Efficient Design for Secure Multipath TCP against Eavesdropper in Initial Handshake

Dong-Yong Kim, Hyoung-Kee Choi

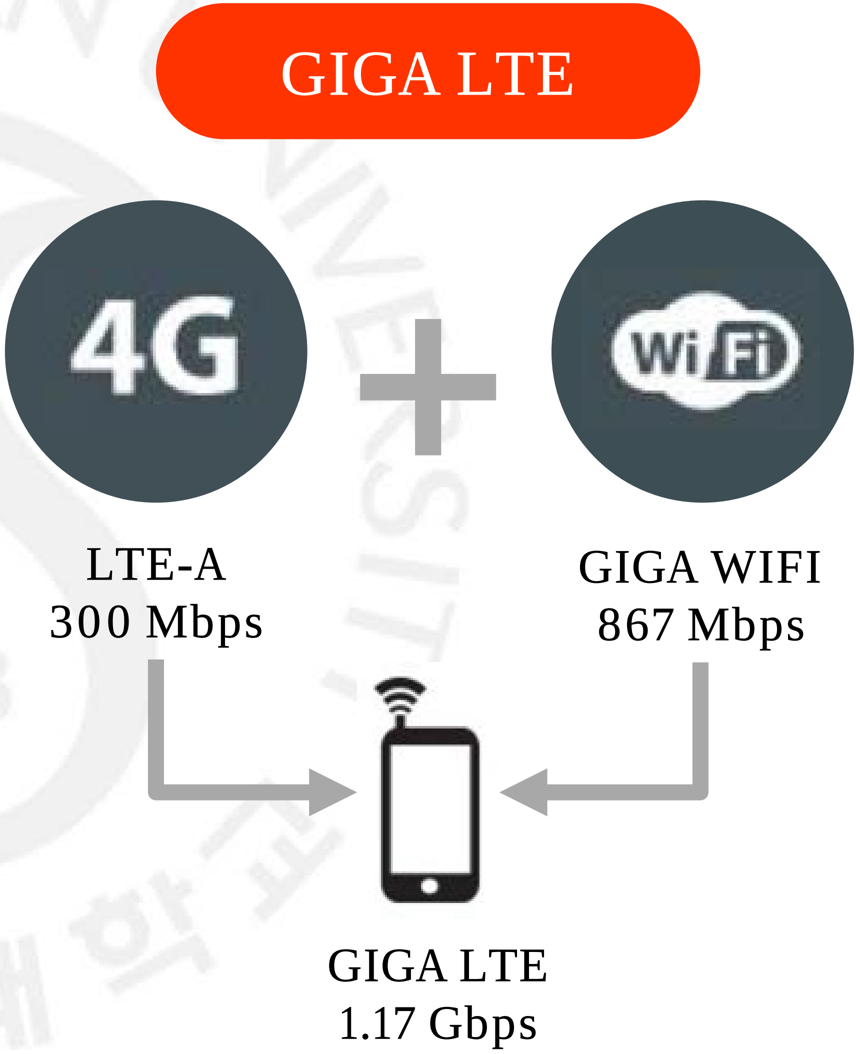
{kdysk93, meosery}@skku.edu

Sungkyunkwan University

IETF 97

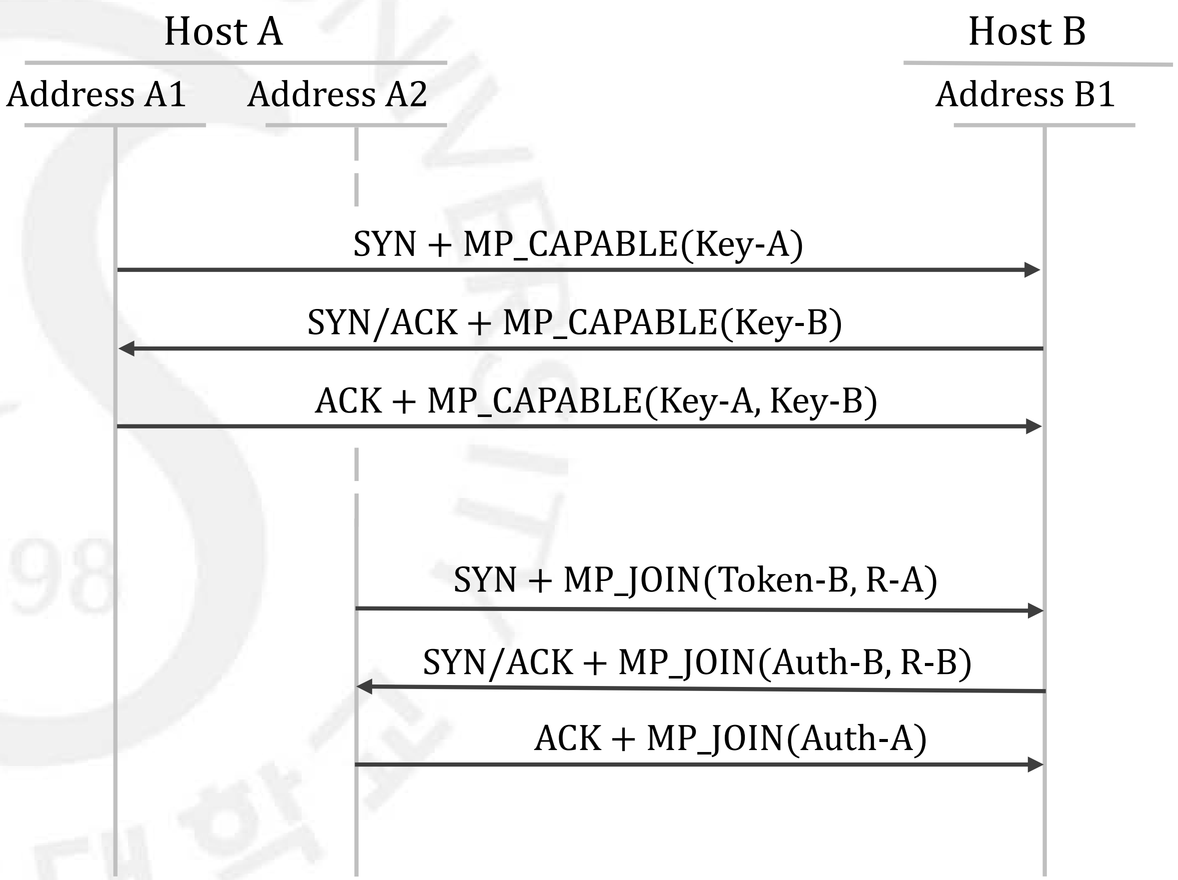
Introduction

- Despite the short history, Multipath TCP(MPTCP) prevails drastically
  - As MPTCP was deployed, security concerns increase
- There have been multiple attempts at verifications to security of MPTCP
  - Initial eavesdropper breaches the primary security goal of MPTCP
- We need new solution for initial eavesdropper!



MPTCP v0 Sequence

- Initial connection setup
  - Three-way handshake with MP_CAPABLE
