3. LITERATURE REVIEW

To resolve the issues of MPTCP related to security, researchers have made many attempts in the last few years, and many probable solutions have been proposed. This section discusses the research work done in the area of MPTCP security. The following subsections discuss major security issues with MPTCP, available solutions, and their limitations.

There are two ways through which an attacker can get access to an MPTCP session: One by intercepting the ongoing MPTCP sub-flows as an eavesdropper and another by forging the MPTCP packet to deceive the receiver into thinking it came from a legitimate MPTCP host.

We initially classify the different attack kinds according to the attackers' location and action to comprehend the attacks over MPTCP [23] [46]. Based on the intruder's location when initiating an assault, there are three categories of intruders: off-path attackers, partial-time on-path attackers, and on-path attackers. The attackers can also be divided into active and passive categories based on their impact or action as shown in Figure 3.1.

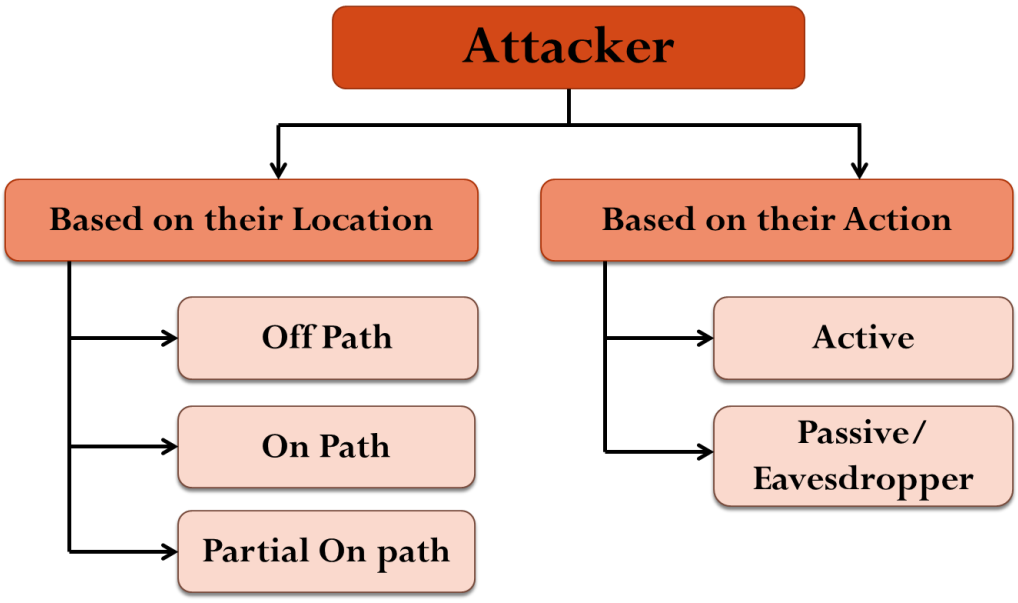

Figure 3.1 Attack Classification

For the lifespan of the connection, on-path attackers can snoop on any of the paths between the talking hosts. In contrast to on-path attackers, off-path attackers are never dependent on any of the MPTCP pathways during the lifetime of the connection. Attackers with partial time-on-path capabilities can spend at least some time on any of the paths connecting the communicating hosts. While a passive attack allows an attacker to capture and read a packet,

it does not allow the packet to be altered, delayed, or destroyed inside the same session. Sub-flows in MPTCP can travel in either direction. During an active attack, the MPTCP packets belonging to the same session may be corrupted, dropped, or delayed. Additionally, it may be moving along the MPTCP path(s) in either direction (forward or backward).

3.1 REVIEW OF MAJOR SECURITY ISSUES WITH MPTCP

The major security concerns on MPTCP Linux kernel implementation identified by IETF in their draft RFC7430 are ADD_ADDR attack, DoS attack on MP_JOIN, SYN Flooding Amplification, Eavesdropper in initial handshake, and SYN/JOIN attacks [23]. Another class of attacks that can be carried out by leveraging the MPTCP option vulnerabilities are traffic diversion attack, data sequence manipulation, etc. In this section, the major security threats to MPTCP are focused on.

3.1.1 ADD_ADDR ATTACK

As the name suggests, the attacker can carry out these attacks using the MPTCP ADD_ADDR option, classified as off-path active attacks, which affects the confidentiality of the connection [23]. An attacker can initiate the attack by forging an authorized user's IP address into the ADD_ADDR packet as an additional IP address while claiming to be the authorized user. Establishing a sub-flow over a secure connection to the same IP address is a guaranteed way to hijack a session or reroute traffic on the compromised path. The host must send the ADD_ADDR packet with the additional IP address and address identification to advertise the additional IP address.

Let's consider an ongoing MPTCP connection between two hosts, Alice and Bob, indicating that they have previously exchanged the necessary security information to initiate a new sub-flow; attacker Eve attempted to initiate a MitM attack, as shown in Figure 3.2. Firstly, Eve will forge an ACK packet with the ADD_ADDR option, indicating that Bob generated and sent the packet to Alice by listing Bob's address as the source and Alice's address as the destination.

