ProScript TLS:
Building a TLS 1.3 Implementation with a Verifiable Protocol Model

Karthikeyan Bhargavan
Nadim Kobeissi
Bruno Blanchet

miTLS and flexTLS

miTLS: reference implementation of TLS 1.0-1.2

- Written in F#, a .NET variant of OCaml
- Verified using a dependent type system (F7)
  + some crypto proofs in EasyCrypt
- Built to support proof from ground up
- See Antoine's talk for new developments

flexTLS: specification-based testing for TLS

- Uses miTLS as a reference spec
- Good for experimenting with new features
- Easy to implement known attacks, find new ones

ProScript TLS

Goal: get developers to run light-weight analysis

- F#/F* too far a leap for many developers
- We want them to write their own tests and attacks
- But don't break miTLS to make it easier to test with!

A reference implementation of TLS in JavaScript

- Protocol core written in a statically-typed, purely functional subset of JavaScript called ProScript
- Typing avoids common JavaScript pitfalls
- Model extraction & verification with ProVerif
- Not a cryptographic proof! Good for finding bugs.

ProScript TLS

Current implementation status

- Implements TLS 1.0-1.3 (1-RTT)
- RSA/DHE/ECDHE, AES-CBC/GCM
- Interoperates with NSS for TLS 1.3 for 1-RTT
- Interoperates with clients/servers for TLS 1.0-1.2

Current verification status

- An extracted model for the core of 1-RTT
- Verifies standard secrecy/authentication properties
- Ongoing work: 0-RTT, PSK, TLS 1.0-1.2