  - Exchange 64 bit key(Key-A, Key-B)
- Adding subflow setup
  - Three-way handshake with MP_JOIN
  - Authentication through ‘Token’ and HMAC
    - Token : most significant 32 bits of SHA1 output with Key-B as a message of hash



Basic Assumption in MPTCP

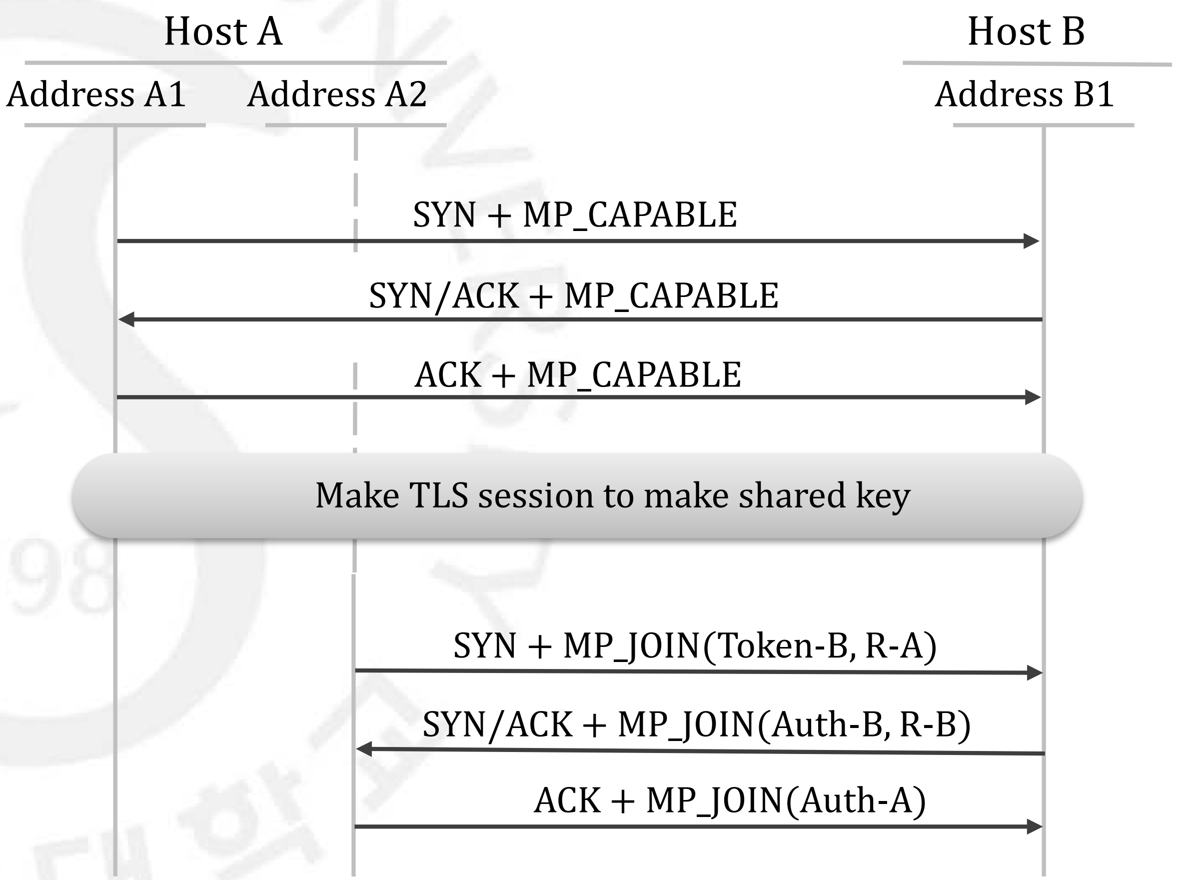
- Initial connected host should be a intended host
  - If other host? TCP Session hijacking, not MPTCP problem
- This assumption is good for using weak authentication
  - Weak authentication verify that corresponding host is the one who was in initial handshake
  - Weak authentication cannot guarantee corresponding host's identity
    - There are no CAs or trusted third parties

