typically indicates this (e.g., port 80 for HTTP, port 443 for HTTPs, etc.). Client port assumption is a difficult task. Scapy, DSniff, Wireshark, and other packet manipulation tools can be used to sniff the packet and extract necessary data such as IP-Port pairs, Sequence numbers, ACK numbers, etc. A tool called SCAPY for MPTCP [59] can gather the necessary four tuples and sequence numbers to launch an ADD_ADDR attack.

It just takes mentioned steps below for an attacker to perform a session hijack if they have the necessary information [23] [24].

a) It is assumed that Alice and Bob exchange information via a sub-flow in which IP addresses IP-A1 and IP-B1 are used. Over here, Eve is using IP-E1 while working in a distant location.
b) To initiate an attack, Eve employs packet-capturing programs like Scapy for MPTCP, etc., to learn the IP addresses and ports each host uses in the connection.
c) When Eve has gathered both hosts' IP and port information, he sends Bob a faked ACK packet with the source address set to IP-A1, the destination to IP-B1, and the ADD_ADDR option set to IP-E1. Here, the ADD_ADDR selection also includes the Address ID, which must be distinct; therefore, any larger random number can be chosen to avoid collisions.
d) Assuming that Alice has sent the ADD_ADDR request, Bob will start the new sub-flow by sending an SYN+ MP_JOIN request to Eve. When Eve receives a packet, she will alter its Source address to IP-E1 before sending it to Alice.
e) Since Alice believes Bob wishes to start a new sub-flow using the IP-E1 address, she will treat this request as valid. Despite Eve changing the source address, all the information necessary to validate the packet from Bob is still present. Including the necessary parameters, Alice will transmit an SYN/ACK+MP_JOIN to IP-E1.
f) By altering the source address, Eve can send the packet to Bob while giving the impression that she responded.
g) Assuming everything has been set up correctly, Bob will transmit the FINAL ACK+MP_JOIN packet to establish the sub-flow towards IP-E1.

In this manner, Eve will play the middleman between Alice and Bob. With this information, Eve can completely hijack the connection by reordering the priorities of the other sub-flows or sending an RST packet to all of the sub-flows. Figure 5.2 shows the illusionary scenario created by Eve vs. the real scenario.

Linux kernel MPTCP v0 is used for performing experiments of session hijacking by MitM, while MPTCP v1 is used for Eavesdropper in the initial handshake. Oracle VirtualBox was used to develop the experimental configuration depicted in Figure 5.1. Two virtual machines running a Linux kernel with MPTCP support are used to simulate the assault [24]. When using VirtualBox, tests may be set up quickly, with high confidence in their results and no chance of crashing or affecting the underlying kernel. The compiled kernel and tools needed to set up this “client-server” configuration may be found on the MPTCP homepage (http://www.multipath-tcp.org). Scapy, a tool for collecting and injecting packets on the

network, is used to simulate the attacker host to launch both attacks. Nicolas Matre has made an enhanced version of Scapy available on the https://github.com/nimai/mptcp-scapy project, tailored specifically for MPTCP [53]. The Scapy tool has built-in features for sniffing, modifying, capturing, and matching the request-response necessary for launching an attack.