As shown in the ADD_ADDR option header format of MPTCP, the address used to promote can be entered in the ADD_ADDR parameter. This parameter can be used by Eve to publicize her address. After receipt of the ACK packet, Alice presumed that this ACK packet with the ADD_ADDR option originated from the legitimate host Bob, so Alice sent an SYN+MP_JOIN packet with Eve's address as the destination address to start a new sub-flow between Alice and Bob. Alice sends the hash value of bob with the SYN+MP_JOIN packet, which was previously

shared in the initial key exchange. Hence, Eve receives the security information required to connect Eve and Bob with the SYN packet. Eve creates a new SYN+MP_JOIN packet with her address as the source address, Bob's address as the destination address, and the hash value of Bob acquired from Alice's SYN+MP_JOIN packet as the payload. Bob assumed that this SYN+MP_JOIN packet came from the genuine host Alice and responded to the message with SYN/ACK+MP_JOIN using HMAC generated from Alice's key and a random value. Alice received this information from Eve and assumed that it came from Bob. Alice responds the same using ACK+MP_JOIN and their own HMAC. When Bob received the ACK+MP_JOIN from Eve, Bob confirmed that Eve was on another sub-flow over the connection between Alice and Bob. Figure 3.2 depicts the whole scenario of the ADD_ADDR attack.

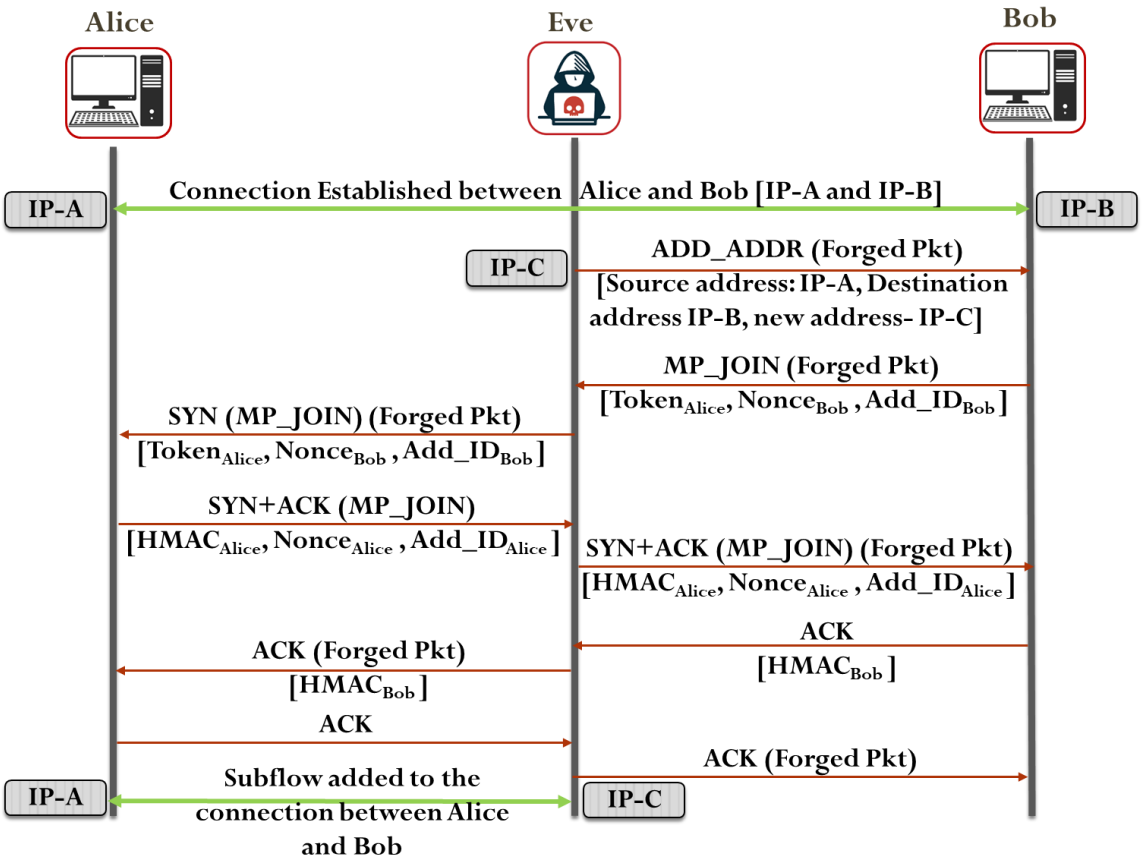

Figure 3.2 ADD_ADDR attack scenario **[24] [27]**

3.1.2 DoS ATTACK ON MP_JOIN

This assault can be considered under a category of an off-path active attack which affects the availability of resources [23]. As discussed in the previous section, a new sub-flow between two hosts can be created using the MP_JOIN option in MPTCP. The security information required to establish the new sub-flow is shared during the initial handshake, which will be

used during the SYN + MP_JOIN to identify the MPTCP. Upon receiving the SYN+MP_JOIN packet, the host establishes the state as the authentication token, and the nonce is not resent with the ACK packet. Hence, the attacker sends numerous SYN+MP_JOIN requests to the host, which creates numerous states on the server machine that exhausts the server by launching numerous partially open sessions. The situation prevents the server from initiating fresh sub-flow on the already established connection. According to the implementation, a host can only keep a certain number of half-open connections. The host becomes worn out when that threshold is crossed; this is how the attacker can use MP_JOIN to launch a DoS attack.

3.1.3 SYN FLOODING ATTACK

This kind of attack is an off-path active attack that affects the availability of resources to legitimate users. The attacker initiates the numerous half-open connections by sending SYN packets to the ports, which will occupy the servers' resources, and legitimate users will be prevented from opening the connection due to the lack of resources. Here, the attacker tries to exhaust the server with fake requests to make the service unavailable to genuine users [23].

The SYN packet can be processed without establishing the state using the SYN cookies. However, SYN + MP_JOIN requires establishing a state during sub-flow establishment, which causes a flood of requests for sub-flow establishment at the server using a wide range of IP addresses and port pairs. The attacker will only occupy one TCP connection while the server has to create a state for every half-open sub-flow connection request from the attacker.

