Multipath TCP D. Kim
Internet-Draft Sungkyunkwan University
Intended status: Standards Track October 24, 2016
Expires: April 27, 2017

Efficient Design for Secure Multipath TCP against Eavesdropper in Initial Handshake
draft-kim-mptcp-semptcp-00

Abstract

Multipath TCP has become the transmission technique of choice for the multi-homed environment. Recently, there have been multiple attempts to verify the security of Multipath TCP; but an eavesdropper in the initial handshake breaches the primary security goal of Multipath TCP. In this paper, we introduce a secure scheme against an initial eavesdropper, using asymmetric key exchange.

We optimize the public parameters to overcome two challenges to the use of asymmetric cryptography. Then we show that compared to previously proposed methods, our scheme has low overhead, and is more secure. Our approach applies to many weak authentication-based protocols that seek to use asymmetric cryptography.

Requirements Language

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC 2119 [RFC2119]

Status of This Memo

1. Introduction

TCP is currently restricted to a single path per connection, yet most state-of-the-art devices often support multiple network interfaces. Multipath TCP (MPTCP) [RFC6824] is a major extension of TCP that enables hosts to use multiple paths to concurrently transfer data for a single connection. Concurrent transfer through multiple subflows for a single TCP session could improve the throughput and overall usage of the network resource.

The primary security goal of MPTCP aims at being no worse than TCP security. MPTCP currently provides security by exchanging keys during the initial handshake. These keys are used to create HMACs to authenticate other hosts. Exchanging keys in plaintext during the initial handshake is vulnerable at the viewpoint of security. An eavesdropper in the initial handshake can hijack the MPTCP session using exchanged keys even after leaving the on-path location. An active attacker can hijack the session by dropping the request for adding subflow, and can then initiate the subflow using received values within the request.

These threats are considered acceptable. The root cause of the threats is that the attacker could exploit the authentication values, whether the shared keys are exposed or not. After establishing the subflow, the attacker can launch the attack [RFC6181].

Asymmetric key exchange allows hosts to share the key without exposure. Adopting SSL, an MPTCP session can negotiate shared keys between the end-points. However, the overhead of SSL handshake is too high, considering that it occurs at every establishment of MPTCP. The overhead of the initial handshake affects the overall TCP throughput.

Moreover, a short connection in MPTCP maximizes the reduction of throughput. Our priority design goal is to minimize the initial handshake. However, low-overhead design using asymmetric cryptography is difficult, since public information needs a large-sized space. MPTCP uses the TCP option, and the maximum size of the option header is 40 byte, excluding the MPTCP header. If public information could not be inserted in the option header, additional packets are required for an exchange, since SYN packets cannot involve TCP payload. Additional packets cause time and space overhead.

We solve these limitations to optimize the public parameters considering the characteristics of MPTCP. We propose a secure design against an eavesdropper in the initial handshake. The proposed design is low overhead, and more secure compared to other schemes that use asymmetric cryptography.

2. Terminology

This document makes use of the following terms:

o Multipath TCP, MPTCP: refers to [RFC6824]. And every operation in MPTCP follows [RFC6824]

o Eavesdropper in initial handshake: refers to an eavesdropper present in the inital handshake where the keys are exchanged can hijack the MPTCP session at any time in the future. This is partial-time on-path eavesdropper and is decribed in [RFC7430]

o Off-path attacker in subflows: refers to an attacker not present in any subflows. This type of attacker could be present in initial handshake.

o On-path eavesdropper in subflows: refers to an eavesdropper present in one or more subflows. This type of attacker could be present in initial handshake. This attacker can acquire information from the subflows, however, cannot change or drop the message between the legitimate parties.

o On-path active attacker in subflows: refers to an active attacker present in one or more subflows. This type of attacker could be present in initial handshake, and this type of attacker can acquire, change and drop the message between the legitimate parties.

o ADD_ADDR Attack: refers to an attack using ADD_ADDR option. Detail explanation is described in [I-D.ietf-mptcp-rfc6824bis].

- Data encryption: refers to the possibility of data encryption using any encryption algorithms without key exposure. It simply means secure key exchange.