Threat Model

- Eavesdropper in the Initial Handshake(EitIH)
  - Has ability to eavesdrop shared keys in initial connection setup
  - Shared keys transmit in plaintext
- Related Solutions
  - Hash Chains
  - MPTCP-SSL(MPTLS)
    - TLS over the TCP
  - Tcpcrypt(SMPTCP)
    - RSA Diffie-Hellman Exchange through four additional packets after initial handshake
    - Encrypt IP datagram

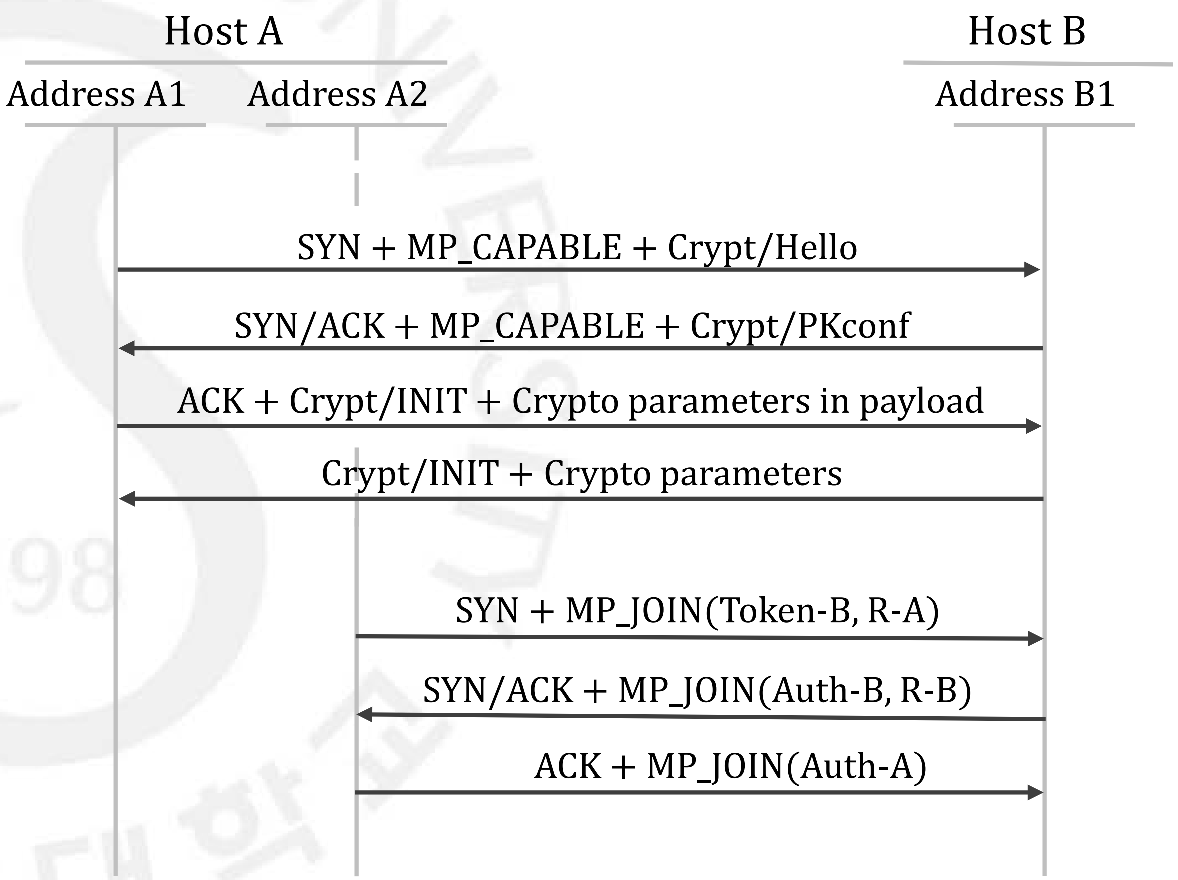
Related Solutions : MPTLS

- MPTLS makes shared key using TLS
- MPTLS needs TLS handshake
  - Inherit the overhead of TLS
- Additional one-way message delay
  - At least, four one-way delay for TLS



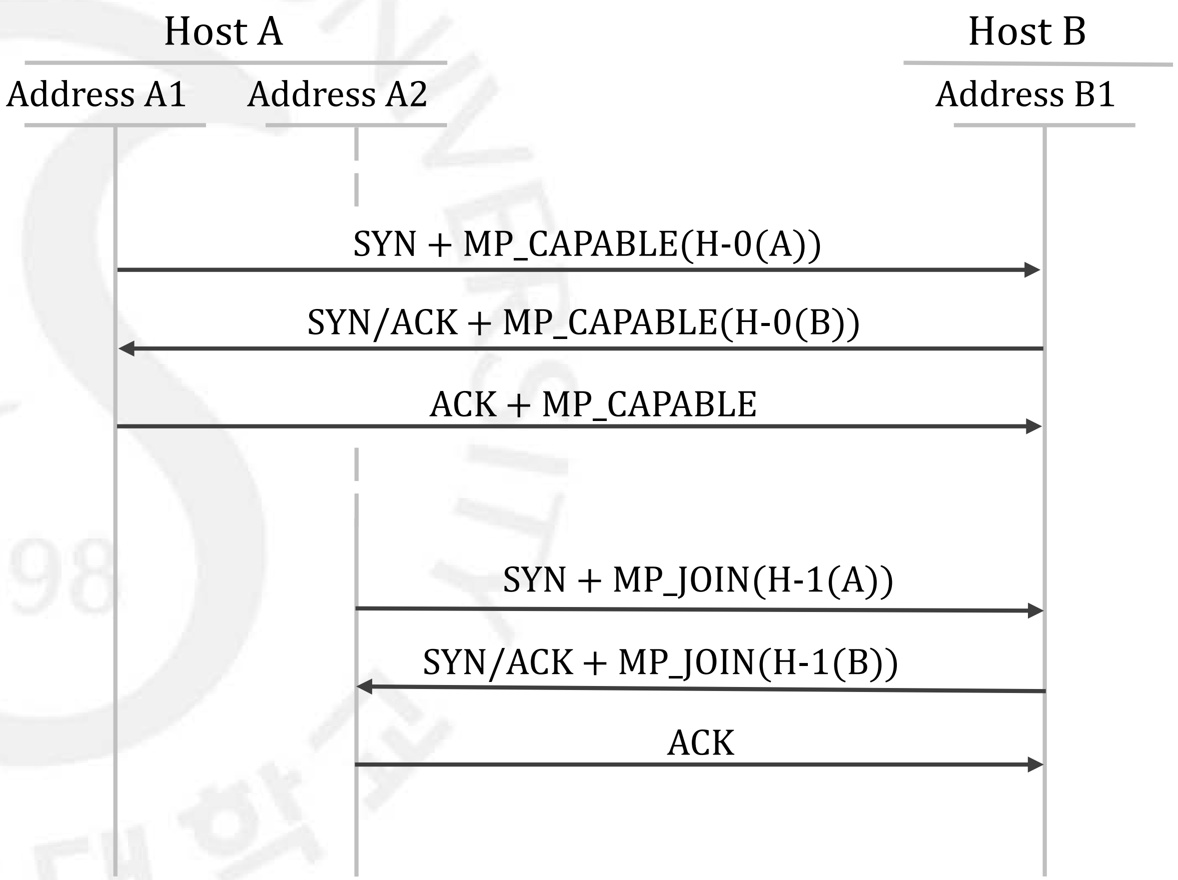
Related Solutions : SMPTCP

- SMPTCP uses Tcpcrypt
  - Tcpcrypt is low-overhead asymmetric key exchange protocol using TCP option
  - To reduce the overhead for TLS handshake
- SMPTCP still has overhead
  - Combine two protocol, double option header
  - One additional one-way delay



Related Solutions : Hash-Chain Based

- Each host makes hash-chain
  - H-0(A), H-1(A), … , H-n(A)
  - H-0(B), H-1(B), … , H-n(B)
  - H-n-1 = Hash (H-n)
- Reveal in reverse order
  - Based on one-way property of hash
- Weak authentication



Problem Definition

- In asymmetric manners, overhead is too big
  - Large space for TLS handshake or tcpcrypt
  - Additional one-way message delays
- In other methods, security is not enough
- We need more efficient and more secure protocol!