3.1.4 EAVESDROPPER IN THE INITIAL HANDSHAKE

Exchanging keys in plain text during the initial 3-way handshake is a major security concern of MPTCP. By staying present during the initial key exchange, the attacker gets access to the keys, which they can use to start up new sub-flows or hijack the session through the MP_JOIN option or ADD_ADDR option[6]. These keys facilitate the attacker's full control of the connection, including the ability to eject reliable hosts. This exploit qualifies as a partial-time on-path attacker, which may affect the availability, integrity, or confidentiality by initiating the various attacks using information gained.

3.1.5 CROSS-PATH INFERENCE AND TRAFFIC DIVERSION ATTACK

Due to the multihoming support of MPTCP, the performance of additional MPTCP sub-flows can be determined by an attacker listening on one of the MPTCP sub-flows and the transmission path of data changed by using MP_PRIO option by inferring one of the sub-flows.

When an attacker examines the local and global sequence numbers (GSN) encoded in MPTCP headers, they can extrapolate the performance of the non-eavesdropped sub-flow. Although they appear to be independent, the MPTCP connection's sub-flows are intrinsically intertwined with one another in reality because they transmit the same MPTCP connection's data. For instance, MPTCP connection has two sub-flows so data will be transmitted through both the sub-flows and the destination host must collect the packets from all the sub-flows and incorporate them into the MPTCP connection [47]. MPTCP uses two types of sequence numbers: a global sequence number combines packets coming from all the sub-flows over same MPTCP connection to ensure reliable transmission over a sub-flow. In contrast, a local sequence number is the traditional sub-flow sequence number encoded in the TCP sequence field. An attacker can learn the performance of the conncection and the performancr of sub-flow by passively intercepting one sub-flow.

Furthermore, an adversary can block an MPTCP sub-flow by sending forged packets that designate it as the backup using the backup flag in the MP_PRIO option [25]. The sender will stop providing data after getting a control packet such as MP_PRIO, and the throughput of the associated sub-flow will decrease to zero. Sadly, unlike MP_JOIN, such a control packet does not require authentication. An attacker controlling only one path can use the correct address ID to designate any sub-flow as a backup. An attacker on one path of a multipath MPTCP connection could cause the backup sub-flow to be established by sending a fake MP_PRIO packet with the address ID of one of the other paths. Using the backup flag vulnerability, an attacker can switch the communication between sub-flows [25].

3.1.6 DATA SEQUENCE SIGNAL MANIPULATION

As discussed earlier, MPTCP uses two distinct levels of sequence spaces: One level of sequence spaces is used by MPTCP at the connection level, which is shared by all sub-flows, and the other level is used for each TCP sub-flow. The standard TCP header will be included in each segment sent through a sub-flow, with the sequence and acknowledgment numbers specific to that sub-flow. Data sequence and acknowledgment numbers make up the new

connection-level sequence space in MPTCP. The primary purpose of this component is to maintain the records of distribution of application data over various sub-flows and preserve a global mapping (across all sub-flows) of how packets are divided among sub-flows and for transmission and to reunited the packets coming from sub-flows in appropriate order before passing it on to the application [26].

Combining TCP optimistic ACKing with DSS manipulation, in which the DSS's Data ACK is changed intentionally, can have negative effects, such as generating non-responsive traffic and maliciously-induced bursts [26].

Table 3.1 Major Security threats to MPTCP

Attack	Category *	Active/ Passive	References	Security Goals Impacted #	Remarks
Eavesdropper in the initial handshake	P	Active	[23]	C	The session keys are transmitted in the plain format during the 3-way handshake, which can be used in a later SYN+MP_JOIN DoS attack or ADD_ADDR attack.
ADD_ADDR attack	F	Active	[23]	C, I, A	To create a sub-flow between the authenticated host and the attacker across a lawful connection, an attacker can forge a packet and deliver it to the genuine user while masquerading as a legitimate source.
ADD_ADDR2 attack	F	Active	[27]	C, I	An attacker can accomplish this attack by listening to a conversation between two hosts during the initial handshake. Using the keys obtained from the conversation, HMAC can be figured out.
DoS attack on MP_JOIN	F	Active	[23]	A	If the server becomes too busy processing fake SYN+MP JOIN requests from malicious users, the legitimate users will be unable to start new sub-flows.

Attack	Category *	Active/ Passive	References	Security Goals Impacted #	Remarks
SYN Flooding attack	F	Active	[23]	A	The SYN packet causes the server to become overloaded, which prevents the client from being served.
Traffic diversion attack	F	Active	[25]	C, A	A malicious actor can use cross-path inference to keep tabs on one of the sub-flows and then use a spoofed MP_PRIO packet to reroute all traffic to their controlled sub-flow.
Cross-path inferences attack	F	Active	[47]	C, A	Side channels allow attackers to deduce an unmonitored path's attributes and sensitive information, which undermines MPTCP's original design intentions.
SYN/JOIN attack	P	Active	[23]	C, I, A	During the exchange of SYN/JOIN messages, an attacker positioned strategically can add any addresses to create a new sub-flow over the connection.
Data Sequence signal manipulation	F	Active	[26]	A	On top of TCP optimistic ACKing, the connection level ACK is altered, resulting in a very effective attack scenario like DoS, flood, etc.

Keys: * Category: O, on-path; F, off-path; P, partial-time-on-path. # Security Goals: C, confidentiality; I, integrity; A, availability.