3. Security Threats in Multipath TCP

The fundamental goal of MPTCP is to provide security that is no worse than TCP. IETF documentation does not concern itself with threats that are applied to both TCP and MPTCP. Of course, threats on TCP can influence MPTCP, the extension of TCP. IETF documentation considers only the threats that are specific to MPTCP and are impossible with TCP. To guarantee security, MPTCP adopts the HMAC-based handshake described in Sections II.A and II.B. Researches that analyze the possible threats of current MPTCP implementation are investigated to verify the security provided to at least TCP level [RFC7430][SecEval-MPTCP]. They classified the attackers depending on location as follows:

- An off-path attacker does not need to be located in any of the subflows of the MPTCP session. The off-path attacker cannot eavesdrop any of the packets of the MPTCP session.

- An on-path attacker needs to be on at least one of the paths during the whole lifetime of the MPTCP session.

The off-path attacker is the most restricted model to attack since she doesn’t know any information for an attack. Vulnerabilities in conditions of the off-path attacker have great impact, because they are vulnerable to any attacker model. It is most difficult to provide security against an on-path attacker who can eavesdrop every packet of information used for an attack. [RFC7430] describes the major and minor threats to MPTCP. Due to the limitations of space, we explain only three of them.

3.1. Eavesdropper in Initial Handshake

The attacker could eavesdrop both MPTCP keys in an initial three-way handshake. This threat is mentioned in [RFC7430], and is considered acceptable. In MPTCP, the valid user is the one who has a shared key from an initial handshake. An eavesdropper to the initial handshake also has the same authority. Reference [I-D.ietf-paasch-mptcp-ssl][I-D.ietf-bagnulo-mptcp-secure][I-D.ietf-bittau-tcp-crypt][Sec-MPTCP-con-approach] describe possible solutions.

An eavesdropper in initial handshake is the most powerful attacker model in MPTCP. An active attacker in the initial handshake is out of the scope of this paper. The initial handshake is a three or four-way handshake in TCP. Modifying this connection is a problem of

TCP, not MPTCP. Threats in MPTCP should arise due to the additional operations of MPTCP which are secure in TCP. The integrity of the initial handshake should be guaranteed.

3.2. DoS Attack on MP_JOIN

A valid token in SYN+MP_JOIN makes the host turn into a receiving state. The host stores two 32-bit random nonces for verifying HMAC. If the attacker does not respond to the third ACK of a three-way handshake, the host maintains the half-open state until the third ACK is received. The number of half-open connections per MPTCP session is limited.

The attacker simply sends multiple MP_JOINs with different four-tuples, evading the limitation of half-open connections to exhaust the resource. The attacker only needs the valid token which is easily achieved, as the token is sent as plaintext, because the token is not to provide security, but to specify the MPTCP session. A partial-time on-path eavesdropper inspecting any one of a MP_JOIN three-way handshake can perform a DoS Attack on MP_JOIN with a valid token.

3.3. ADD_ADDR Attack

The ADD_ADDR attack is a MPTCP session hijacking using a man-in-the-middle (MitM) attack. An off-path active attacker can perform an ADD_ADDR attack. The attacker creates MitM configuration using the ADD_ADDR option, even if she is not in the middle of the path between the hosts. To prevent this, ADD_ADDR format is modified to include HMAC. However, it is still vulnerable to an eavesdropper in the initial handshake. First, we describe the attack for the previous ADD_ADDR format. We then look at the threats of the modified format.

Assume that hosts-A and -B have the secure MPTCP session. The attacker wants to add a subflow to host-A. The attacker sends her IP address and Address ID to host-B, using the ADD_ADDR option. Host-B considers it as the advertisement of a redundant IP address from host-A, and tries to begin an MP_JOIN handshake to the attacker’s IP address.

Host-B is a valid user who can make the valid token for A, Token-A. Host-B sends Token-A and a random value, R-B to the attacker and she relay these values to host-A. Host-A verifies Token_A then sends HMAC-B and R-A to the attacker. The attacker delivers these values to host-B. Finally, host-B sends HMAC-A to the attacker. The attacker could finalize the authentication using HMAC-A.