Pitfall of Integrity

- Only guarantee integrity of initial handshake
  - NOT provide confidentiality
  - Eavesdropper does not modify the connection
- Eavesdropper can get every information to bypass authentication

- Solution : Asymmetric key exchange!
  - If we can guarantee the corresponding host

Space Limitation

- MPTCP uses TCP option for compatibility to previous system
- TCP header length has maximum length of 60 bytes
- Except for basic header(20 byte), only for option? Maximum 40 byte
  - Except for option header(8 byte), only 32 bytes are available

- Solution : Parameter Minimizing(optimizing)!
  - To squeeze public parameters into a limited space

Short Connection Problem

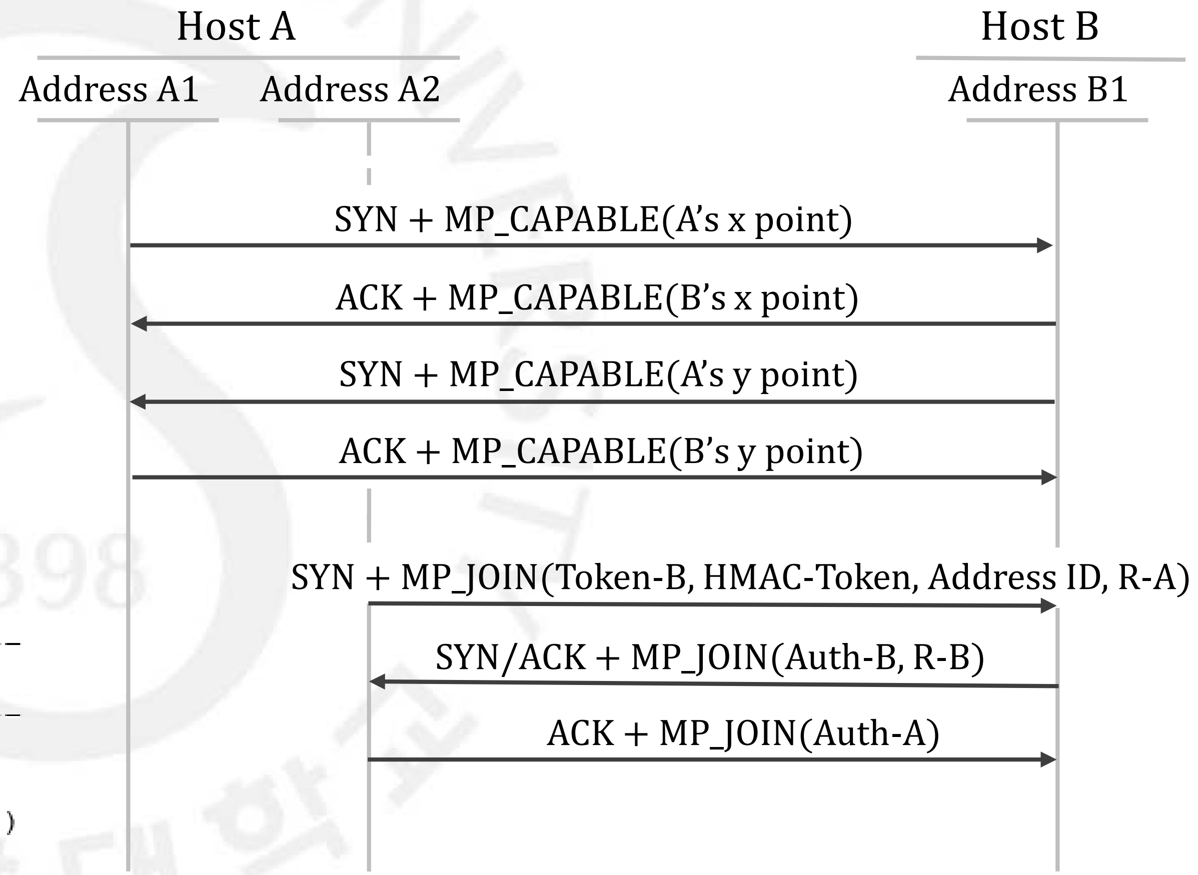
- In internet nature, there are a lot of short connection
  - Short connection : TCP connection which has short lifetime
- Short connections do not need subflows
  - Data are already sent through initial connection
- Establishing and terminating subflows cause the degrade of total throughput