3.2 REVIEW OF EXISTING SECURITY SOLUTIONS FOR MPTCP

Numerous ways exist to strengthen MPTCP's security by thwarting various assaults like session hijacking, traffic diverting, DoS attacks, etc. To uncover the unexplored avenues for MPTCP security research, the various solutions are discussed and assessed in this section. A major security concern with MPTCP to satisfy the fundamental security requirements (confidentiality, integrity, and availability) is to protect the keys communicated at the time of connection establishment process (3-way handshake) from eavesdroppers. The attacker can use

these keys to launch additional attacks. To address the various security concerns of MPTCP, many researchers have offered various methods to enhance security, which can be classified as encryption-based, hashing-based solutions, opportunistic encryption-based solutions, and so on, as shown in Table 3.1. Still, MPTCP uses the TCP header in which only 64-bit space is available for key exchange.

Table 3.2 Cryptographic solutions for MPTCP

Cryptographic Technique	Reference	Working with MPTCP	Limitations
Elliptic curve cryptography	[48]	The points required to plot the Elliptic curve are shared in a clear format during the initial four-way handshake.	One extra packet is required to share all four points to generate the keys. Vulnerable to time-shift attack.
Hash Chain-based Encryption	[8]	During the initial handshake, the random value will be exchanged, which will be used to produce the chain of hash by applying the hashing function for the authentication while adding sub-flows and advertisement of new Internet Protocol (IP) Address (network interface).	Vulnerable to session hijacking using ADD_ADDR Attack.
Sum Chain-based encryption	[9]	It uses a mathematical equation to create a chain instead of the normal hash function.	Vulnerable to eavesdroppers in initial handshake attack.
Asymmetric Key Cryptography	[6] [7]	Public key cryptography can be used to avoid the key exchange during the initial handshake. tcpcrypt and TLS use asymmetric key cryptography.	Computational cost increases the overhead of MPTCP. Moreover, the TCP header option size is also limited.
Authentication using Hash-based Message Authentication Code (HMAC)	[21]	The truncated HMAC calculated from the keys communicated at the time of connection establishment will authenticate the user during ADD_ADDR and MP_JOIN.	Vulnerable to eavesdroppers in the initial handshake.

3.2.1 ENCRYPTION-BASED SOLUTIONS

However, the fact that MPTCP exchanges the keys in plain text format during the initial handshake poses the biggest security risk for MPTCP. However, one of the best solutions for resolving this security issue of MPTCP is using asymmetric key cryptography to secure keys from unauthorized access during the initial handshake; the space and computation cost of asymmetric key cryptography is much higher [48]. MPTCP uses the TCP packet header; only 64 bits are allocated for exchanging keys during the initial handshake. The space restriction of the TCP header as well as the complex computation of public key cryptography made applying asymmetric key encryption to encrypt the initial keys challenging.

Kim and have suggested using the Elliptic curve Diffie-Hellman key exchange to resolve the issue [48]. Elliptic curve usage reduced the space needed, necessitating the TCP payload to deliver the essential data. However, a 4-way handshake was required instead of the standard TCP 3-way handshake for the crucial negotiation phase as shown in Figure 3.3. In order to generate the shared key using elliptic curve cryptography, two points on the curve need to be exchanged. Assume that hosts Alice and Bob want to communicate using MPTCP. To initiate the connection using MPTCP, Alice sends his point x on the curve with SYN + MP_CAPABLE packet. In reply to this packet, Bob sends his point x with ACK + MP_CAPABLE. Bob sends another SYN packet with his point y, and in reply to the packet, Alice sends ACK with his point y. Communicating hosts can exchange points by using the mentioned four-way handshake. The hosts generate their shared key using the exchanged points over an elliptic curve, which will be used during MP_JOIN for authentication. Here, the points over the curve are communicated in clear form, which can pave the way for a future time-shifted attack.

