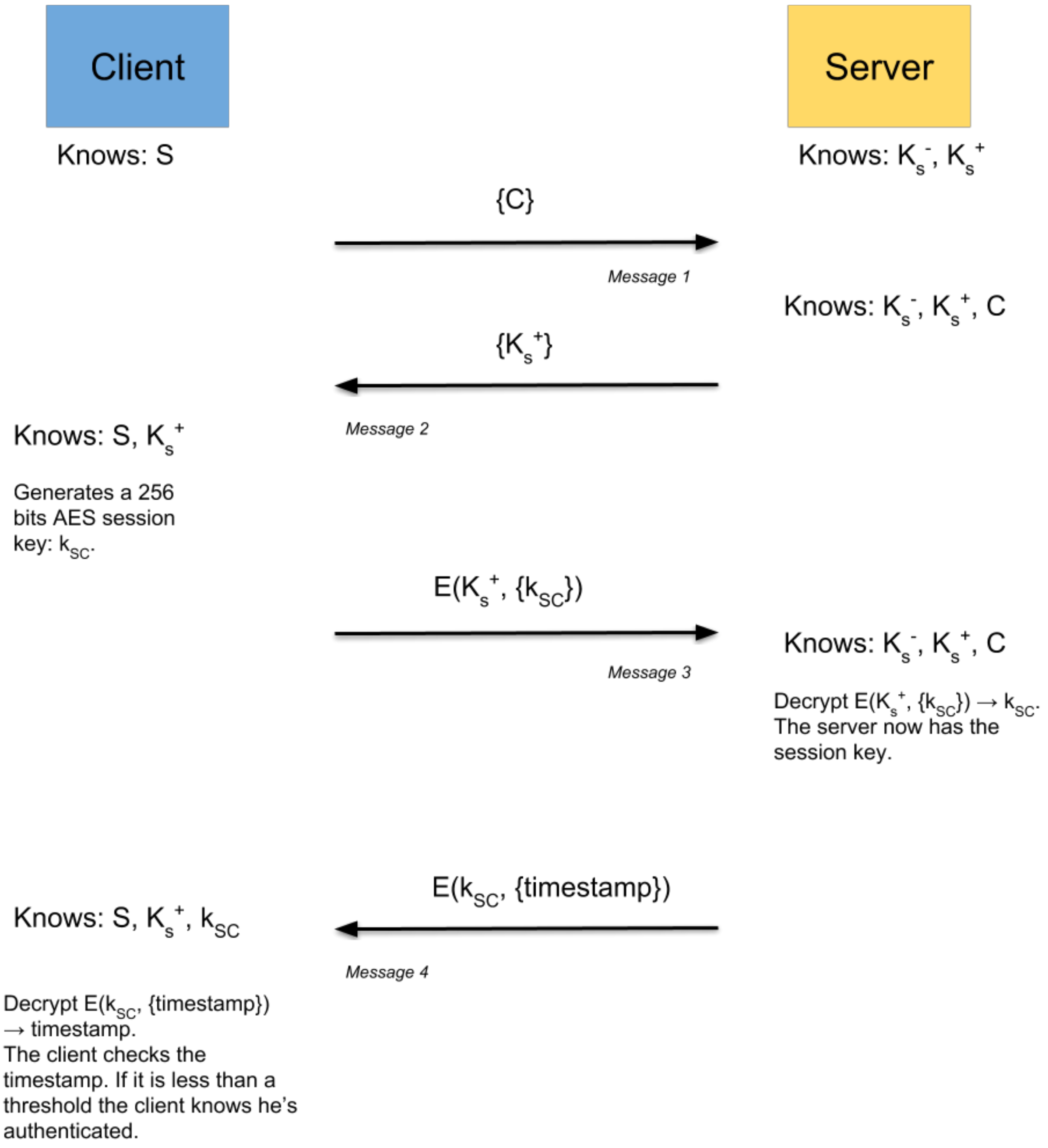
SSH Protocol – Initial version



Client

Knows: S

Server

Knows: K_s^-, K_s^+

{C}

Message 1

Knows: K_s^-, K_s^+, C

{K_s^+}

Message 2

Knows: S, K_s^+

Generates a 256 bits AES session key: k_{SC}.

