Analyzing and fixing the QACCE security of QUIC

Yoshikazu Hanatani

(joint work with Hideki Sakurada, Kazuki Yoneyama, and Maki Yoshida)

Background

- TLS over TCP is most widely used transport security protocol, but its handshake has high latency.
- As an alternative, the QUIC protocol has been developed by Google, and some of the ideas have been incorporated to the TLS 1.3 draft.
- QUIC has been analyzed in some previous work: [Fischlin, Günther ’14] [Lychev et al. ’15] [Iseki, Fujisaki ’15]

This work

- Analyzes security of QUIC using the ProVerif automatic protocol verifier.
- The security model and the formalization of QUIC are based on [Lychev et al. ’15].
- Founds errors in the results of [Lychev et al. ’15].

QUIC

Transport security protocol developed by Google.

- For simplicity, omits server authentication and allows only restricted sets of ciphersuites
- For achieving low latency,
  - Uses UDP and omits TCP handshake.
  - Allows for a server DH value to be used in multiple sessions (for 0-RTT handshake) updated later for forward secrecy.

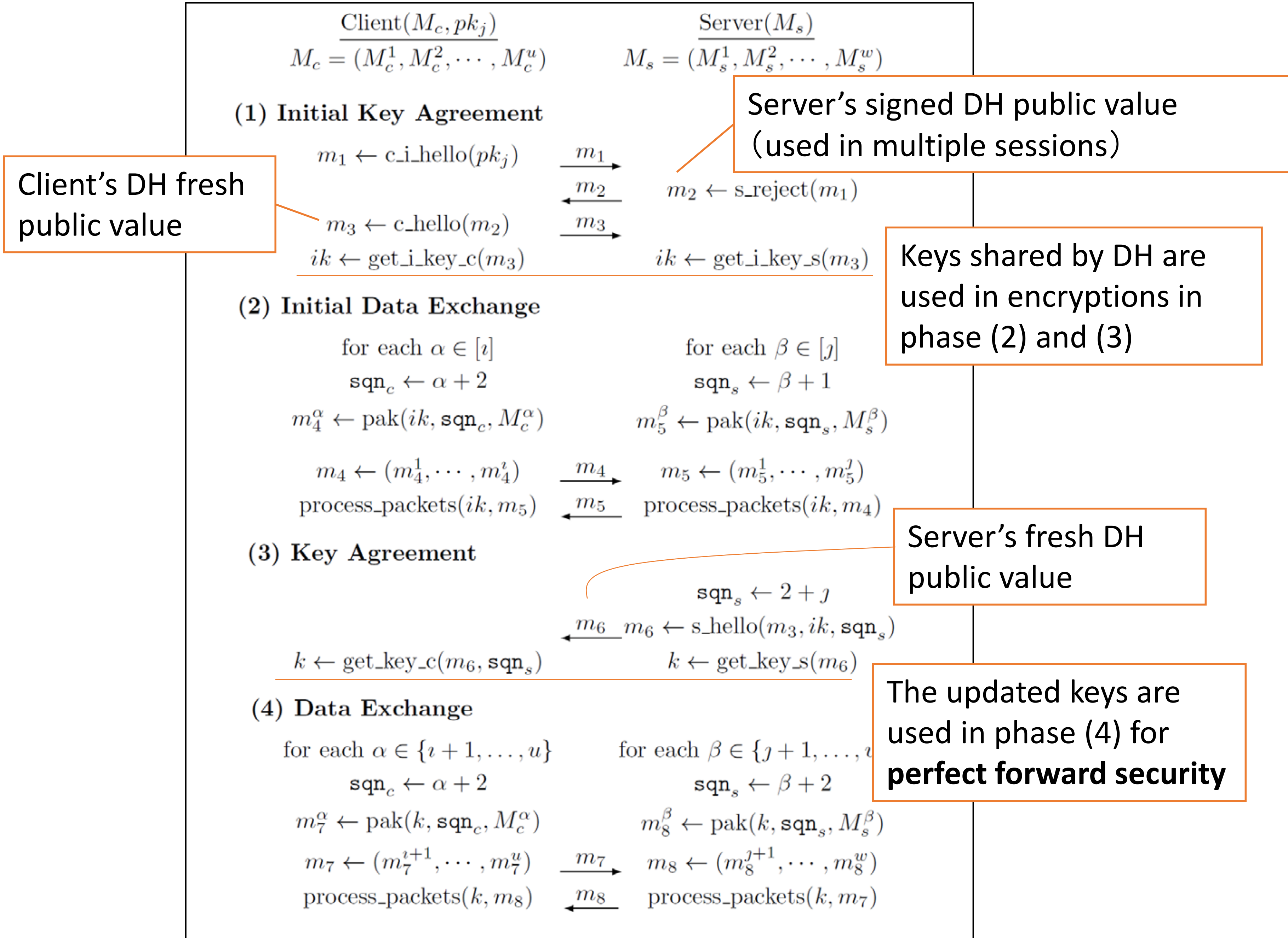


Figure 1: Summary of QUIC's 1-RTT Connection Establishment

[Lychev et al., 2015]

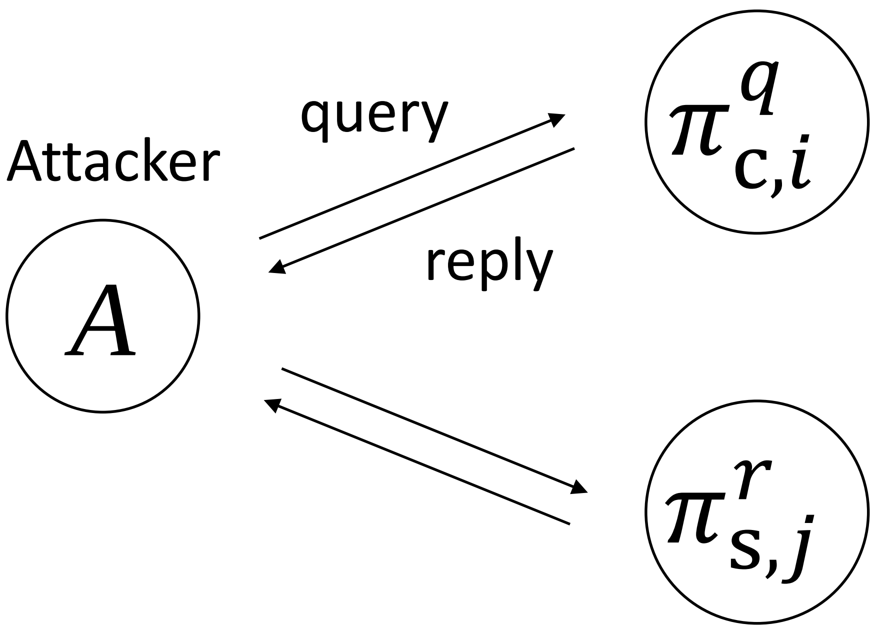