- Solution : Delayed initiation
  - In case of short connection, MPTCP does not make subflows

Proposed Solution

- Use asymmetric key exchange
  - To block eavesdropper
- Minimize public parameters
  - Limit algorithm and key size
- Minimize overhead for Initial handshake
  - Can not distinguish which protocols will be used
  - Reduce overhead for short connection

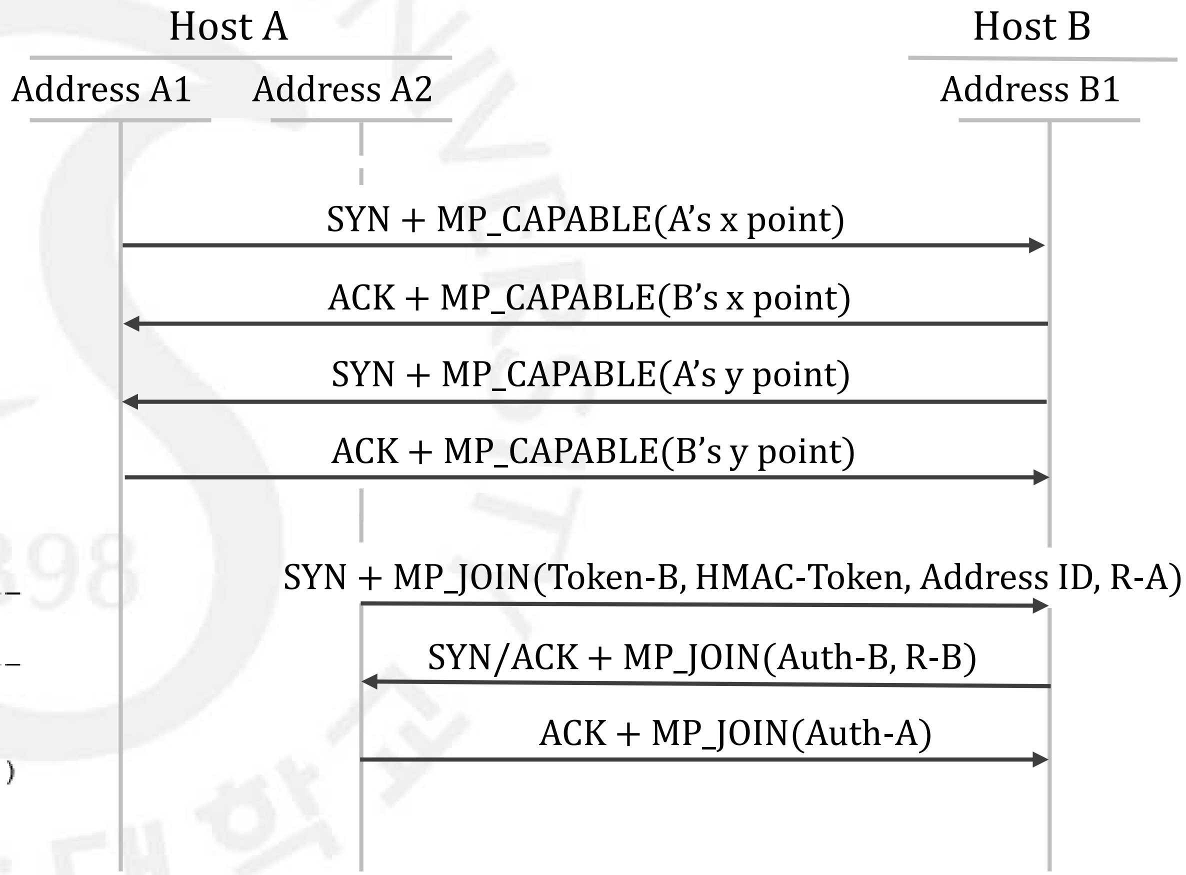
Notations	Value
K	Hash(X_AB||Y_AB)
Token_B	lsb_32(Hash(X_B||Y_B))
HMAC_Token	lsb_32(HMAC(K, Token_B||Address ID||R_A))
Auth_B	msb_64(HMAC(K, R_B||R_A))
Auth_A	HMAC(K, R_A||R_B)