The ADD_ADDR attack is a typical MitM attack except that the attacker could launch the attack whenever she wants. The connection requests could be refused when Address ID in the received ADD_ADDR collides with that already assigned in the subflows. However, the collision could be ignored, considering that the default number of the subflow in the current kernel is two, and that subflows are finite due to the lack of network interfaces in the normal network configuration.

The root cause of an ADD_ADDR attack is that there are no authentication values for ADD_ADDR operation allowing the attacker to masquerade as hosts-A or -B. [I-D.ietf-mptcp-rfc6824bis] modifies this to only legitimate users being able to advertise their IP address using truncated HMAC. The parameters for HMAC are defined in Section II.C. However, an eavesdropper in the initial handshake generates a truncated HMAC using both keys and still launches an ADD_ADDR attack. Even then, that attacker could calculate the valid token and HMAC. Using these values, she constructs the MitM configuration or adds a subflow to the victims.

3.4. Design Consideration

Considering the widespread nature of TCP, it is hard to use PKIX [RFC5280], which has scalability issues. Even though it is possible, it has limited advantages because not all users have trusted certificates. It is not practical to use trusted third parties. MPTCP is based on weak authentication [Weak-auth]. The weak authentication is cryptographically strong authentication among unknown parties without trusted third parties. It does not authorize the hosts' real identity such as X. 509 certificates, since there is no trusted third party, and pre-shared secrets cannot be used.

The other host is unknown before establishing a connection. MPTCP should exchange the secrets in the initial handshake. Due to the leap of faith, which is one of the techniques supporting weak authentication, it cannot validate the actual credentials of entities, but ensures that entities are those who communicate from the beginning. For example, hosts-A and -B are valid users who have a MPTCP session. When host-B want to create a new subflow, hosts-A and -B authenticate each other with Key-A and Key-B, not using the real information of the hosts. Assuming that the key exchange is secure, the entities who have both keys are the valid users. The hosts cannot know if the other entities are hosts-A or -B, but they ensure that the other hosts are legitimate entities. However, the key exchange proceeds in plaintext. An eavesdropper in the initial subflow knows both keys, and this means that she is a valid user. Before the initial handshake, hosts-A and -B don't know each other. It is difficult to send the key securely between unknown parties.

3.4.1. Asymmetric Key Exchange

If using the asymmetric property, the key exchange could occur without key exposure between the unknown parties. There are two challenges to adopting an asymmetric key in MPTCP. The former is the space limitation of the TCP option and the latter is the cost of asymmetric computation. MPTCP is over the TCP option. The maximum length of TCP option is 40 bytes and the MPTCP header uses four bytes. Asymmetric key exchange is hard to implement only using the TCP option without using TCP payload. It generally needs a large space for trading cryptographic parameters. However, a SYN flagged packet typically does not include the data for negotiating the initial sequence number. At least two packets in TCP handshake could not be used for sending data, which results in extra packets for trading public parameters. Despite space and time overheads, this concept was used in the prototypes of securing MPTCP [SecEval-MPTCP] and SMPTCP [I-D.ietf-bagnulo-mptcp-secure] to cover an eavesdropper in initial handshake. They deal with additional packets in an initial handshake for key exchange.

3.4.2. Minimizing Initial Handshake

The short connection of MPTCP subflow degrades the overall TCP performance [Shortflow]. Not every MPTCP session transfers a large amount of data. Some of them are terminated right after or before subflow is established. When a short connection occurs, the operation of adding subflow reduces the TCP performance since it makes an unnecessary connection. However, a transport layer cannot estimate the volume of application data. It is difficult to predict the necessity of subflow before making the connection. Delaying the point of creating subflow reduces the damage of short connection problem. Only the connection with long lifetime wants to make a new subflow. But an initial handshake is inevitable. The overhead of the initial handshake has a critical impact on the whole network since it occurs each connection. To minimize the handshake, the current implementation exchanges keys in plaintext, even though these are vulnerable to an eavesdropper in initial handshake.