Analysis in [Lychev et al. ’15]

- Defines a security model (the QACCE model) for protocols like QUIC that allow for both (initial) non-PFS and PFS encryption.
- Defines the QACCE security for such protocols.
- Proves that QUIC satisfy the QACCE security.
- Shows that QUIC is vulnerable to some DoS attacks (outside the scope of QACCE security).

QACCE = Quick ACCE
ACCE = The Authenticated and Confidential Channel Establishment [Jager et al., 2012]

The QACCE security model

The attacker controls the sessions through queries to the oracles.



Oracle for the q-th instance of client C_i

Oracle for the r-th instance of server S_j

Queries in the QACCE model

- connect($\pi_{\mathrm{c},i}^{q}$, $\pi_{\mathrm{s},j}^{r}$): Gets the connection request message from client oracle $\pi_{\mathrm{c},i}^{q}$ to server oracle $\pi_{\mathrm{s},j}^{r}$.
- send($\pi_{\mathrm{p},i}^{q}$, m): Sends m to oracle $\pi_{\mathrm{p},i}^{q}$, and gets the reply.
- encrypt($\pi_{\mathrm{p},i}^{q}$, m, H, init), decrypt($\pi_{\mathrm{p},i}^{q}$, C, H, init): Makes oracle $\pi_{\mathrm{p},i}^{q}$ encrypt / decrypt message using authenticated encryption with header H and keys shared by the (initial if init=1) key agreement phase.