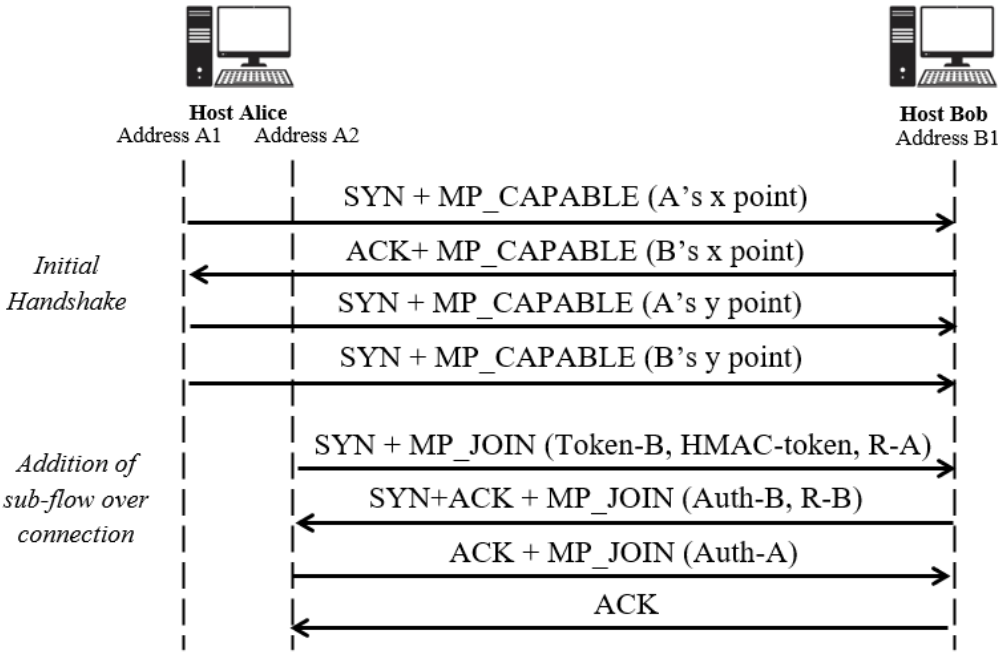

Figure 3.3 Elliptic Curve Cryptography-based solution **[48]**

3.2.2 HASHING-BASED SOLUTIONS

Authors proposed one hashing-based solution to address the issue of session hijacking using a MitM attack [8]. The hash chain approach used by the authors is based on recursive hash algorithm execution to produce a list of chained hash values. In this case, we will assume a seed value and use it to generate ***i*** hash values, H_0-H_i, by iteratively running the hash algorithm. Both hosts will trade their initial random numbers during the handshake, and the host will need to submit chained values for authentication during the subsequent sub-flow establishment as shown in Figure 3.4 [8].

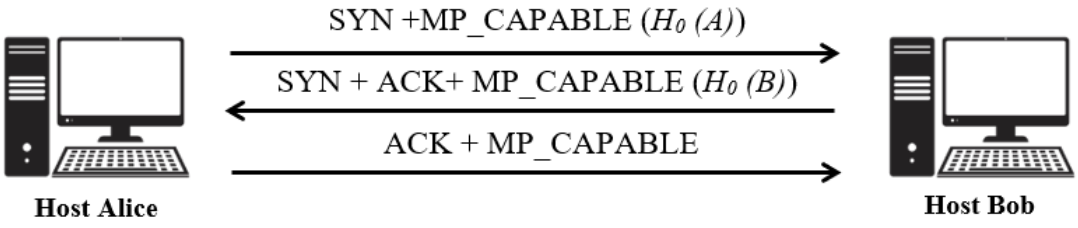

Figure 3.4 Hash Chain-based initial handshake

H_i = *Hash (seed)* *Here, seed= random value*

H_{i-1} = *Hash* (H_i)

...

H_1 = *Hash* (H_2)

H_0 = *Hash* (H_1) == *Anchor value*

Hash Chain = $H_0, H_1, H_2, \ldots H_{i-1}, H_i$

As shown in Figure 3.4, communicating hosts exchange their respective anchor values, H_0 and MP_CAPABLE. The next hash chain value will be used to authenticate the upcoming sub-flow establishment process, which the peer host can verify by matching the same with the stored hash chain value. The last chain value can be replaced with the new chain as well. When a hash chain is about to reach its limit, a new one can be generated (using a new random number as the seed). A special message is sent with the following information to join the two chains together: the penultimate value of the old hash chain $(old)H_{n-1}$; the anchor value, new H_0 of the new hash chain, authenticated by $(old)H_{n-1}$; and a keyed-HMAC to prevent the insertion of a forged hash chain. With this HMAC, the new anchor is incorporated, and the previous hash chain's final value $(old)H_{n-1}$ serves as the key.

However, it doesn't offer protection against attackers who might drop the genuine MP_JOIN request during the first handshake. In addition, new hosts can be authenticated using hash values; however, the ADD_ADDR vulnerability is not addressed. Moreover, hashing algorithms demand high computation power.

Another hashing-based solution that uses mathematical approaches to generate new sum chain-based solutions have been proposed to address the issues of memory and computation power related to the hash chain-based approach [9].

Picking a big number (S_0) and a big enough prime (p) yields the sum chain. Then, two random numbers, a_0 and b_0 are created. In this case, the initial link in the sum chain would be S_0. After summing S_0, a_0, and b_0 and dividing that total by p, the residual is S_1, the next element in the sum chain. New values for a and b are generated for the sum chain. It is possible to give a mathematical expression for the procedure, which goes as follows:

S_0 = large random number

a_0, b_0 are random numbers & $S_0 < a_0, b_0$

p is a large prime number

$S_1 = (S_0 + a_0 + b_0) \bmod p$

In general,

$S_{i+1} = (S_i + a_i + b_i) \bmod p$

Although the suggested approach may handle low memory requirements, it is susceptible to integrity time-shifting attacks. Both of these techniques have been proposed to improve MPTCP's security. Still, neither can prevent an eavesdropper from listening in on the protocol's initial handshake, which can open the door to other assaults.

3.2.3 OPPORTUNISTIC SECURITY SOLUTIONS

tcpcrypt (an extension of TCP) [7] and Secure Multipath Key Exchange (SMKEX) [49] are opportunistic security protocols that use asymmetric key cryptography with the help of certificate authorities to enhance the security of MPTCP. tcpcrypt offers cryptographic security for session data and enables MPTCP to distinguish between individual TCP sub-flows using a "session-id" not dependent on an IP-Port combination. tcpcrypt employs an asymmetric cryptosystem to avoid key disclosure throughout the handshake, necessitating an additional message for key exchange. To some extent, tcpcrypt is more secure, but a MitM can attack it because of the first handshake [7].

To provide some protection against active MitM attack, SMKEX employs the DH key exchange protocol. Multiple channels allow the parties to communicate the security key [49]. However, it doesn't work when synchronized active attackers are on every path. Transport layers opportunistic security solutions like tcpcrypt and SMKEX rely heavily on RSA and DH, which require a lengthy key to exchange the security key. Techniques using the DH paradigm are equally vulnerable to the MitM attack. It takes a lot of work to calculate the security parameters while exchanging a long security key between communicating hosts; hence the packet header must be large.