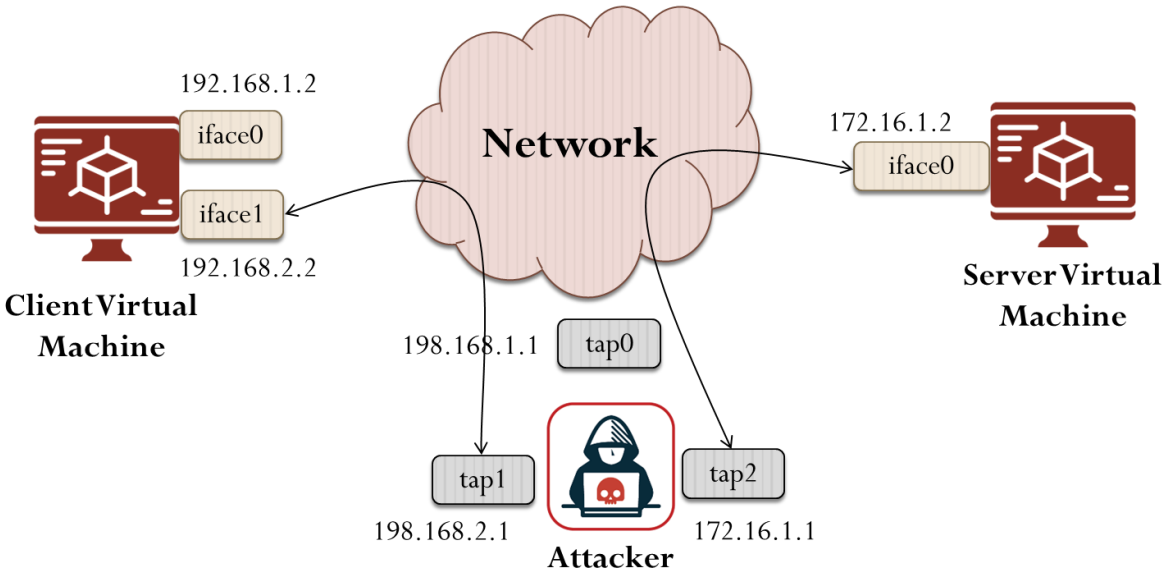

Figure 5.1 Experiment setup for performing attacks

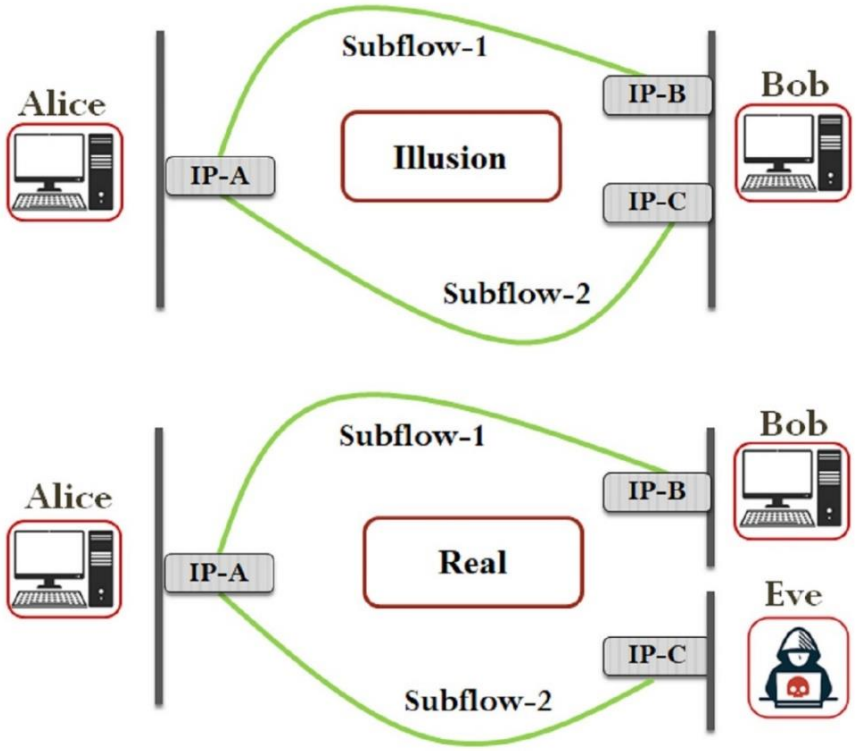

Figure 5.2 Real vs. illusion for Alice and Bob during the session hijacking attack **[28]**

As shown in Figure 5.1, an MPTCP Linux kernel has been installed on the Client Virtual Machine (VM), the Server Virtual Machine (VM), and the Host Machine. In this MPTCP setup, the client must have more network interfaces, while the server only needs one or more. Here, to build a multi-homing environment [24], the Host Machine has been outfitted with three tap interfaces (virtual network interfaces), as shown in Figure 5.3.

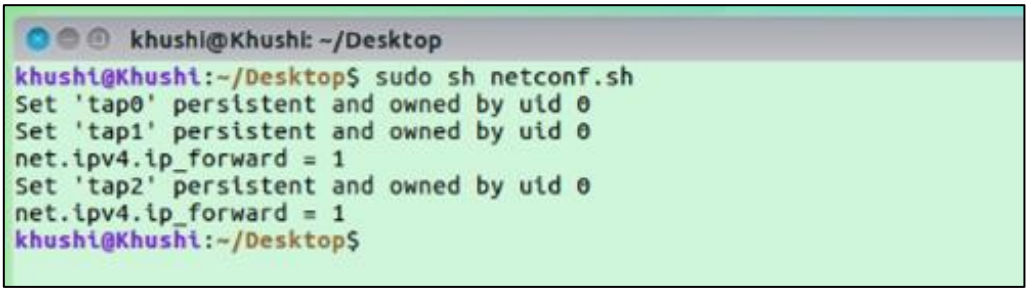

Figure 5.3 Configuration of tap interfaces

Both a chat application and a file transfer application built with the JAVA socket API on the MPTCP Linux kernel v0 were subjected to a session hijacking experiment with positive results. On average, you'll need to capture a couple of packets to collect the necessary data, including source and destination IP addresses, SEQ numbers, ACK numbers, etc. The average number of packets sent from client to server to launch an attack is 6:3. The average session hijacking attack succeeds 77% of the time.

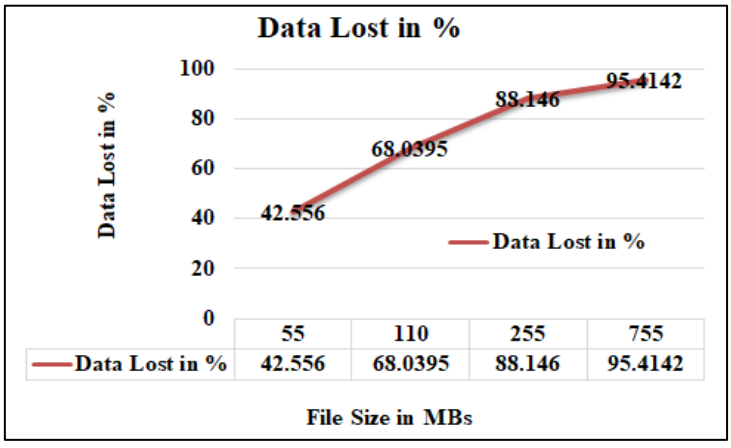

Figure 5.4 Data lost analysis during different-sized file transfer with hijacking attack

There are numerous runs of the assault done to capture a wide range of file sizes and types. Figure 5.4 demonstrates the Percentage of Data Lost (in Percentage) when Transferring Files of Varying Sizes between Client and Server. Multiple runs of the experiments are conducted, and the average loss is used as a proxy for statistical significance.