4. The Proposed Design

Previous methods using an asymmetric key increase the overhead of the initial handshake resulting from the additional packet. This breaches the latter design consideration. We minimize public parameters for an asymmetric key. Optimized parameters are able to be embedded in the TCP option, and don’t require additional packets, except for a four-way handshake. Considering SSL/TLS, the public information is too large to be in the TCP option. MPTCP relies on weak authentication, which doesn’t care about other host’s real identity. Our scheme skips the exchange of certificates. It cannot guarantee publicity of the asymmetric key, but authenticates the subflows that originate from the owner of the MPTCP session. Another challenge is the size of the public key. To reduce the key size, we apply the Elliptic Curve and Elliptic Curve Diffie-Hellman [RFC4492].

```
                 Host A                                    Host B
        ------------------------                      -------------
        Address A1 | Address A2                         Address B1
        ------------------------                      -------------
            |               |                                 |
            |               |                                 |
            |          SYN + MP_CAPABLE(A’s x point)          |
       -----|------------------------------------------------>|
       |    |          ACK + MP_CAPABLE(B’s x point)          |
       |    |------------------------------------------------>|
       A    |          SYN + MP_CAPABLE(A’s y point)          |
       |    |------------------------------------------------>|
       |    |          SYN + MP_CAPABLE(B’s y point)          |
       -----|------------------------------------------------>|
            |               |                                 |
            |             SYN + MP_JOIN(Token-B, HMAC-token, R-A)
            |               |-------------------------------->|-----
            |               |  SYN/ACK + MP_JOIN(Auth-B, R-B) |    |
            |               |<--------------------------------|    |
            |               |       ACK + MP_JOIN(Auth-A)     |    B
            |               |-------------------------------->|    |
            |               |                ACK              |    |
            |               |<--------------------------------|-----
            |               |                                 |
```

Figure 1: Basic operation of the proposed Multipath TCP

Notations	Value
K	Hash(X_AB||Y_AB)
Token_B	lsb_32(Hash(X_B||Y_B))
HMAC_Token	lsb_32(HMAC(K, Token_B||Address ID||R_A))
Auth_B	msb_64(HMAC(K, R_B||R_A))
Auth_A	HMAC(K, R_A||R_B)

Figure 2: Parameter Notations and Thier Values for the Proposed MPTCP Scheme

4.1. New MP_CAPABLE handshake

Fig.1.A describes the sequence of a modified handshake. Parameters of the Elliptic Curve use the named curve defined in [SEC2]. The length of the x point and y point relates to the type of elliptic curve. The modified MP_CAPABLE needs a four-way handshake. First, Host-A sends SYN with A’s x point and stuffing the one of unused bits in MP_CAPABLE option. Host-B responds with ACK including B’s x point. Host-B sends SYN containing B’s y point. Finally, Host-A responds with ACK with A’s y point.

```
                       1                   2                   3
   0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
  +---------------+---------------+-------+-------+---------------+
  |     Kind      |    Length     |Subtype|Version|A|B|C|D|E|F|G|H|
  +---------------+---------------+-------+-------+---------------+
  |   EC type     |                                               |
  +---------------+                                               |
  |             Sender or Receiver’s x or y point in E            |
  |               (Length is depending on EC type)                |
  +---------------------------------------------------------------+
```

Figure 3: New MP_CAPABLE option

Fig.3. shows the format of the new MP_CAPABLE. The current implementation uses "A" and "H" flags and reserves "B" flag for an extension. "C"-"G" flags remain for cryptographic negotiation. Our design selects the flag among them. The proposed design supports backward compatibility. It requests a connection initiation set to an unused flag. Receivers who do not support our scheme reject the connection, since our request uses an unused flag. It simply returns to current implementation, which uses "H" flags. Receivers who support our scheme but do not want to use asymmetric key exchange reply that "H" flag will be used with their key, Key-B. Receivers do not drop the request packet, to avoid repetition of connection initiation. Key-A is the least significant 64-bits of sender’s x point in the request packet. The randomness of Key-A is ensured, because x point is also arbitrary value.