E(K_s^+, {k_{SC}})

Message 3

Knows: K_s^-, K_s^+, C

Decrypt E(K_s^+, {k_{SC}}) → k_{SC}. The server now has the session key.

E(k_{SC}, {timestamp})

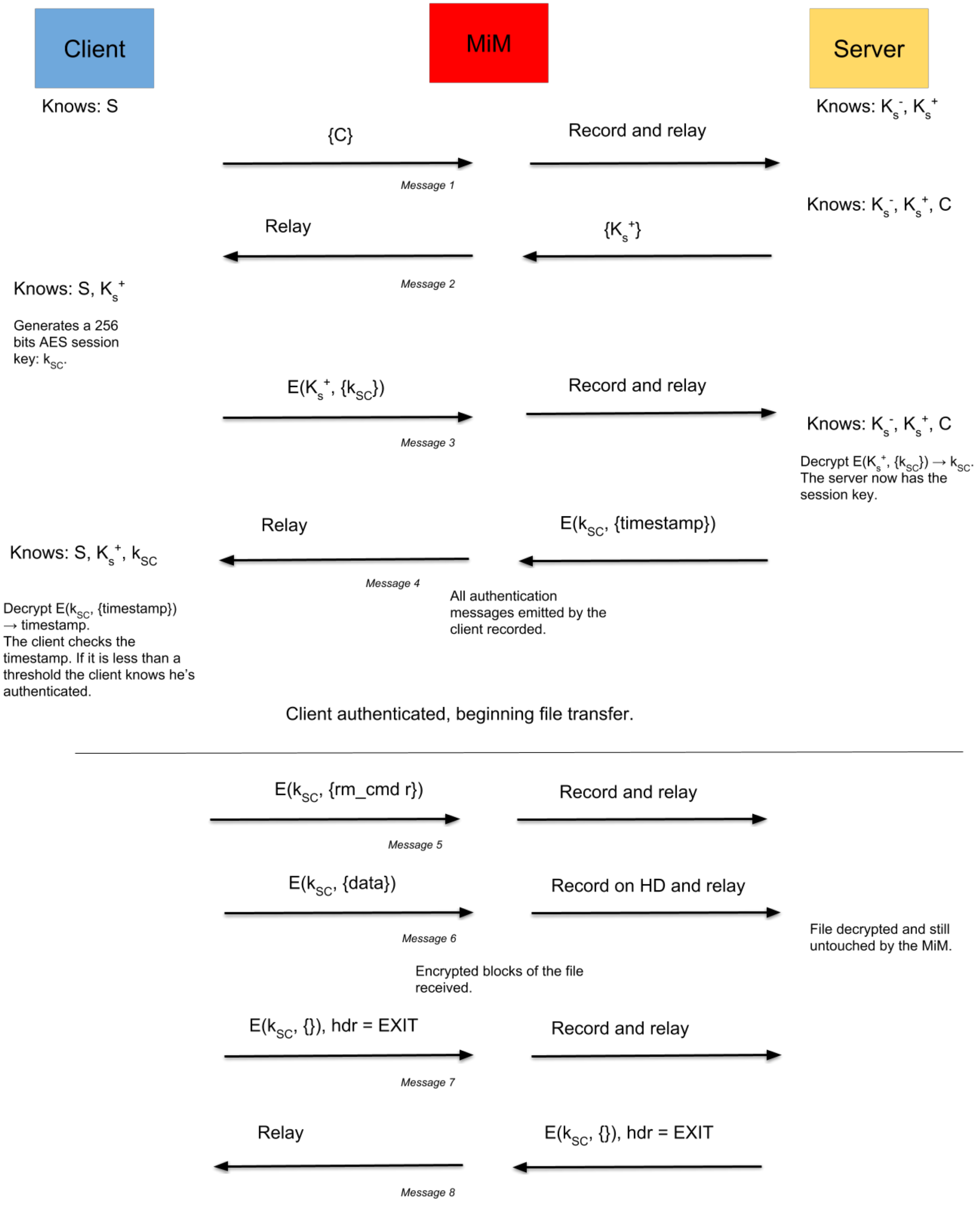
Message 4

Knows: S, K_s^+, k_{SC}

Decrypt E(k_{SC}, {timestamp}) → timestamp. The client checks the timestamp. If it is less than a threshold the client knows he's authenticated.