DoS Attack on MP_JOIN

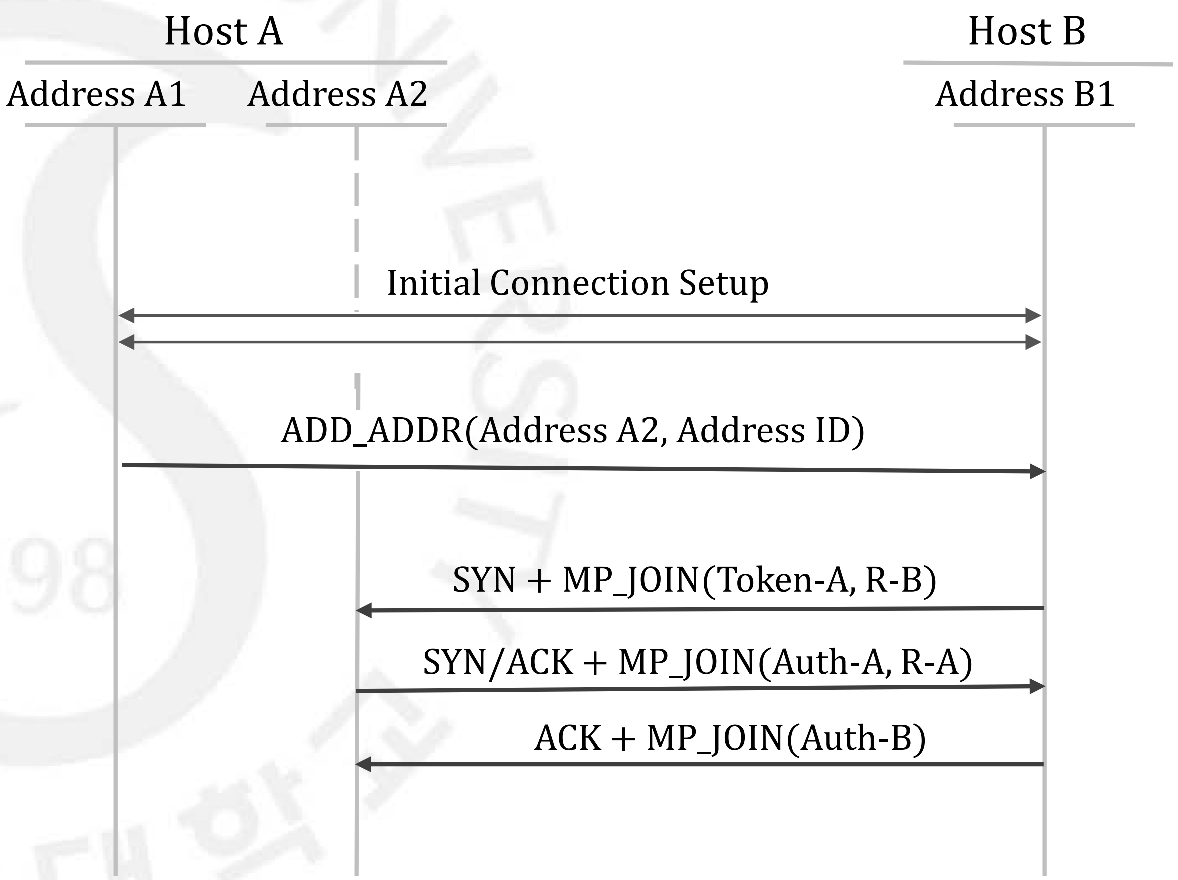
- Valid token in SYN+MP_JOIN makes the host turn into a receiving state
- Token reuses in same MPTCP session
  - Eavesdrop once, reuse until the end of session
  - Or, just guessing. It is only 32-bit
- Limited number of half-open state
  - Use different five tuple

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)