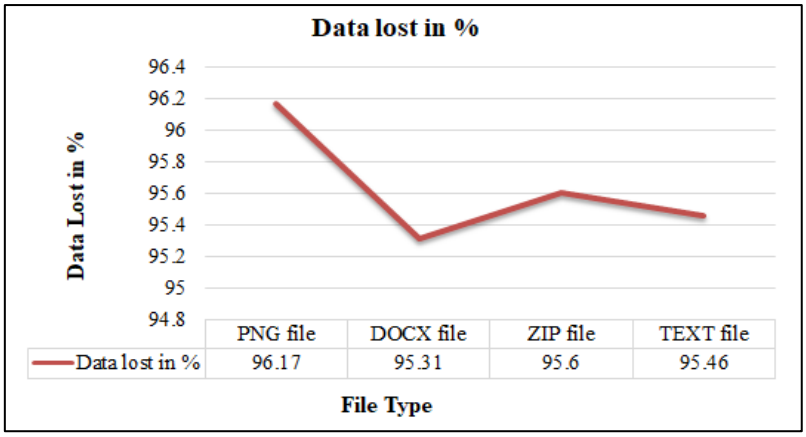

Figure 5.5 Data lost analysis in % during file transfer of different format

The accompanying graph demonstrates that data loss grows with file size—the probability of losing more than 90% of one's data after a certain point increases. Multiple runs of the same experiment are conducted using various data formats, with the results summarized in Figure 5.5. Research demonstrates that, on average, data loss ranges from 95% to 96% across file types and formats.

Another version of ADD_ADDR was integrated into the MPTCP Linux kernel v1 to fix the problem with ADD_ADDR; however, the new version is also susceptible to eavesdroppers during the initial handshake. The MPTCP v1 protocol is vulnerable to an ADD_ADDR MitM attack if an eavesdropper on the network during the initial handshake steals the keys. Here, the ADD_ADDR MitM attack can be carried out in the same way as described above once an Eavesdropper attack has been carried out using the Scapy tool to collect client-server keys. Figures 5.6 and 5.7 show the equivalent Wireshark capture to extract the keys in MPTCP v0.

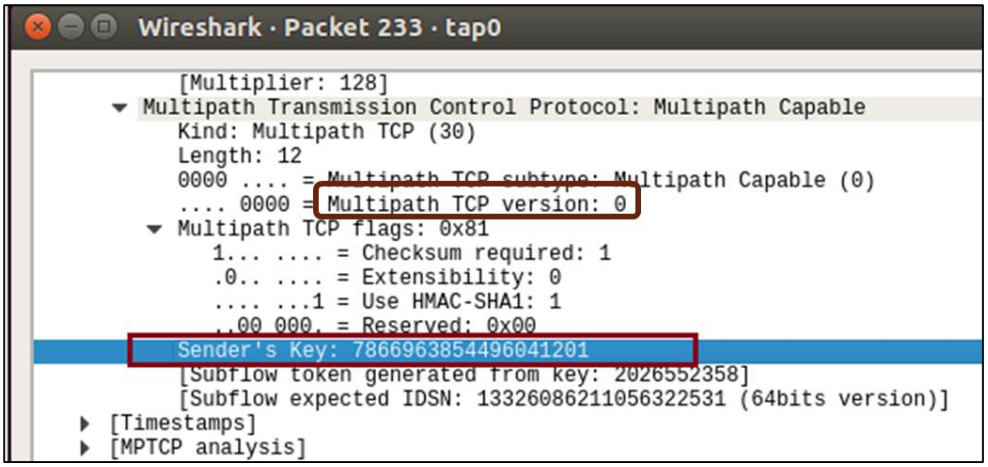

Figure 5.6 Wireshark capture for Eavesdropper at initial handshake capturing keys in clear form in MPTCP version 0