For message number four, the paper doesn't specify the kind of confirmation the server should send. I chose to send a timestamp, I also hesitated to send the session key encrypted with the session key… The timestamp sounded like a better option.

MiM Record and replay – Recording phase



Client

MiM

Server

Knows: S

Knows: K_s^-, K_s^+

{C}

Record and relay

Message 1

Knows: K_s^-, K_s^+, C

Relay

{K_s^+}

Message 2

Knows: S, K_s^+

Generates a 256 bits AES session key: k_{SC}.

E(K_s^+, {k_{SC}})

Record and relay

Message 3

Knows: K_s^-, K_s^+, C

Decrypt E(K_s^+, {k_{SC}}) → k_{SC}. The server now has the session key.

Relay

E(k_{SC}, {timestamp})

Knows: S, K_s^+, k_{SC}

Message 4

All authentication messages emitted by the client recorded.

Decrypt E(k_{SC}, {timestamp}) → timestamp. The client checks the timestamp. If it is less than a threshold the client knows he's authenticated.

Client authenticated, beginning file transfer.

E(k_{SC}, {rm_cmd r})

Record and relay

Message 5

E(k_{SC}, {data})

Record on HD and relay

Message 6

File decrypted and still untouched by the MiM.

Encrypted blocks of the file received.

E(k_{SC}, {}), hdr = EXIT

Record and relay

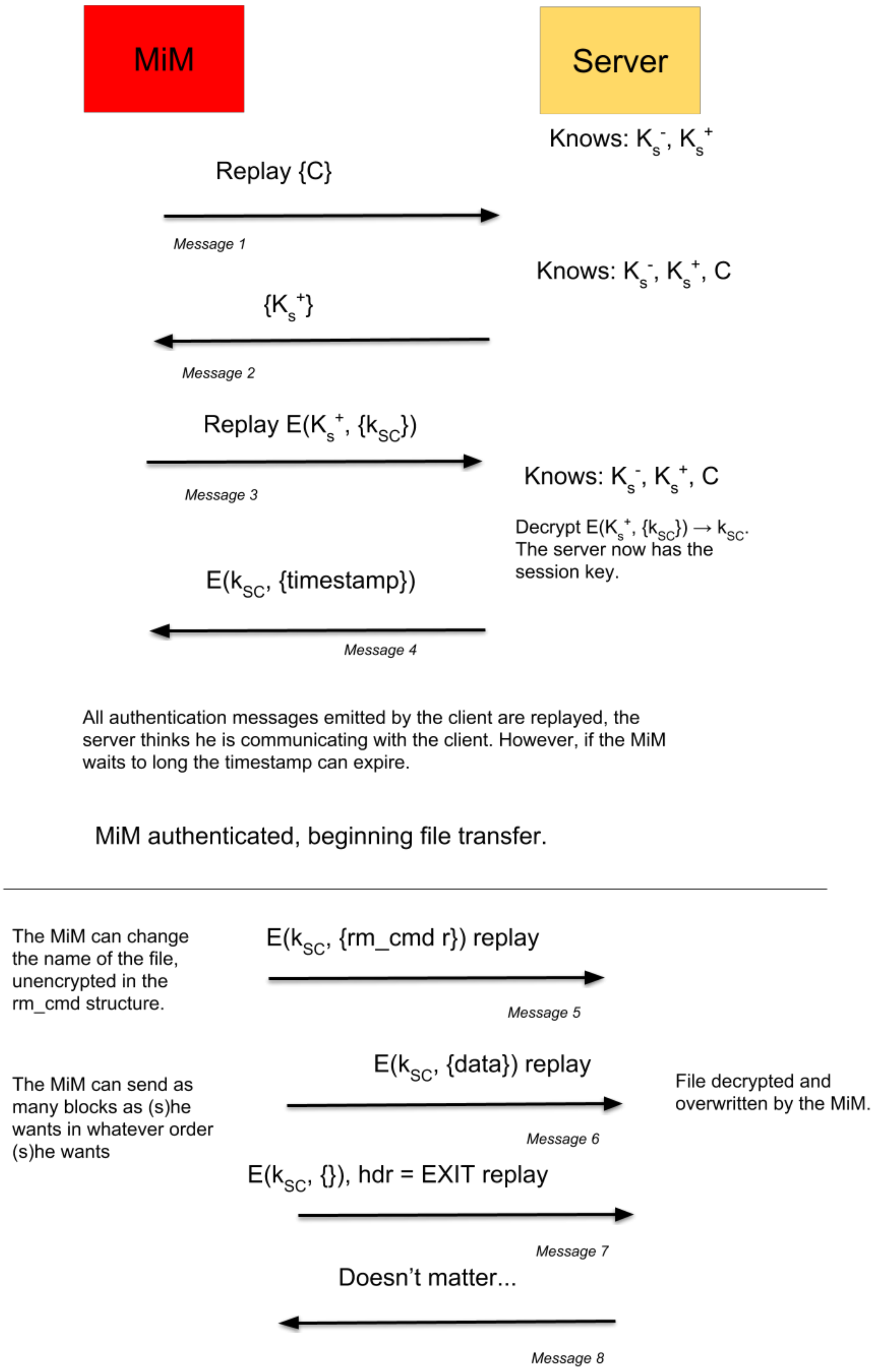
Message 7

Relay

E(k_{SC}, {}), hdr = EXIT

Message 8

MiM Record and replay – Replay phase



MiM

Server

Knows: K_s^-, K_s^+

Replay {C}

Message 1

Knows: K_s^-, K_s^+, C

{K_s^+}

Message 2

Replay E(K_s^+, {k_{SC}})

Message 3

Knows: K_s^-, K_s^+, C

Decrypt E(K_s^+, {k_{SC}}) → k_{SC}. The server now has the session key.

E(k_{SC}, {timestamp})

Message 4

All authentication messages emitted by the client are replayed, the server thinks he is communicating with the client. However, if the MiM waits to long the timestamp can expire.

MiM authenticated, beginning file transfer.

The MiM can change the name of the file, unencrypted in the rm_cmd structure.

E(k_{SC}, {rm_cmd r}) replay

Message 5

The MiM can send as many blocks as (s)he wants in whatever order (s)he wants

E(k_{SC}, {data}) replay

Message 6

File decrypted and overwritten by the MiM.