3.2.4 MPTCPSec

To safeguard application-level data and to identify and recover from packet injection attacks, MPTCP Secure (MPTCPSec) was proposed [41]. They structured the protocol into three phases

to accomplish these goals. The three phases include encryption suite negotiation, a secure handshake, and protecting data and control. One of the fundamental activities of the original MPTCP is key negotiation. However, they found that the primary problem with MPTCPsec was sending keys in MP_CAPABLE. To negotiate the encryption parameters between the nodes, they altered MP_CAPABLE by removing the keys and adding a new option called MPTCPesn [41]. The keys and session IDs were generated using the chosen secure protocol and MPTCPesn. Another use of MPTCPsec is data protection using AHEAD methods. Many TCP options were used in MPTCP. MPTCPsec calculated an authentication tag and added it to the TCP payload to safeguard these TCP options' integrity. Therefore, verifying whether the middleboxes have changed the TCP options' content is possible.

3.2.5 ADD_ADDR2

In MPTCP, the ADD_ADDR option is used to communicate the accessible interfaces hosts. Attackers can take advantage of a MitM attack on MPTCP connections by choosing this option. The format of the ADD_ADDR option is changed to the ADD_ADDR2 option as a remedy to lessen this vulnerability as shown in Figure 3.5 [24] [21]. The suggested remedy is to include a new field with an HMAC value at the time of advertisement of address for the authentication as shown in header format in Figure 3.6. The key used in the original key exchange is the HMAC key. Even if the attacker intercepts the initial key exchange, they may still be able to use this attack.

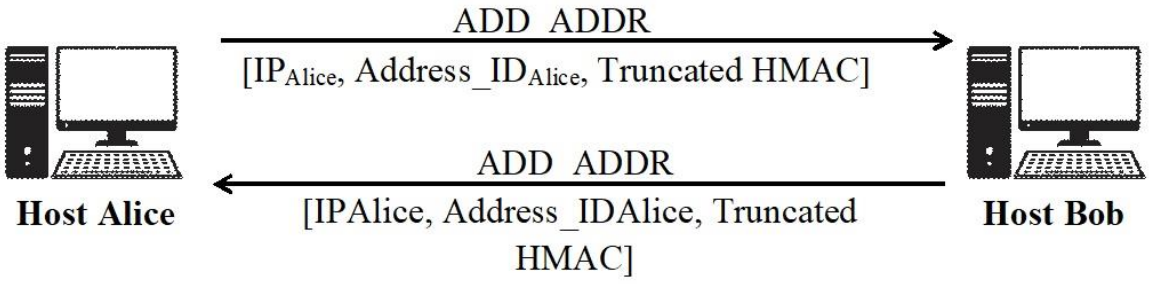

Figure 3.5 ADD_ADDR2 Packet exchange

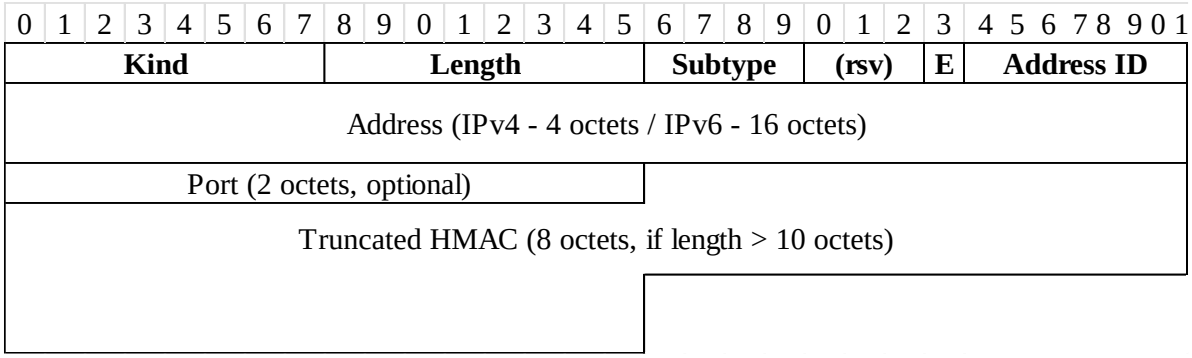

Figure 3.6 ADD_ADDR2 Option

3.2.6 OTHER SOLUTIONS

I. Use of external keys

One of the solutions proposed by researchers to authenticate the MPTCP connection is to use external keys, such as Secure Socket Layer (SSL) or Transport Layer Security (TLS), already negotiated in the application layer. A technique for transferring keys from the application layer to the two MPTCP socket types has been proposed [50]. An MPTCP ENABLE APP KEY or MPTCP KEY can notify the MPTCP protocol that application-level keys are being used for authentication. An MPTCP KEY can supply the MPTCP layer with the application-level key.

Additionally, Multipath Transport Layer Security (MPTLS) proposes combining TLS with MPTCP to fix MPTCP's security flaws [51] [6]. As a result of the adjustments made to the MPTCP and TLS, the two protocols are now more tightly coupled. TLS has been tested to function with MPTCP without affecting performance [6]. However, MPTCP needs to support message mode service instead of the byte stream, and keys generated by TLS must be used with the MAC approach to authenticate messages and a new sub-flow.