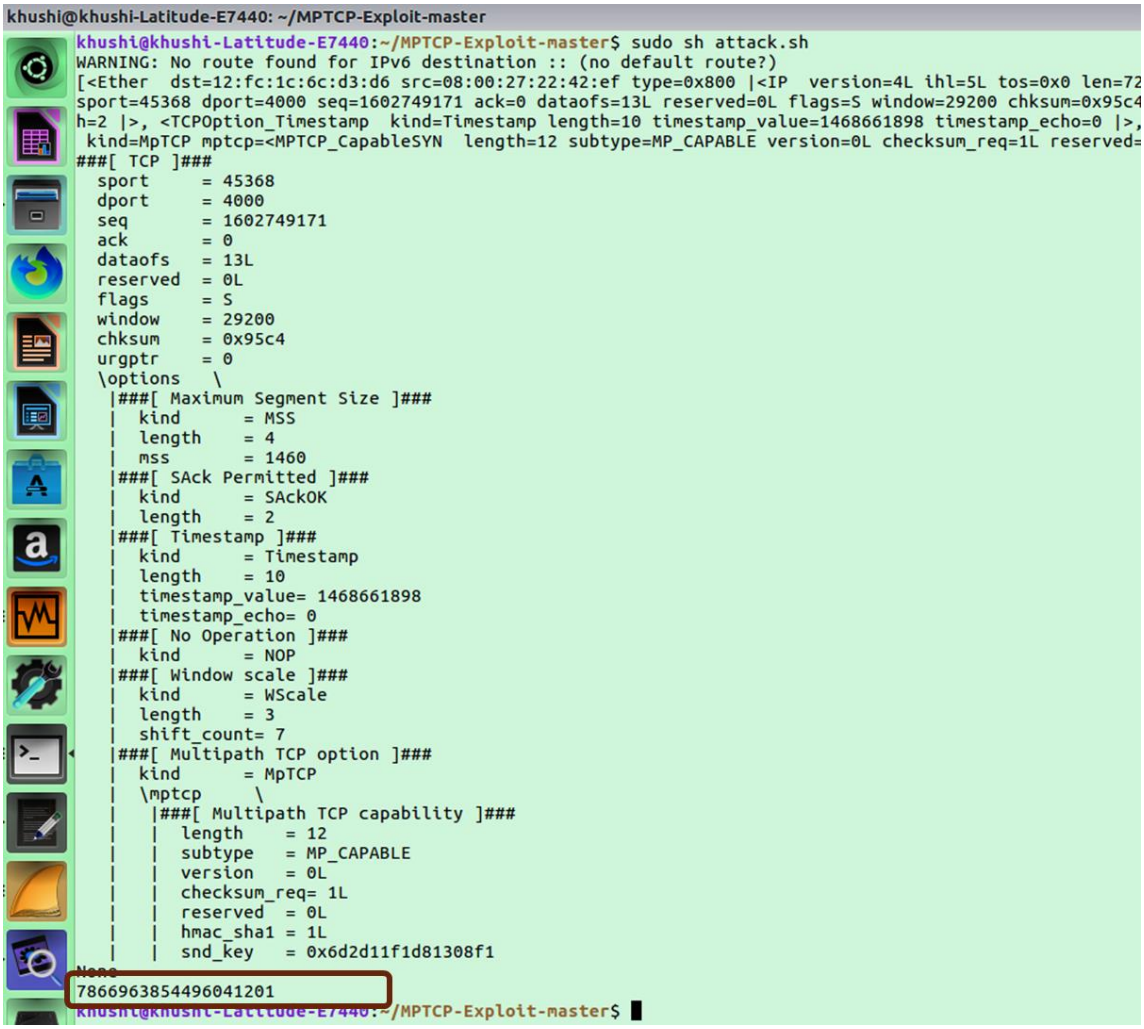

Figure 5.7 Extracting keys captured during the initial handshake to perform ADD_ADDR attack by using python script for MPTCP version 0

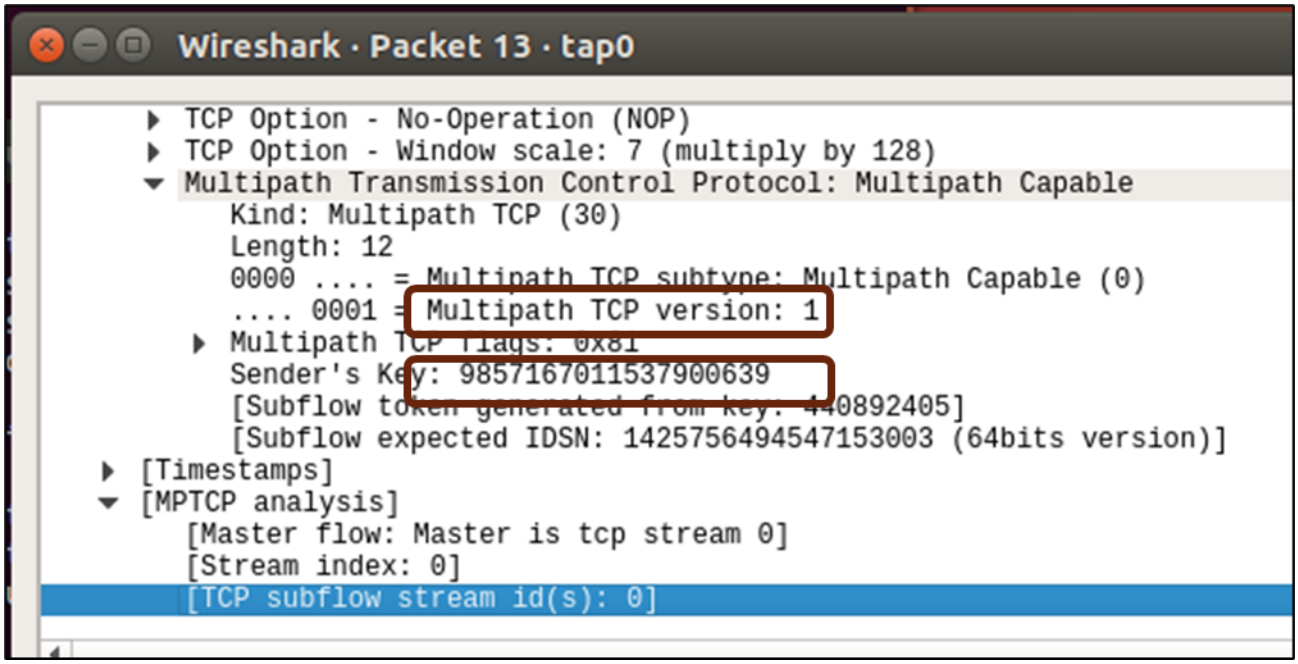

Figure 5.8 Wireshark capture for Eavesdropper at initial handshake capturing keys in clear form in MPTCP version 1