  ➡ models encryption / decryption in (initial) data exchange
- revealk($\pi_{\mathrm{p},i}^{q}$), revealik($\pi_{\mathrm{p},i}^{q}$), corrupt($\pi_{\mathrm{p},i}^{q}$): Makes oracle to reveal keys and long-term secret.

QUIC in the QACCE model

Modeled by connect and send queries: Server and client (oracles) output messages, following protocol spec.

Modeled by encrypt and decrypt queries: Server and client encrypt / decrypt message (with an attacker-chosen authentication header).

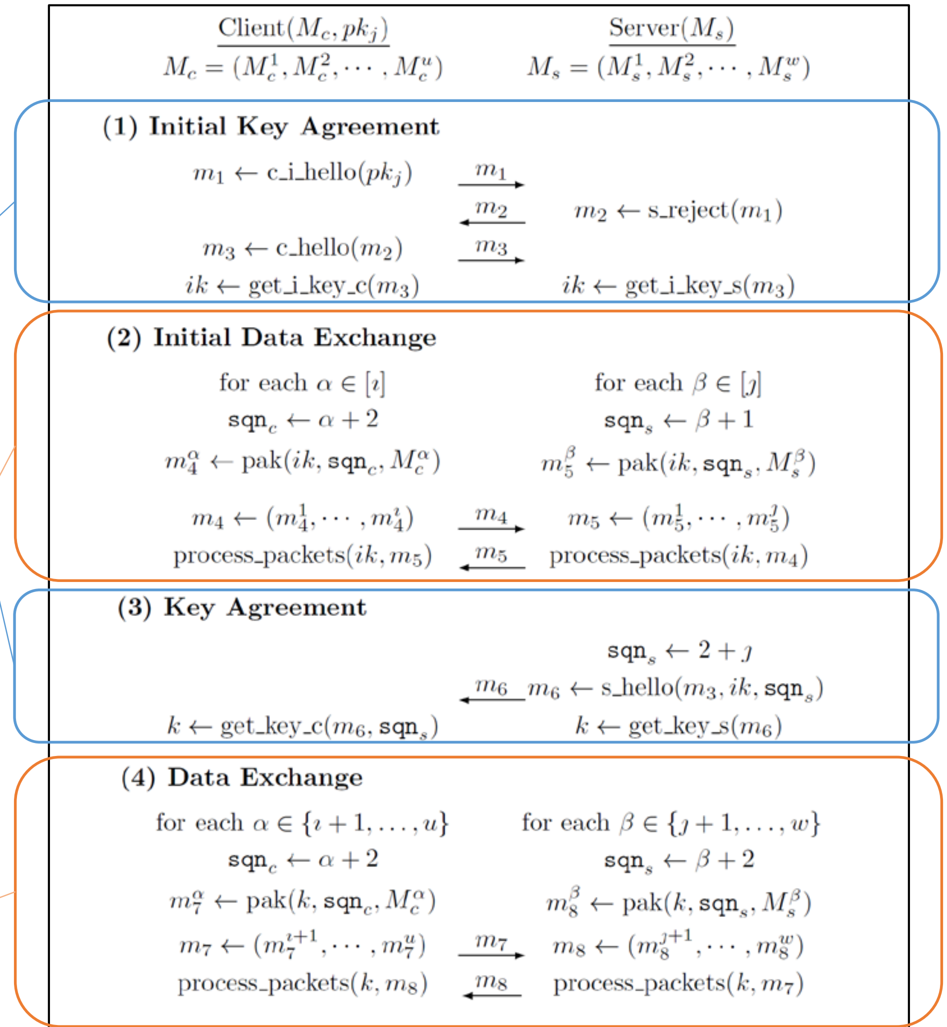


Figure 1: Summary of QUIC's 1-RTT Connection Establishment

[Lychev et al., 2015]

The QACCE security

Defines security against a number of attacks

- Server impersonation attack: succeeds if a client shares a key with someone that has no matching conversation.
- Channel corruption attack: succeeds if data, conveyed by messages in (initial) data exchange phase, are read or inserted.
- IP Spoofing attack: succeeds if a forged message is accepted by a server (assuming that the attacker does not see messages in this session.)