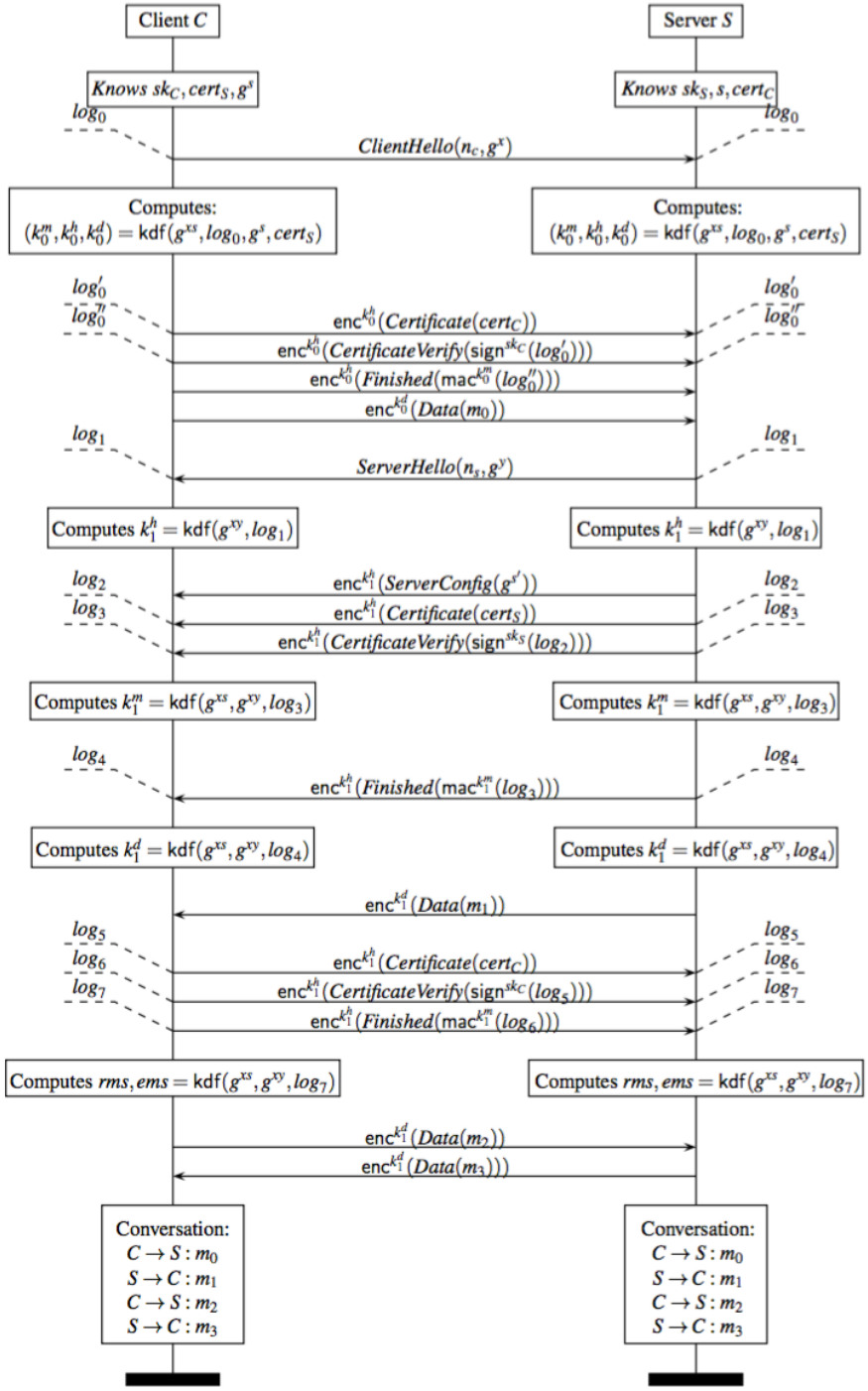
TLS 1.3 Protocol

Draft 11 specification

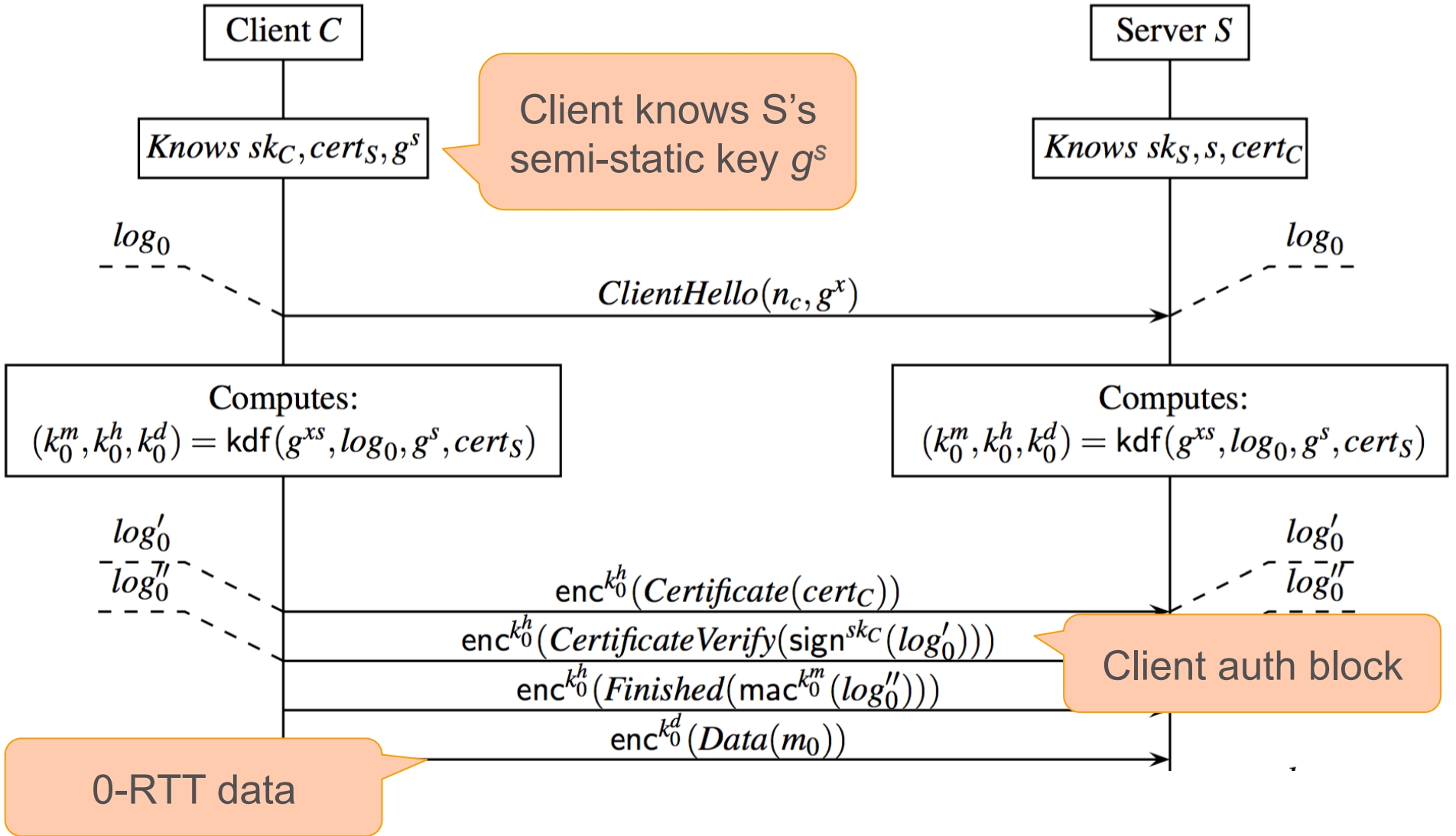
- 0-RTT with DH/PSK
- 0.5 RTT server data
- 0- or 1-RTT client auth