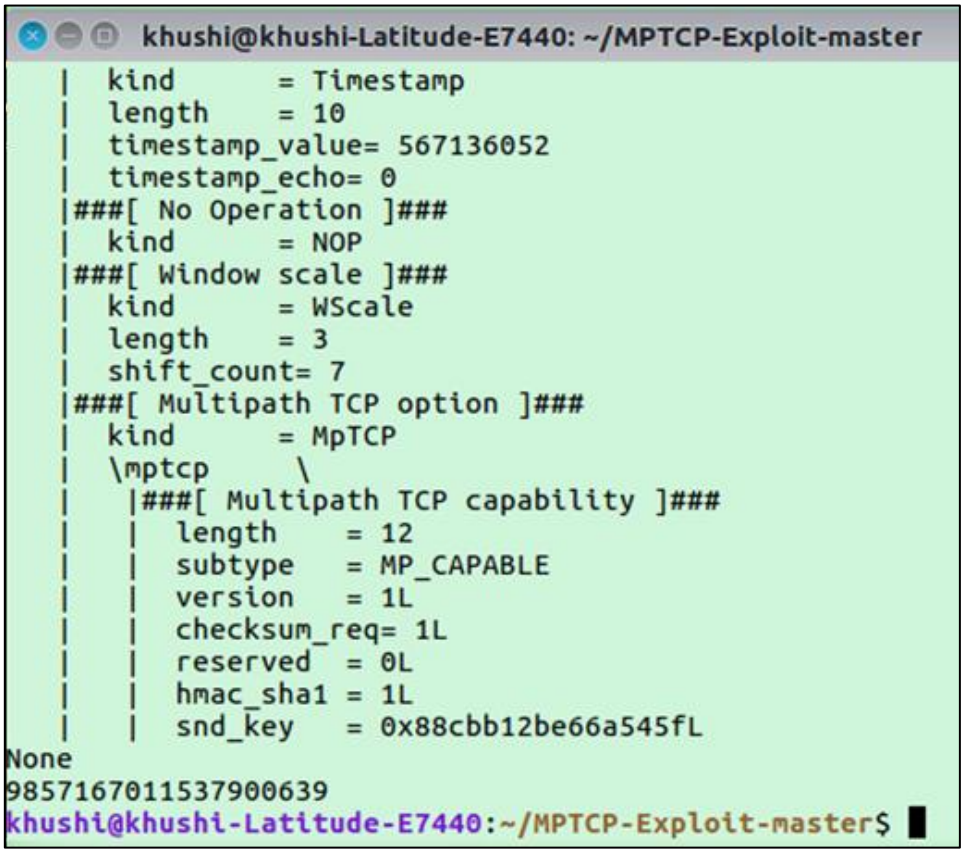

Figure 5.9 Extracting keys captured during the initial handshake to perform ADD_ADDR attack by using python script for MPTCP version 1

Session hijacking through ADD_ADDR is possible in MPTCP v0 as an off-path attack by avoiding the active communication channel and in MPTCP v1 as a partial-time on-path

attack by collecting keys while present on the communicating path for a brief time before returning to the off-path. The initial investigation makes it clear that neither version of the MPTCP Linux kernel currently available is adequate to fix the ADD_ADDR flaw. Figure 5.6 and 5.8 shows the equivalent Wireshark capture to extract the keys in MPTCP v1.

5.2 RESULTS AND DISCUSSION OF PROPOSED SECURE KEY EXCHANGE MODEL FOR MPTCP (SKEXMTCP) USING IDENTITY-BASED ENCRYPTION

5.2.1 IMPLEMENTATION SETUP

The MPTCP implementation in the Linux kernel tests the proposed system. The Oracle VirtualBoxes are used to build up the environment of the proposed approach by constructing two VMs, client and server, as shown in Figure 5.10. Both the client and server virtual machines have MPTCP, which is implemented with the Linux kernel. The PKG is set up on the host computer in this stage. The tap interfaces are utilized to establish a connection between PKG and host.

For the IBE to generate system parameters and distribute private keys based on the ID of the host, PKG is required. The primary function of PKG is to set up the master share and system parameters that will be utilized for MPTCP key exchanges. Data is encrypted using IBE in the proposed approach without requiring the sharing of encryption keys or the need for a dedicated communication host. In this scenario, PKG is crucial in authenticating users and disseminating the master private key for identity-based private key generation. Each connection will be authenticated using a digital signature, and session keys will be generated using ECC in our suggested paradigm.