The ProVerif Tool

- Automatic cryptographic protocol verifier mainly developed by B. Blanche (INRIA).
- Verifies various security properties including
  - Weak secrecy (reachability)
  - Strong secrecy (indistinguishability)
  - Correspondence (authenticity)
- Assuming crypto primitives have perfect security (“Dolev-Yao”).
- Outputs “true” or “false” and an attack, if any.

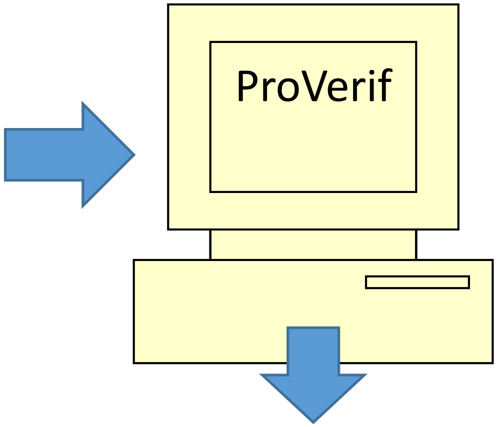
The ProVerif Tool (cont.)

```
fun pk(skey): pkey.
fun encrypt(bitstring, pkey): bitstring.
reduc forall x: bitstring, y: skey;
  decrypt(encrypt(x,pk(y)),y) = x.
  …
let client(pkS: spkey, skA: skey, skB: skey) =
  in(c, (xA: host, hostX: host));
  if xA = A || xA = B then
  let skxA = if xA = A then skA else skB in
  let pkxA = pk(skxA) in
  event beginBparam(xA, hostX);
  out(c, (xA, hostX));
  …
let server(pkS: spkey, skA: skey, skB: skey) =
  …
query x: host, y: host;
  inj-event(endBparam(x,y))
  ==> inj-event(beginBparam(x,y)).
 …
```

Assumptions on Crypto Primitives and Attacker

Protocol Descriptions

Security Requirements



Analysis Results (and attack, if insecure)

```
 …
new … at {1}
out(c, pk(skA_28379)) at {3}
insert keys(A,pk(skA_28379)) at {4}
new skB creating skB_28378 at {5}
out(c, pk(skB_28378)) at {7}
 …
RESULT
inj-event(endBparam(x,y))
==> inj-event(beginBparam(x,y))
is false.
```

Description of QUIC in ProVerif

- Oracles (clients and servers) are written as sequences of commands such as inputs and outputs (over network).
- Additionally, *Events* are issued when some queries are successfully processed, for specifying security.

```
let client(pk_s: bitstring, IP_c: bitstring, ...)=
  (* Initial Key Agreement *)
  new cid: bitstring;
  let m1 = (IP_c, IP_s, port_c, port_s, cid,  ...) in
  out(c, m1);
  in(cp, m2: bitstring);
  ...
  (* Initial Data Exchange *)
  (! Oenc((role_server, m1, m2, m3), ...)) |
  (! Odec((role_server, m1, m2, m3), ...)) |
  ...

let Oenc(matching_conversation: bitstring, ...)=
  in(c, (msg: bitstring, H: bitstring));
  let (cid: bitstring, sqn: bitstring) = H in
  let C = E(key, (iv, sqn), msg, H) in
  event encrypt(sess, ph, sender_role, C, H);
  out(c, (H, C)).
```

QACCE security in ProVerif

- Described as six assertions ("query"), which refer to events in protocol descriptions.
- E.g., to assert that messages cannot be inserted,

```
query S: bitstring, cid: bitstring, ph: bitstring, sender_role: bitstring,
        C: bitstring, H: bitstring;
  event(decrypt(S, cid, ph, sender_role, C, H)) ==>
  event(encrypt(cid, ph, sender_role, C, H))
  || event(revealed(cid, ph, sender_role))
  || event(corrupted(S)).
```

"if a <u>decryption</u> query on ciphertext C succeeds, then
an <u>encryption</u> query yielding C must be issued in the session,
or the session secret is <u>revealed</u>,
or the server is <u>corrupted</u>."