Complex key schedule

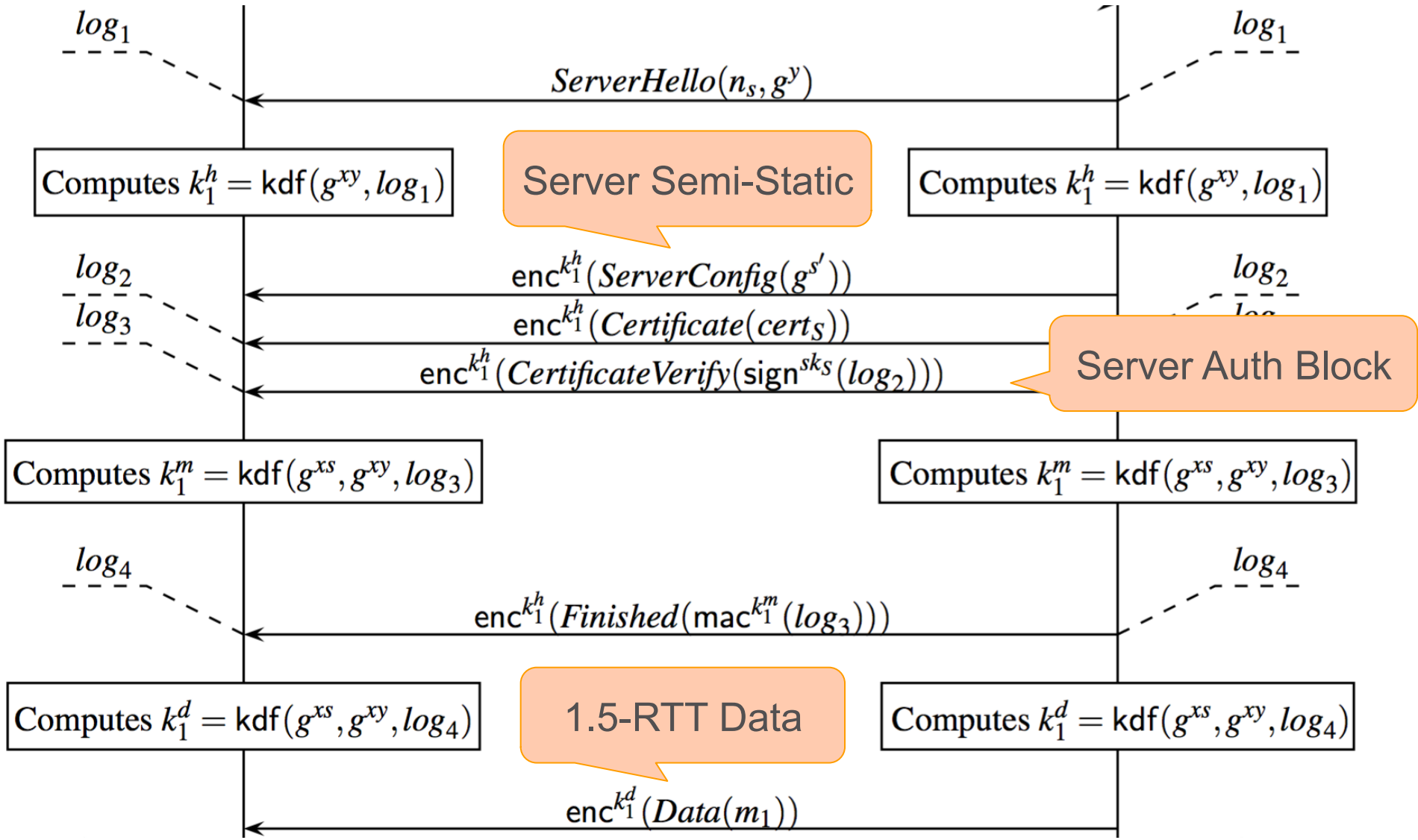
- Keys at derived at multiple stages
- Record keys change at different places
- Difficult to debug