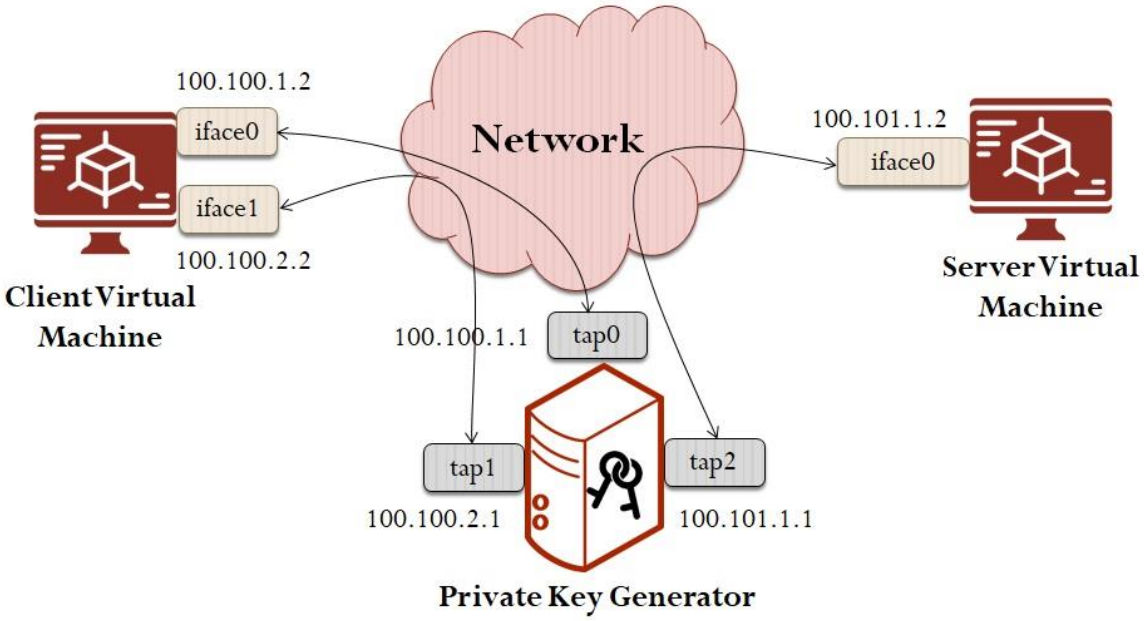

Figure 5.10 Experimental setup for testing the proposed work

Figure 5.11 and 5.12 shows the setup of the private key generator and system parameter generation for IBE. Figure 5.13 shows the master key extraction from the PKG. Figure 5.14 depicts the encyption command and step to genrate the private key correspnding to the given ID with the use of master key share.

```
khushi@Khushi: ~/ibe
khushi@Khushi:~/ibe$ ./pkghtml
Private Key Generator
running on port 12345
certificate file: ca.cert
key file: ca.priv
params file: params.txt
share file: share
mailer: /usr/bin/mail
html file: pkgform.html
banned file: banned
HelloHellokhushi@Khushi:~/ibe$
```

Figure 5.11 Setting up the private key generator (PKG) of IBE

```
khushi@Khushi:~/ibe$ sudo ./gen
[sudo] password for khushi:
Generating IBE public parameters and secret...
Parameters:
system name: test
params file: params.txt
1-out-of-1 sharing
512-bit prime
160-bit subgroup
share files:
   share
```

Figure 5.12 Public Parameter and Security Parameter Generation of IBE

```
khushi@Khushi:~/ibe$ sudo ./ibe request 10.1.1.1
request 10.1.1.1
SSL CTX: doneThreshold: 1
trying: localhost:12345
opening socket...
connecting...
server name: rooster
share password:
Verifying - share password:
10.1.1.1 testserver: Key share saved for 10.1.1.1
```

Figure 5.13 Request for master key and private key share to the PKG

```
khushi@Khushi:~/ibe$ sudo ./ibe encrypt <firstmessage.txt >cipher.txt
khushi@Khushi:~/ibe$ sudo ./ibe combine 10.1.1.1
Usage: combine ID KEYFILES...
khushi@Khushi:~/ibe$
```

Figure 5.14 Encryption step and combine the key share to generate private key for decryption

5.2.2 SECURITY AND PERFORMANCE EVALUATION

The session keys exchanged at the initial handshake are encrypted using the IBE method in the proposed model, SKEXMTCP. The client's IP address is used as a public key to implement IBE, while the server's private key is obtained via the PKG. Here, the IP address and port number are used for server authentication; using the PKG public key, they are encrypted before being digitally signed by the server. A hacker would need to defeat the encryption method and the hashing algorithm to discover the private key of PKG and decrypt the packet or alter the server's digital signature and fake the packet. So, the model's security complexity depends on the intricacy of the IBE and the encryption algorithm used to secure the private key request data.

For asymmetric encryption schemes, the chosen-ciphertext attack (CCA) is a well-accepted attack concept in which attackers can decrypt plaintexts corresponding to the chosen cipher

text. It means that the chosen-ciphertext attack (CCA) model, which may be further classified as adaptive (IDA-ID-CCA) or non-adaptive, must be used to demonstrate that IBE is secure (IDA-CCA). However, the FullIn-dent proposed by FUSAKI-OKAMOTO is a secured chosen-cipher text IBE, while the basic indent (IBE) proposed by BONAH and FRANKLIN was not, since the security of this IBE scheme relies on the bilinear Diffie-Hellman assumption (BDH) [44]. For encryption in SKEXMTCP, we assume that BDH is hard in groups and employ the FullIndent variant of safe IBE. This variant is secure in a random oracle against the chosen-ciphertext attack [44].

Messages sent between PKG and the communicating node must be encrypted using a public key encryption strategy to ensure privacy (such as RSA, Elgamal, ECC, etc.). Despite the smaller key size, ECC provides the same level of protection as RSA. If the attacker needs an exponential amount of time relative to the key size to launch an attack, the public key cryptosystem is more secure.

Among the most common approaches to solving ECC are naive exhaustive search, Baby Step Giant Step (BSGS), the square root, and Silver-Pohling-Hellman (SPH) [54]. The naive exhaustive search method solves ECC by repeatedly adding point P until it reaches Q= kP; it is computationally infeasible for many-step problems. Even though the BSGS is an improvement over the naive exhaustive search, its space and time complexity are prohibitively large because of the need for volatile memory for n points and additional n steps. Public key cryptosystems are safe from the attack since it takes an exponentially growing amount of time to solve ECC using the square root approach, and hence ECC can be regarded as secure against the square root method. In addition, the SPH is only applicable when the order of the curve is defined by picking the product of small prime numbers. It is because the computation time for SPH varies significantly for products of big prime integers, making ECC safe against SPH. The Pollard-p algorithm is another general-purpose approach for solving ECC; its time complexity is comparable to BSGS, although its space complexity is O(1). Because of the computational complexity of the Pollard approach, the cost to attack ECC-163 in 1 year is roughly USD 200 million. The current stage of the ECC security breach is described as “a 112-bit key for the prime field and a 109-bit key for the binary field being the extreme level security breach till date” [54]. As a result, ECC is the most appropriate method for protecting data in transit between nodes and PKG. To generate a session key with a higher level of security, the HMAC can be utilized, as it employs a hash function. Because of its superior security complexity, HMAC is rarely used.