EC type	Named Curved	RSA/DSA
0	secp160k1	1024
1	secp160r1	1024
2	secp160r2	1024
3	secp192k1	1024
4	secp192r1	1024
5	secp224k1	2048
6	secp224r1	2048
7	secp256k1	3072
8	secp256r1	3072
Reserved	Reserved	Reserved

Figure 4: Supported Elliptic Curve Type and Security Level Compared to RSA/DSA

4.2. New MP_JOIN handshake

A 32-bit token identifies an MPTCP session where a new subflow wants to join in. Assume that a sender inserts a random value in a token to defend against reuse of the token. It is problematic for a receiver to distinguish the requested MPTCP session. The receiver generates hash values of all stored MPTCP identifiers with a random value to compare with the token. This degrades overall TCP performance in proportion to the currently existed MPTCP sessions. To solve this problem, the proposed design sends the token in plaintext for clarity. It protects the token using HMAC whose messages are the token, Address ID, and random value. Although an attacker knows the valid token, she could not launch the attack, since calculating HMAC for a different Address ID is impossible. In the case of reusing a previously delivered HMAC, the connection requests are refused, due to the collision of Address IDs.

Fig.1.B describes the sequence of the new MP_JOIN handshake. Fig.2 describes details of the parameters. Using (X-A, Y-A) and (X-B, Y-B), both hosts calculate (X-AB, Y-AB) with Elliptic Curve Diffie-Hellman key exchange. Then, they calculate the Token-B and K. These computations could be pre-processed. Host-A sends SYN with Token-B, HMAC-Token, and a random value, R-A, in MP_JOIN. Host-B verifies HMAC-Token, and checks that Address ID has no collision. Host-B sends SYN/ACK with Auth-B, which originates from R-B, R-A, X-AB, and Y-AB. Only a legitimate user who has the pre-shared secret, (X-AB, Y-AB), can make the right authentication values. The responses ACK with Auth-A are made by R-A, R-B, X-AB, and Y-AB.

4.3. ADD_ADDR

Assuming the ADD_ADDR operation is vulnerable, even in the proposed design, the attacker creates a subflow using the same method described in Section III.B without knowing the shared key. The current MPTCP denies the requests when the sender’s IP address is different from the IP address, a component of HMAC. But, an eavesdropper in initial handshake who knows both keys still derives a new HMAC with her IP address as an input. In the proposed design, the attacker could not acquire the shared key. Maintaining current ADD_ADDR format mitigates against ADD_ADDR attack.

5. Evaluation

This section evaluates the proposed design compared to the previous defense technique described in Section VI. MPTLS and SMPTCP calculate the shared key for authentication right after a key exchange over the initial handshake. Calculating the shared key occurs whenever an MPTCP session is established, causing the increase of overall overhead. This calculation violates our design consideration, of minimizing the initial handshake. The proposed design exchanges public keys in the initial handshake, but derives a shared key in adding subflows, to decrease the computational overhead of the whole network. In the case of a short connection, it does not calculate a shared key, since MP_JOIN does not arise. Our scheme optimizes not only the computational but also the space and time overheads, through MPTCP specific design.

	Proposed Design	SMPTCP	MPTLS	Hash Chain	MPTCP
MP_CAPABLE					
- Key exchange(bytes)	148	202	7468	52	32
- Number of RTT/2	3	4	7	3	3
MP_JOIN					
- Identify MPTCP session(bytes)	16	12	12	24	12
- Authentication(bytes)	40	40	40	28	40
Eavesdropper in initial handshake					
& Off-path attacker in subflows	O	O	O	O	X
& On-path eavesdropper in subflows	O	O	O	O	X
& On-path active attacker in subflows	O	O	O	X	X
DoS Attack on MP_JOIN	O	X	X	X	X
ADD_ADDR Attack					
& Eavesdropper in initial handshake	O	O	O	X	X
& On-path any attacker in subflows	O	O	O	X	O
Data encryption	O	O	O	X	X

Figure 5: Comparison of the proposed design and previous MPTCP schemes in terms of space overhead(bytes), time overhead(RTT), security, and data encryption

Fig.5 outlines our evaluation. We explain the overhead of the proposed design and then discuss the security aspect. Asymmetric methods have a high space overhead represented by bytes, due to the size of public information. Each method has a different handshake of packets for key exchange. We adopt an expression as a notation, rather than using total bytes to declare this characteristic. The operands of addition are the size of each packet, except the TCP header. The proposed scheme has the lowest space overhead in MP_CAPABLE among asymmetric schemes. To cover DoS attack on MP_JOIN, it includes HMAC of token causing a relatively big overhead caused by identifying the MPTCP session. The time overhead represents the number of RTT/2 which means the one-way message latency. Although it needs a four-way handshake on MP_CAPABLE, the number of RTT/2 is three, since the second ACK packet and third SYN packet can pass concurrently. MPTLS has a large overhead of space and time depending on the TLS handshake. The number of RTT/2 of MP_JOIN is the same as three in every scheme, so we intentionally omit this outcome in Fig.5.