II. Use of ADD_ADDR packet confirmation

An attacker persistently forges an ADD_ADDR option to generate bogus sub-flows and then hijacks the connection between the MPTCP endpoints is one of the attacks considered for the proposed ADD_ADDR packet confirmation-based solution [11]. A digital signature-based system is a foolproof and easy way to verify the authenticity of previously transmitted ADD_ADDR. A backward address confirmation-based solution is proposed to eliminate the issue of key leakage during the initial handshake. Upon receiving the ADD_ADDR2 option, a host retransmits the packet back to the peer host to verify that the peer host received the ADD_ADDR2 option [11]. Figure 3.7 shows the packet confirmation sequence. Suppose we assume that Host Eve could overhear the original handshake and learn the keys of

communicating hosts, Key_A and Key_B. Host Eve generates an acceptable HMAC and sends it to Host Bob along with the ADD_ADDR2 packet. Without verifying the integrity of the received ADD_ADDR2 option, Host Bob sends it through connection IP A1 to the connected Host Alice. If the received address matches the one supplied in the ADD_ADDR2 option, Host Alice sends Host Bob a signed ACK0 message. Otherwise, Host Alice will deliver a Warning message if the two addresses do not match. Upon receiving the Ack from Host Alice, Host Bob will initiate the new sub-flow.

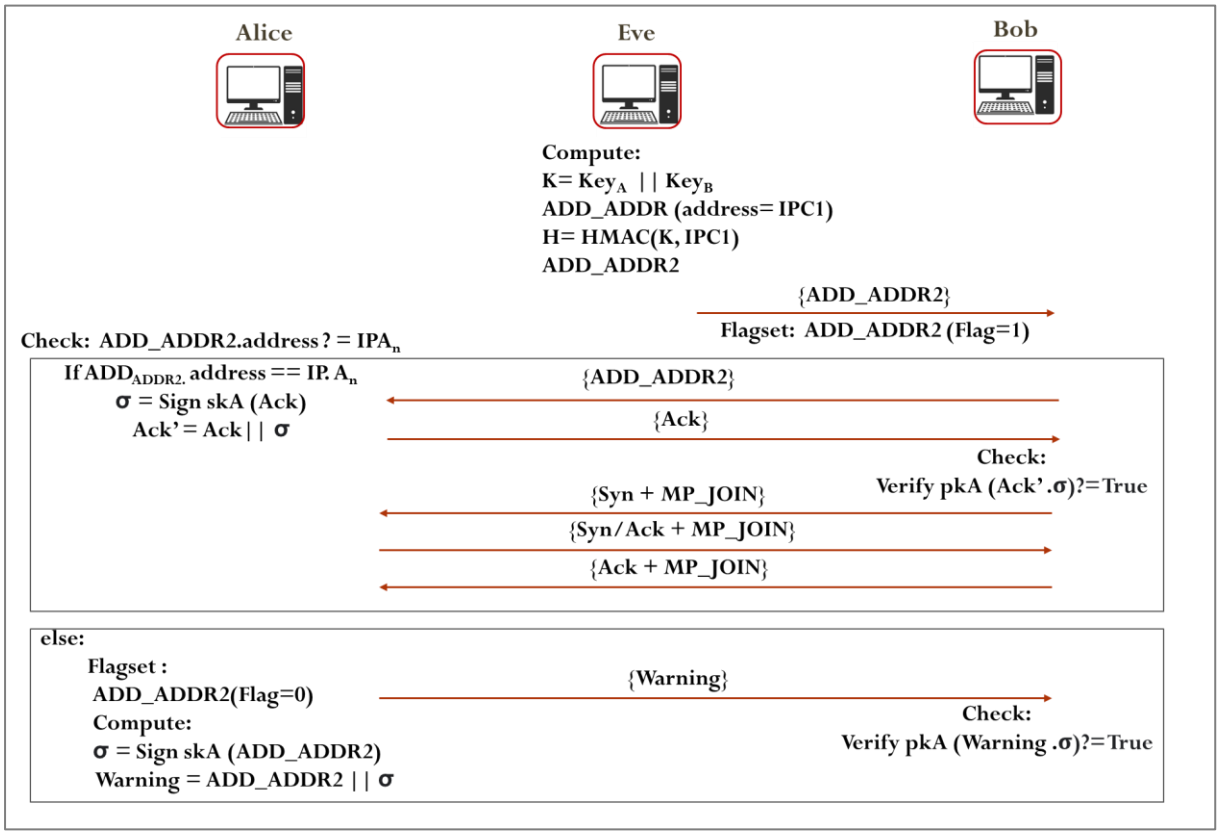

Figure 3.7 Secure and lightweight solution based on packet confirmation **[11]**

III. Key exchange using Software Defined Network (SDN)

One of the rising networking paradigms, Software Defined Network (SDN), consists of three layers: application, control plane, and data plane; it enables programmability, flexibility, manageability, and centralized control logic that delivers a global network perspective for security benefits. These features of SDN can be utilized to secure the initial key exchange by securely distributing the keys between communicating hosts over multiple paths. The SDN controller plays a vital role as the intelligence and complexity of whole network is being manged by it, and it also configures network nodes according to predetermined rules. At the same time, Data plane devices in SDN are reduced to generic packet forwarding nodes [10].

A third-party security module (python) for distributing session keys among communication hosts runs atop an SDN controller, taking advantage of the control plane's centralization. The module listens for controller packets seeking a session key, authenticates MPTCP users, requires receiver confirmation, and transfers keys after receiving a request [10].

Figure 3.8 demonstrates the process of session key distribution [10] among communicating hosts.