Analysis results

ProVerif finds attacks against QACCE security on Lychev et al's formalization of QUIC:

- Server-impersonation: a man-in-the middle attacker replac… ious sessio… r having no ma…
- Channel-corruption attack: an attacker can insert a (key-agreement) message in the initial data exchange phase.
- IP spoofing attack: an attacker can make server accept a message ("stk") in the previous session as a message in the cu…

Due to the (strong) definition of matching conversation in definition of QACCE security

Due to the too strong definition of IP-spoofing in definition of QACCE security

ProVerif output

⋮

new x_s' creating x_s'_2911430 at {64} in copy a_2911399, a_2911403, a_2911402, a_2911401, a_2911414

event(server_k_set(...)) at {74} in copy a_2911399, a_2911403, a_2911402, a_2911401, a_2911414

out(c, ...) at {75} in copy a_2911399, a_2911403, a_2911402, a_2911401, a_2911414

insert conversations(...) at {76} in copy a_2911399, a_2911403, a_2911402, a_2911401, a_2911414

in(c, ...) at {145} in copy a_2911399, a_2911398, a_2911413

event(client_k_set(...)) at {152} in copy a_2911399, a_2911398, a_2911413

The event client_k_set(...) is executed.
A trace has been found.
RESULT event(client_k_set(conv,sess,S)) ==> event(server_k_set(conv,sess)) || event(revealed(sess,phase_initial_data_exchange,role_server)) || event(corrupted(S)) is false.

This (very long) part describes the attack found by ProVerif

An attack found by ProVerif

$\underline{\text{Client}(M_c, pk_j)}$ $\qquad$ $\underline{\text{Server}(M_s)}$

$M_c = (M_c^1, M_c^2, \cdots, M_c^u)$ $\qquad$ $M_s = (M_s^1, M_s^2, \cdots, M_s^w)$

Decryption query succeeds without obtaining a ciphertext m_6 by encryption query, because same keys are used in (2) and (3).

➡ Key agreement message is inserted as data exchange message.

$ik \leftarrow \text{get\_i\_key\_c}(m_3)$ $\qquad$ $ik \leftarrow \text{get\_i\_key\_s}(m_3)$

(2) Initial Data Exchange

for each $\alpha \in [\imath]$ $\qquad$ for each $\beta \in [\jmath]$

$\texttt{sqn}_c \leftarrow \alpha + 2$ $\qquad$ $\texttt{sqn}_s \leftarrow \beta + 1$

$m_4^\alpha \leftarrow \text{pak}(ik, \texttt{sqn}_c, M_c^\alpha)$ $\qquad$ $m_5^\beta \leftarrow \text{pak}(ik, \texttt{sqn}_s, M_s^\beta)$

$m_4 \leftarrow (m_4^1, \cdots, m_4^\imath)$ $\xrightarrow{m_4}$ $m_5 \leftarrow ($

process_packets(ik, m_5) $\xleftarrow{m_5}$ process_p

Attacker blocks m_5.
Client is still waiting.

Attacker captures m_6 and sends it as a decryption query decrypt$(\pi^q_{p,i}, m_6, H, \text{init})$.

(3) Key Agreement

$\texttt{sqn}_s \leftarrow 2 + \jmath$

$\xleftarrow{m_6}$ $m_6 \leftarrow \text{s\_hello}(m_3, ik, \texttt{sqn}_s)$

$k \leftarrow \text{get\_key\_c}(m_6, \texttt{sqn}_s)$ $\qquad$ $k \leftarrow \text{get\_key\_s}(m_6)$

(4) Data Exchange

Is QUIC really insecure?