SKEXMTCP's proposed model employs IBE to encrypt session keys to authenticate entities during sub-flow creation and address broadcasting. Extra packets must be transmitted between communicative nodes and the PKG to retrieve the public parameters of IBE and private key; however, this does not increase the communication overhead when using MPTCP. In addition, the host is not required to have a public key to utilize IBE to encrypt the packet. Any random string can be used to encrypt the messages, unlike in the case of public key cryptosystems like ECC, the hosts must produce the session keys before encryption. Figure 5.15 compares the time needed for key creation and encryption using ECC and IBE. The graph demonstrates that ECC takes longer because it must produce session keys for encryption, but IBE is unrestricted in its choice of encipherment string. The overall performance suffers because the developers of [31] employed ECC for session key creation and intend to use the keys for authentication.

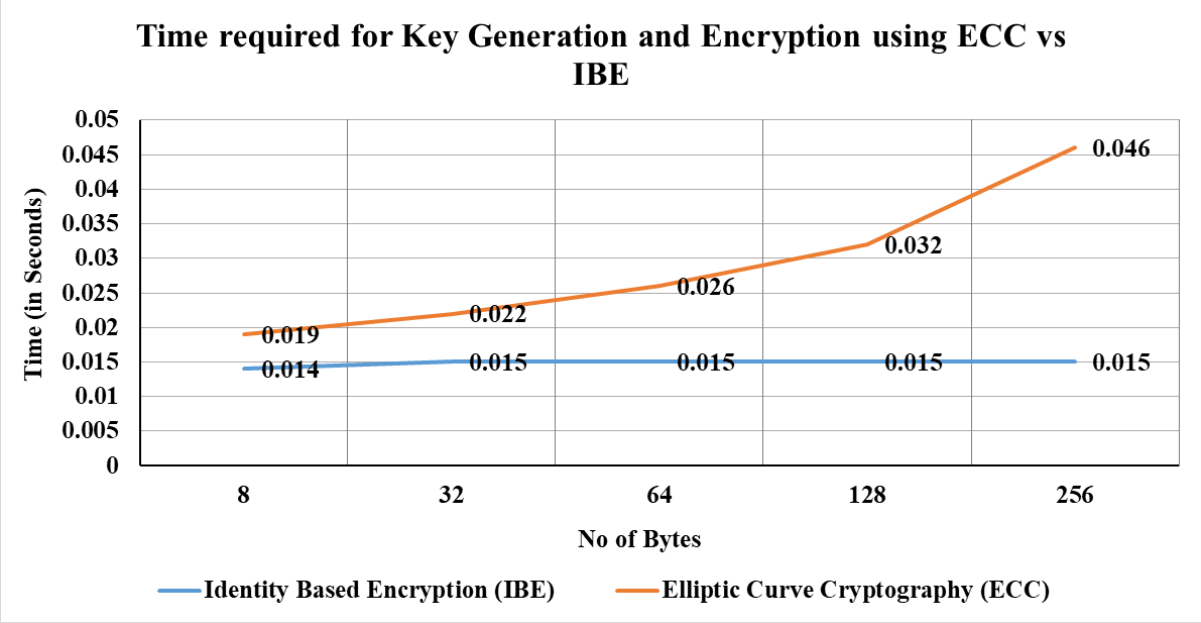

Figure 5.15 Comparison between times required for the key generation and encryption using ECC and IBE.

The cost of the proposed model in terms of implementation can be calculated by considering: (i) the cost of key generation, (ii) the cost of communication between hosts and PKG, and (iii) the cost of the 3-way handshake.

- Let us assume that the cost of key generation is n.
- One needs to consider the cost of a request for a private key from a host to PKG and a reply from PKG to a host with a private key to obtain the cost of communication between the hosts and PKG.

- Assume that the cost of a request for a private key from a host to PKG is $n1$ and the cost of a reply from PKG to a host with a private key is $n2$.
- Thus, the cost of communication between Alice and PKG to deliver a private key to Alice is $n1 + n2$, and the cost of communication between Bob and PKG to deliver a private key to Bob is also $n1 + n2$.
- Thus, the total cost for communication between PKG and hosts is $2(n1 + n2)$.
- Now, let us calculate the cost of a 3-way handshake SYN, SYN+ACK, and ACK is $n3, n4,$ and $n5$respectively.
- Thus, the overall cost is

$$N1 = 2\ (n1 + n2) + n3 + n4 + n5$$

- If we consider that the overall cost of the model is $\mathrm{O}(N) = \mathrm{O}(N1)$, then

$$N1 \ll N1 \times N1$$

Several methods have been proposed to improve MPTCP's security; these methods are compared in Table 5.1 regarding the number of bytes needed for key exchange and the delay for packet exchanges [11] [48]. You can see how different potential solutions to improve MPTCP's security stack up against one another regarding the number of bytes needed for key exchange and the absence of delays [10]. It is an example of a one-way delay, where the delay represents the additional packets that must be sent. Figure 5.16 illustrates how the suggested approach exhibits the same behavior as MPTCP regarding the number of bytes needed for the key exchange and the time it takes to complete.