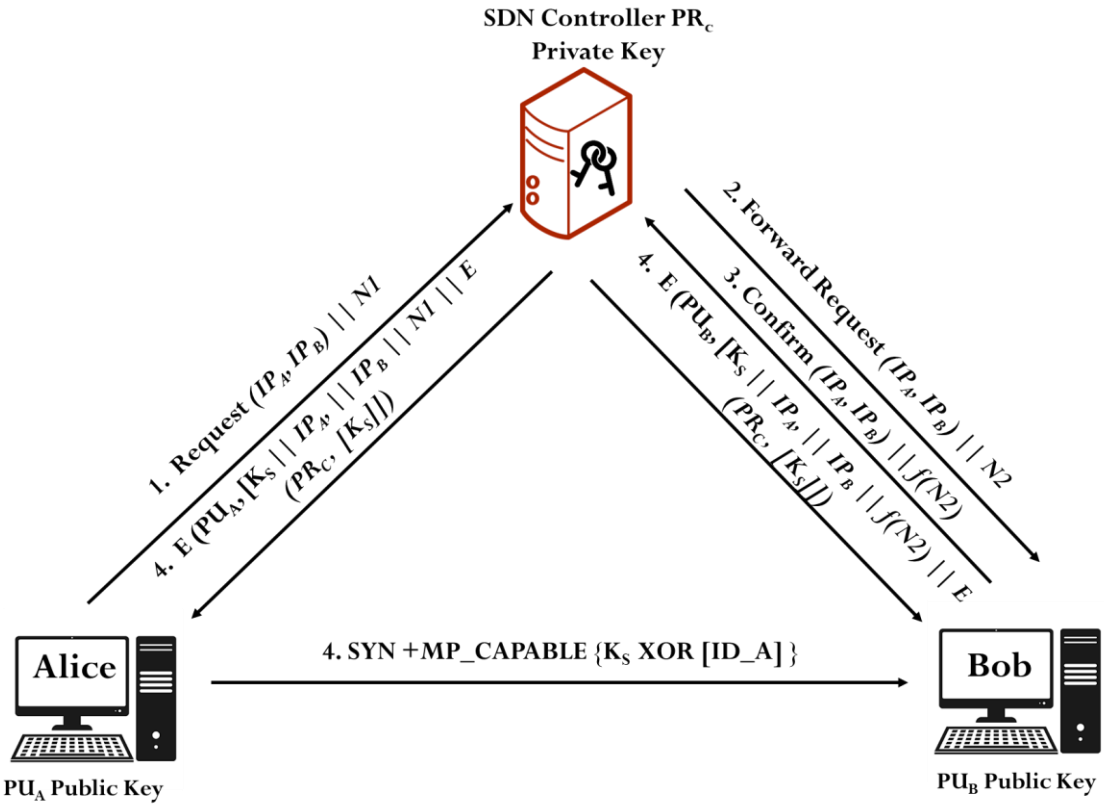

Figure 3.8 Session Key Distribution using SDN **[10]**

Host Alice first requests K_s from the controller. The request packet contains IP addresses IP_A and IP_B of Host Alice and Host Bob, as well as a nonce N1 to protect the request packet from a replay attack. In the next step, the security module transmits a “Forward-Request” packet with the two hosts' IP addresses and a new nonce N2 to Host Bob. In the next step, Host Bob sends a “Confirm” packet which includes the IP_A, IP_B, and N2 protocols. In step four, the security module generates a 64-bit session key and encrypts data packets destined for Host Alice (PU_A) and Host Bob (PU_B) with their respective public keys. The security module signs the payload, which is only encrypted with public keys. The module's signature is an encrypted version of the session key that was generated using the private key PRC of the security module.

Finally, the security module stores IP addresses, session keys, request times, and delivery times in its database for forthcoming usage. Thus, the security module manages user authentication and record-keeping databases.

Hence, the connection can be established by following 3-way handshake procedure only after getting the session key K_s. the value calculated by XORing the session key and user ID will be communicated with SYN+MP_CAPABLE packet. In this method, the sender's and receiver's identities are concealed from eavesdroppers on the same sub-flow. Thus, the user-specific 64-bit IDs will not be inserted as plaintext but instead sent after encryption.

Hence, using SDN, numerous assaults, such as the "Eavesdropper in the Initial Handshake" attack, can be avoided by encrypting "user-specific" IDs and keeping the ciphertext to a maximum of 64 bits [10].

Table 3.3 Probable Security Solutions proposed by various researchers

Reference	Year	Solution	Category	Remarks
[8]	2011	Hash chain-based solution	Hashing based	Unfortunately, it does not protect against on-path active threats.
[7]	2014	Tcpcrypt	Opportunistic	Because it does not perform public key authentication, it leaves itself open to assaults involving MitM.
[6]	2016	MPTLS	Asymmetric key cryptography	It generates computation overhead during the initial handshake. We need to modify the packet sequence.
[51]	2015			
[48]	2016	Modified initial handshake	Asymmetric key cryptography	Expenses are incurred during the first handshake due to computation. It demands altering the packet's sequence.
[9]	2017	Sum chain-based solution	Hashing based	Vulnerable to time-shifted attack.
[52]	2017	Data Scrambling technique for privacy		The suggested model is limited to concentrating on the eavesdropper on untrusted routes and thus does not function in the

				traditional sense. In addition to this, there is no assurance that the data will not be altered.
[21]	2018	ADD_ADDR2	ADD_ADDR	Vulnerable to time-shifted attack.
[10]	2019	Key exchange through SDN	Other	Single point of failure.
[43]	2020	Secure connection Multipath TCP (SCMTCP)	Other	It produces a distinct key for each available choice for every new connection request, increasing the work needed.
[11]	2019	Secure and lightweight connection establishment scheme	Other	It's insecure against eavesdropping during the initial handshake, adds unnecessary overhead to every packet it sends, and confirms the new address.

The main obstacle to delivering robust encryption-based security in MPTCP is the maximum length of the TCP option field. Numerous solutions, including those based on hashes, sum chained hashes, elliptic curves for transferring keys, etc., have been put forth and implemented, but they still fall short of expectations and are susceptible to other types of attacks.