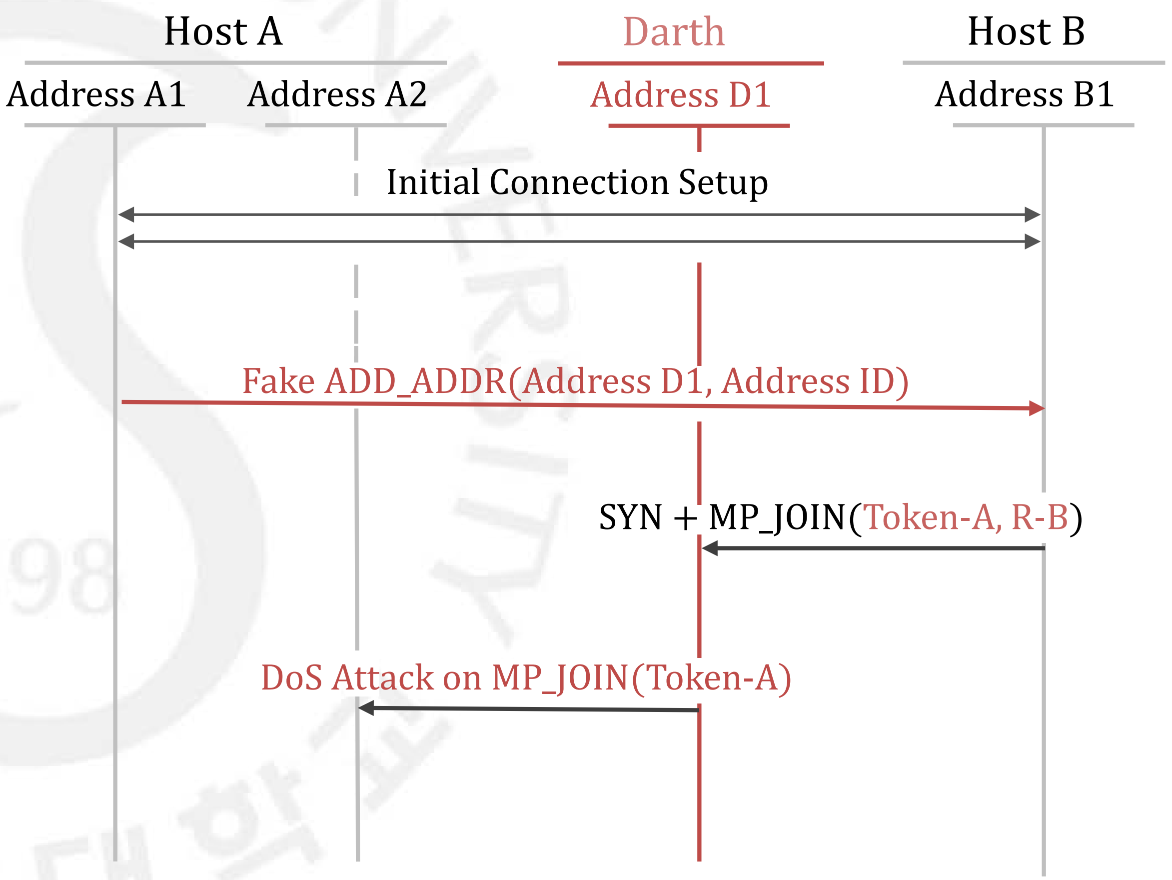
Address Advertisement in MPTCP v0

- Hosts may want to advertise their address
  - When host could not initiate the connection due to the middleboxes
- Address advertisement
  - Host A sends ADD_ADDR option with residue addresses(A2)
  - Host B initiates subflow handshake to A2
  - Finish subflow handshake



ADD_ADDR Attack

- Attacker sends fake ADD_ADDR
  - Host B sends Token-A, HMAC to Attacker
- DoS Attack on MP_JOIN
  - Attacker knows the valid token
- MPTCP v1 modifies ADD_ADDR
  - ADD_ADDR includes HMAC
  - However, EitIH knows the key for MAC



Data Encryption

- Asymmetric key exchange makes shared key without key exposure
  - Symmetric key exchange reveals shared key
- Using the shared key, data encryption is possible
- Two use cases of shared key
  - Only for authentication
  - Datagram encryption

Result

TABLE III. COMPARSION OF THE PROPOSED DESIGN AND PREVIOUS MPTCP SCHEMES IN TERMS OF SPACE OVERHEAD(BYTES), TIME OVERHEAD(RTT), SECURITY, AND DATA ENCRYPTION. IN THE CASE OF ASYMMETRIC SCHEMES, THE KEY EXCHANGE ALGORITHM IS ECDHE, AND THE MODULUS SIZE OF THE ELLIPTIC CURVE IS 256.

	Proposed Design	**SMPTCP [x]**	**MPTLS [x]**	**Hash Chain [x]**	**MPTCP**
MP_CAPABLE					
- Key exchange(*bytes*)	37+37+37+37	15+15+86+86	4+4+4+7456	24+24+4	8+16+8
- Number of RTT/2	3	4	3+4	3	3
MP_JOIN					
- Identify MPTCP session*(bytes)*	16	12	12	24	12
- Authentication*(bytes)*	16+24	16+24	16+24	24+4	16+24
Eavesdropper in initial handshake					
& Off-path attacker in subflow	Secure	Secure	Secure	Secure	Insecure
& On-path eavesdropper in subflow	Secure	Secure	Secure	Secure	Insecure
& On-path active attacker in subflow	Secure	Secure	Secure	Insecure	Insecure
DoS Attack on MP_JOIN	Secure	Insecure	Insecure	Insecure	Insecure
ADD_ADDR Attack					
& Eavesdropper in initial handshake	Secure	Secure	Secure	Insecure	Insecure
& On-path any attacker in subflow	Secure	Secure	Secure	Insecure	Secure
Data encryption	Possible	Possible	Possible	Impossible	Impossible

Conclusion

- Deliverable
  - Three design consideration
  - Secure Design for EitIH model
- Discussion
  - Scalability Issue - Limited algorithm and key size
  - Empirical evaluation
  - Omitted computation overhead
- Future Work
  - Real kernel implementation