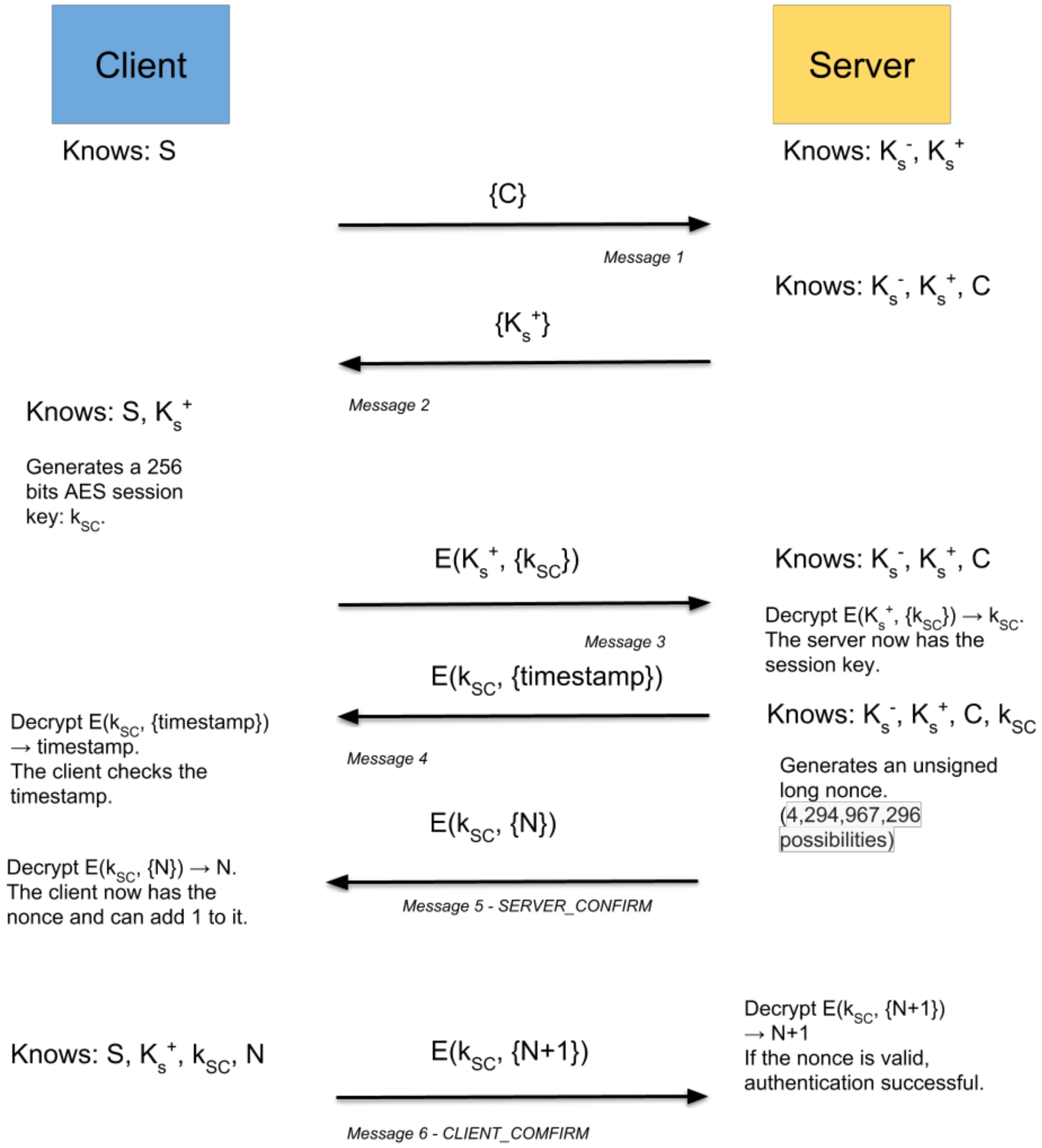
E(k_{SC}, {}), hdr = EXIT replay

Message 7

Doesn't matter...

Message 8

SSH Protocol – Replay resistant version



Generates a 256 bits AES session key: k_{SC}.

Decrypt E(K_s^+, {k_{SC}}) → k_{SC}. The server now has the session key.

Decrypt E(k_{SC}, {timestamp}) → timestamp. The client checks the timestamp.

Generates an unsigned long nonce. (4,294,967,296 possibilities)

Decrypt E(k_{SC}, {N}) → N. The client now has the nonce and can add 1 to it.

Decrypt E(k_{SC}, {N+1}) → N+1 If the nonce is valid, authentication successful.

This version counters the replay authentication because of the nonce. An adversary won't be able to replay message 6 because the nonce N is unique.

However, this doesn't solve the integrity issues. An adversary forwarding (not replaying) all the messages during the authentication can still modify the file sent by the client. The adversary records the file sent by the client, modifies the order of the blocks or the number of blocks sent and sent it to the server. The server doesn't check for the integrity. There should be a global file HMAC in addition of the tag check during the decryption.

VIE VALENTIN