Asymmetric methods are secure against an eavesdropper in initial handshake. Key exchange without key exposure makes data encryption possible. Hash Chain is also a research into the same security threats, but that scheme is insecure to the on-path active attacker in subflow. She drops the MP_JOIN requests of legitimate users and then makes her MP_JOIN request using the hash value received from the legitimate user. Hash Chain has no mitigation for an ADD_ADDR attack. It authenticates hosts using a hash chain, so there are no comments about the HMAC and its keys. If it simply uses a stored hash as a key of HMAC, the exchange of hash values has the same meaning as the exchange of keys in plaintext. It is still insecure to ADD_ADDR attack towards an eavesdropper in initial handshake. But if it uses the ADD_ADDR format of the current MPTCP with the assumption that the hash value is a key, it would be changed to "secure" towards an on-path active attacker in Fig.5. A notable difference is DoS Attack on MP_JOIN. In other methods, the attacker can undertake a DoS attack using a valid token. However, in the proposed design, the attacker knows a valid token but she could not make HMAC due to ignorance of the shared key. If the attacker reuses HMAC, rather than making a new one, the receiver denies the connection, by checking the collision of address IDs.

6. Related Work

We discuss previous work for the secure schemes on security threats mentioned in Section III. MPTLS [I-D.ietf-paasch-mptcp-ssl] uses an asymmetric key to avoid the key exposure caused by key exchange in plaintext. Hosts negotiate the shared key for HMAC using TLS. TLS authenticates both hosts with certificates and operates the key exchange algorithm to create the shared key. MPTCP operations are performed with this key. However, MPTLS inherit the overhead of TLS handshake.

SMPTCP is another method that uses an asymmetric key. It uses tcpcrypt [I-D.ietf-bittau-tcp-crypt] to secure an MPTCP session. Using tcpcrypt, both hosts negotiate a cryptographic protocol that protects the TCP payload. A shared key calculated by the negotiated cryptographic protocol is used for authentication for MP_JOIN. Tcpcrypt uses the TCP option for implementation so it is easy to integrate with MPTCP. Due to restrictions of the TCP option size, tcpcrypt requires one additional message to perform the key exchange. Despite one-way message latency, tcpcrypt is much more efficient than TLS, since it focuses on the key exchange. Likewise MPTLS, operations in SMPTCP perform the same as MPTCP, except the key for HMAC is determined by tcpcrypt. Tcpcrypt is vulnerable in MitM attack, but MitM in the initial handshake is out of the scope of this paper.

The Hash Chain-based solution [Sec-MPTCP-con-approach] makes a list consisting of chained hash values generated by recursively executing a hash function. The host makes the key list, H0-Hn by repeating the hash function with the initial random value as a message until pre-defined length, n, of the hash chain. During the initial handshake of the MPTCP session, both hosts exchange their last hash values Hn. During adding subflow, each host sends the next value of their previous hash values, i.e., Hn-1. The one-way property of the hash function blocks the attacker from gaining the previous hash values. Only legitimate hosts know the full hash chain. Next adding subflow authenticates both hosts using the hash chain in reverse order. Once the subflow is established, the host replaces the stored hash with the last received hash. However, an active attacker could drop the SYN+MP_JOIN from the legitimate host, and establish a new subflow using a hash value in that SYN packet, without knowing the hash chain.

7. IANA Considerations

This document requests an MPTCP unused flag for this option:

o Asymmetric Key Exchange Option

NOTE: Implementations may use "e" flag among unused flags

8. References

8.1. Normative References

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <http://www.rfc-editor.org/info/rfc2119>.