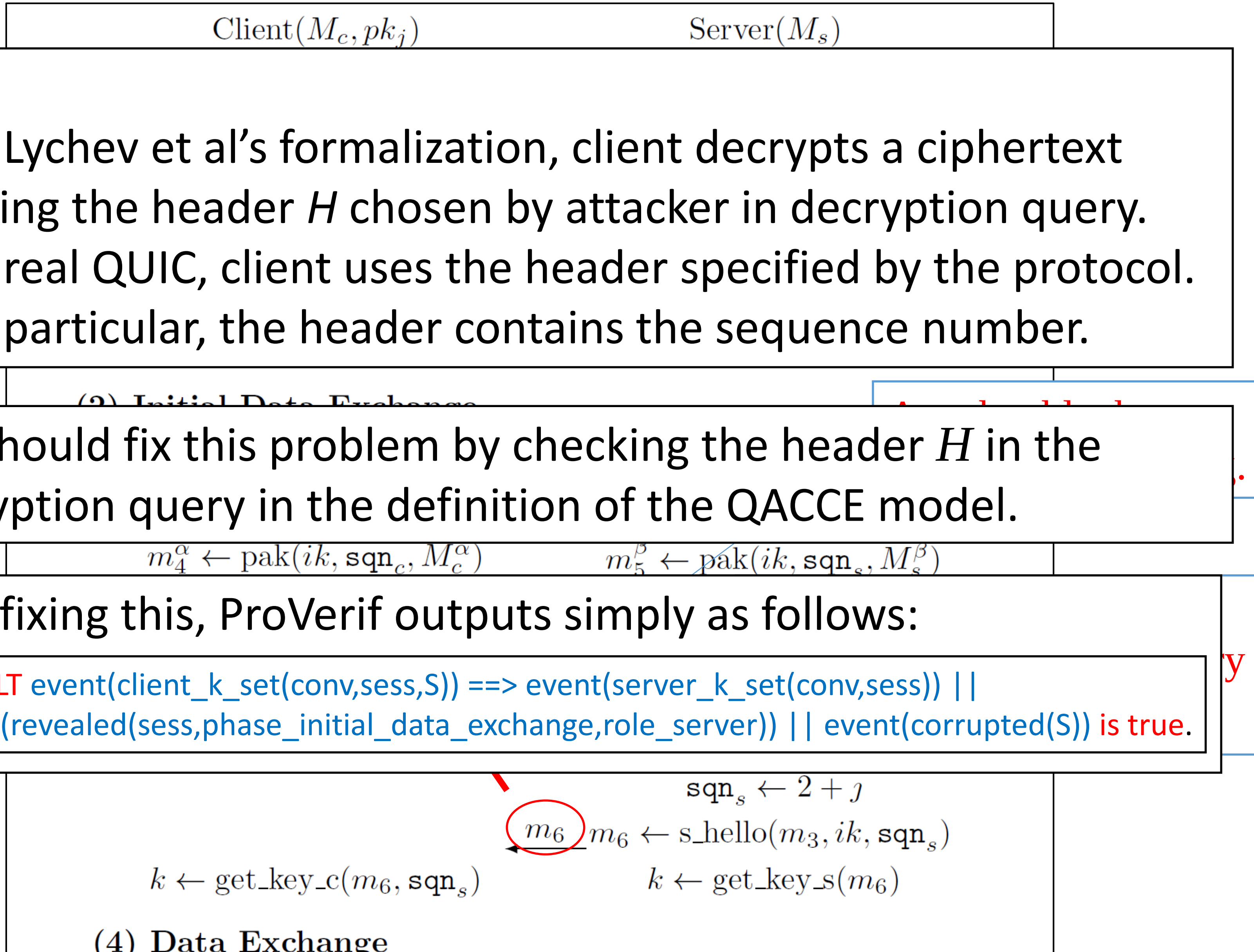
No.

- In Lychev et al's formalization, client decrypts a ciphertext using the header *H* chosen by attacker in decryption query.
- In real QUIC, client uses the header specified by the protocol. In particular, the header contains the sequence number.

We should fix this problem by checking the header H in the decryption query in the definition of the QACCE model.

After fixing this, ProVerif outputs simply as follows:

RESULT event(client_k_set(conv,sess,S)) ==> event(server_k_set(conv,sess)) || event(revealed(sess,phase_initial_data_exchange,role_server)) || event(corrupted(S)) is true.



Summary of the analysis

- 400 lines of ProVerif script, including protocol, security requirements, and crypto primitives definitions.
- Time required by analysis:

Security	Before fix the model	After fix the model
Server impersonation	7[min] 55[sec]	8[min] 39[sec]
Channel-corruption (message insertion)	7[min] 11[sec]	6[min] 20[sec]
Channel-corruption (secrecy)	63[min] 32[sec]	65[min] 57[sec]
IP spoofing	7[min] 58[sec]	6[min] 7[sec]

(Security against channel-corruption attack is divided into security against message insertion and secrecy)

Conclusion

- We analyzed the QACCE security of QUIC by using ProVerif.
- ProVerif found a number of attacks on QUIC.