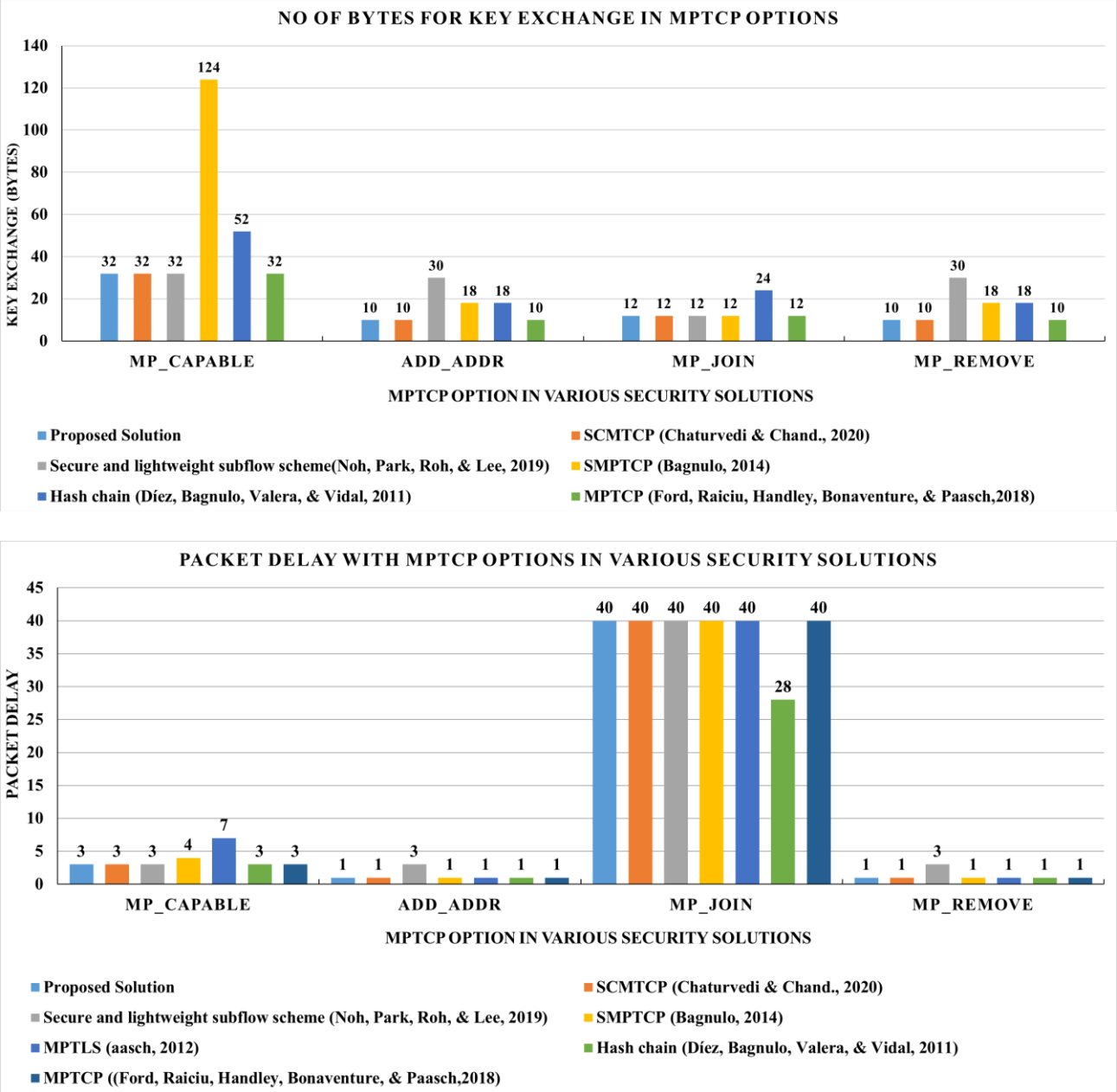

Figure 5.16 Comparative study of bytes required in key exchange with various MPTCP options in security solutions

Table 5.1 Comparative evaluation of existing security solutions

	PROPOSED SOLUTION	**SCMTCP** [43]	**SECURE AND LIGHTWEIGHT SUB-FLOW SCHEME** [11]	**SMPTCP** [55]	**MPTLS** [56]	**HASH CHAIN** [8]	**MPTCP** [21]
MP_CAPABLE							
– Key exchange (bytes)	32	32	32	124	7468	52	32
– No of delay	3	3	3	4	7	3	3
ADD_ADDR							
– Key exchange (bytes)	10	10	30	18	18	18	10
– No of delay	1	1	3	1	1	1	1
MP_JOIN							
– Key exchange (bytes)	12	12	12	12	12	24	12
– No of delay	40	40	40	40	40	28	40
MP_REMOVE							
– Key exchange (bytes)	10	10	30	18	18	18	10
– No of delay	1	1	3	1	1	1	1

Table 5.2 compares the probable security solutions of MPTCP in terms of prevention of attacks.

Table 5.2 Comparison of probable MPTCP security solutions

ATTACK	**TYPE**	**PROPOSED SOLUTION**	**SCMTCP** [43]	**SECURE AND LIGHTWEIGHT SUB-FLOW SCHEME** [11]	**SMPTCP** [55]	**MPTLS** [50]	**HASH CHAIN** [8]	**MPTCP** [21]
Session hijacking using ADD_ADDR Vulnerability	**Off Path Active attack / Partial Time on Path Active attack**	Y	Y	Y	Y	Y	N	N
Eavesdropper in the initial handshake	**On Path Attack**	Y	Y	N	Y	Y	N	N