[RFC4492] Blake-Wilson, S., Bolyard, N., Gupta, V., Hawk, C., and B. Moelier, "Elliptic Curve Cryptography (ECC) Cipher Suites for Transport Layer Security (TLS)", RFC 4492, DOI 10.17487/RFC4492, May 2006, <http://www.rfc-editor.org/info/rfc4492>.

[RFC5280] Cooper, D., Santesson, S., Farrell, S., Boeyen, S., Housley, R., and W. Polk, "Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List(CRL) Profile", RFC 5280, DOI 10.17487/RFC5280, May 2008, <http://www.rfc-editor.org/info/rfc5280>.

[RFC6824] Ford, A., Raiciu, C., Handley, M., and O. Bonaventure, "TCP Extensions for Multipath Operation with Multiple Addresses", RFC 6824, DOI 10.17487/RFC6824, January 2013, <http://www.rfc-editor.org/info/rfc6824>.

8.2. Informative References

[RFC6181] Bagnulo, M., "Threat Analysis for TCP Extensions for Multipath Operation with Multiple Addresses", RFC 6181, DOI 10.17487/RFC6181, March 2021, <http://www.rfc-editor.org/info/rfc6181>.

[RFC7430] Bagnulo, M., Paasch, C., Gont, F., Bonaventure, O., and C. Raiciu, "Analysis Residual Threats and Possible Fixes for Multipath TCP (MPTCP)", RFC 7430, DOI 10.17487/RFC7430, July 2015, <http://www.rfc-editor.org/info/rfc7430>.

[I-D.ietf-mptcp-rfc6824bis]
Ford, A., Raiciu, C., Handley, M., Bonaventure, O., and C. Paasch, "TCP Extensions for Multipath Operation with Multiple Addresses", draft-ietf-mptcp-rfc6824bis-05 (work in progress), January 2016.

[I-D.ietf-paasch-mptcp-ssl]
Paasch, C. and O. Bonaventure, "Securing the Multipath TCP handshake with external keys draft-paasch-mptcp-ssl-00", I-D.ietf-paasch-mptcp-ssl-00 (work in progress), October 2012, <https://tools.ietf.org/pdf/draft-paasch-mptcp-ssl-00.pdf>.

[I-D.ietf-bagnulo-mptcp-secure]
Bagnulo, M., "Secure MPTCP draft-bagnulo-mptcp-secure-00", I-D.ietf-bagnulo-mptcp-secure-00 (work in progress), February 2014, <https://tools.ietf.org/id/draft-bagnulo-mptcp-secure-00.txt>.

[I-D.ietf-bittau-tcp-crypt]
Bittau, A., Boneh, D., Hamburg, M., Handley, M., Mazieres, D., and Q. Slack, "Cryptographic protection of TCP Streams (tcpcrypt) draft-bittau-tcp-crypt-04.txt", I-D.ietf-bittau-tcp-crypt-04 (work in progress), February 2014, <https://tools.ietf.org/id/draft-bagnulo-mptcp-secure-00.txt>.

[SecEval-MPTCP]
Demaria, F., "Security Evaluation of Multipath TCP", M.S. thesis Computer Engineering, KTH Royal Institute of Technology, March 2016.

[Sec-MPTCP-con-approach]
Diez, J., Bagnulo, M., Valera, F., and I. Vidal, "Security for multipath TCP: a constructive approach", International Journal of Internet Protocol Technology Vol. 6. No. 3., 2011.

[Weak-auth]
Arkko, J. and P. Nikander, "Weak Authentication: How to Authentication Unknown Principals without Trusted Parties", International Workship on Security Protocols Springer Berlin Heidelberg, April 2002.

[Shortflow]
Kheirkhah, M., Wakeman, I., and G. Parisis, "Short vs. Long Flows: A Battle That Both Can Win", ACM SIGCOMM Computer Communication Review Vol. 45. No. 4., August 2015.

[SEC2] Certicom Research, , "SEC 2: Recommended Elliptic Curve Domain Parameters", SEC2 Version 1.0, September 2000, <http://www.secg.org/SEC2-Ver-1.0.pdf>.

Author’s Address

Dongyong Kim
Sungkyunkwan University
Suwon 16419
South Korea

Email: kdysk93@skku.edu