➡ Lychev et al.'s proof of QACCE security of QUIC contains some errors.

- These attacks are due to inappropriate formalization of QUIC and definition of QACCE model and do no real harm in reality.
- But show that hand-written proofs may contain errors (even by an expert and in paper accepted in top conference.)

Source-address token (stk)

- Client's IP address encrypted using the key known only by a server.
- Used for avoiding IP-spoofing attack.
- On receiving an initial connection request, the server makes stk and sends to the client.
- A client sends stk to the server in the c_hello message, and the server checks if

  the source IP address = the IP address in stk.
- An stk can be used in a later session, in which the c_i_hello and s_reject message are omitted.

QUIC in the QACCE model

Modeled by connect and send queries: Server and client outputs messages, following the protocol specification.

Modeled by encrypt and decrypt queries: Server and client encrypt / decrypt message (with an attacker-chosen authentication header).

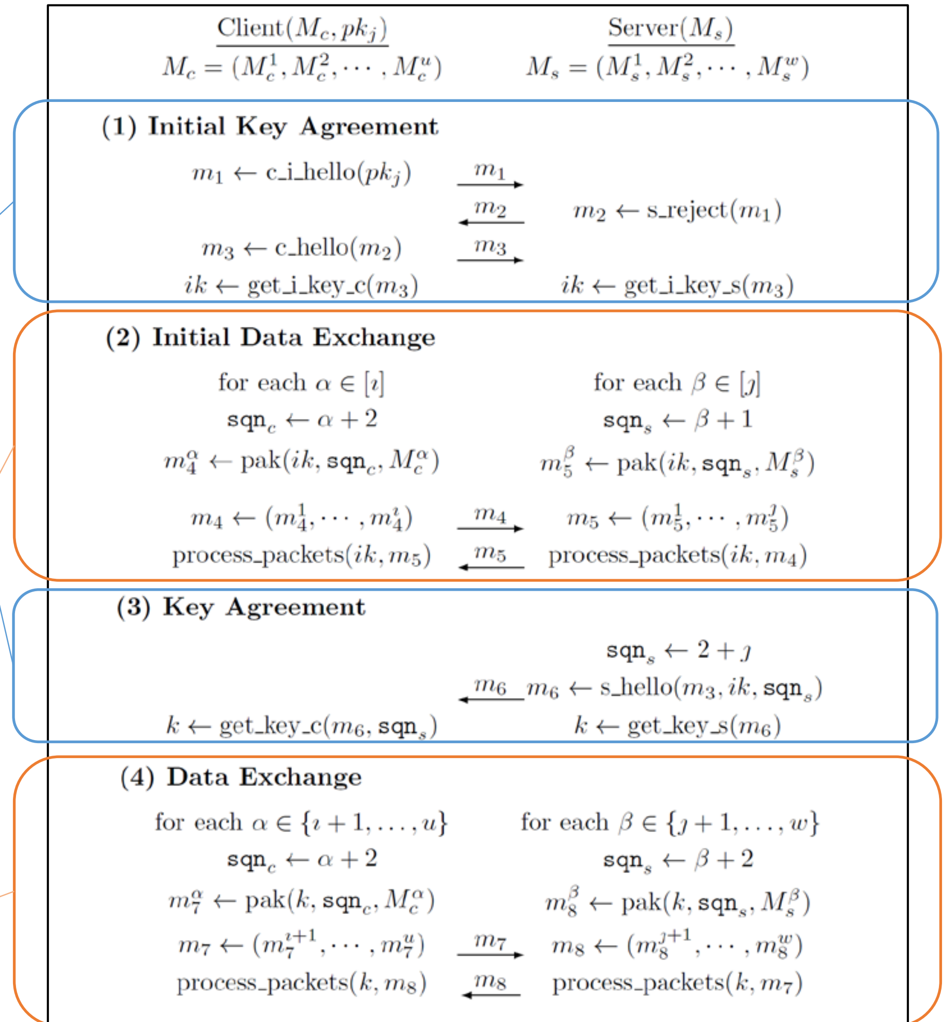


Figure 1: Summary of QUIC's 1-RTT Connection Establishment

[Lychev et al., 2015]