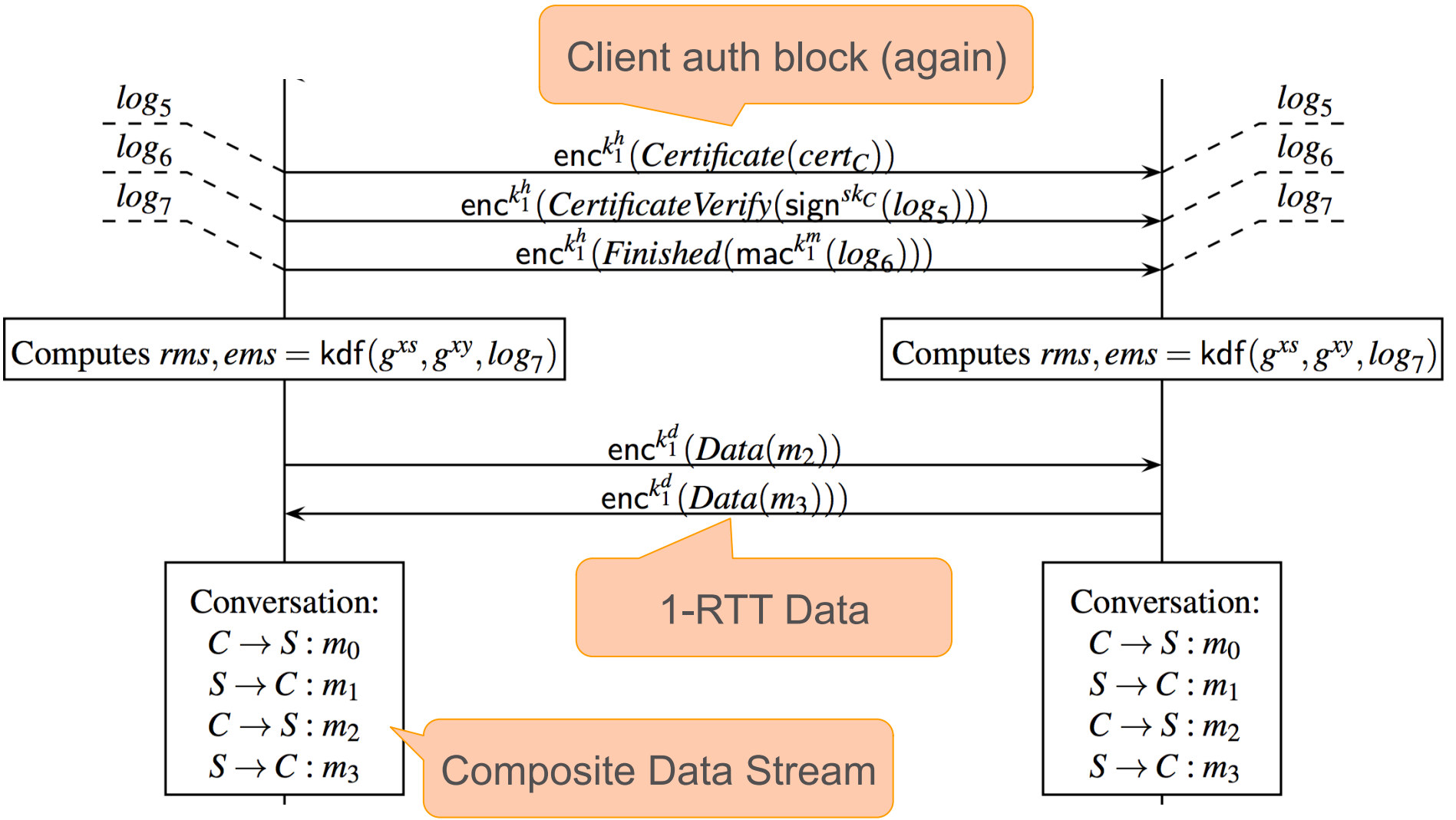
(1) Client Auth + 0-RTT Data



(2) Server Auth + 0.5-RTT Data



(3) Client Auth + 1-RTT Data



TLS 1.3 in JavaScript

Handshake messages processed in flights

- Record layer collects message flights, then calls a purely functional callback function
- Callbacks return message flights to send back
- Some flights broken up to support key changes

Client	Server
send_client_hello	recv_client_hello
recv_server_hello	send_server_finished
recv_server_finished	recv_client_finished
write_data	write_data
read_data	read_data

FlexTLS in JavaScript

Add new features, implement attacks

- Often only need to edit one callback
- Demos for Skip, Freak, Sloth

- **Ongoing work**: systematic testing for TLS 1.3 (a la SMACK)

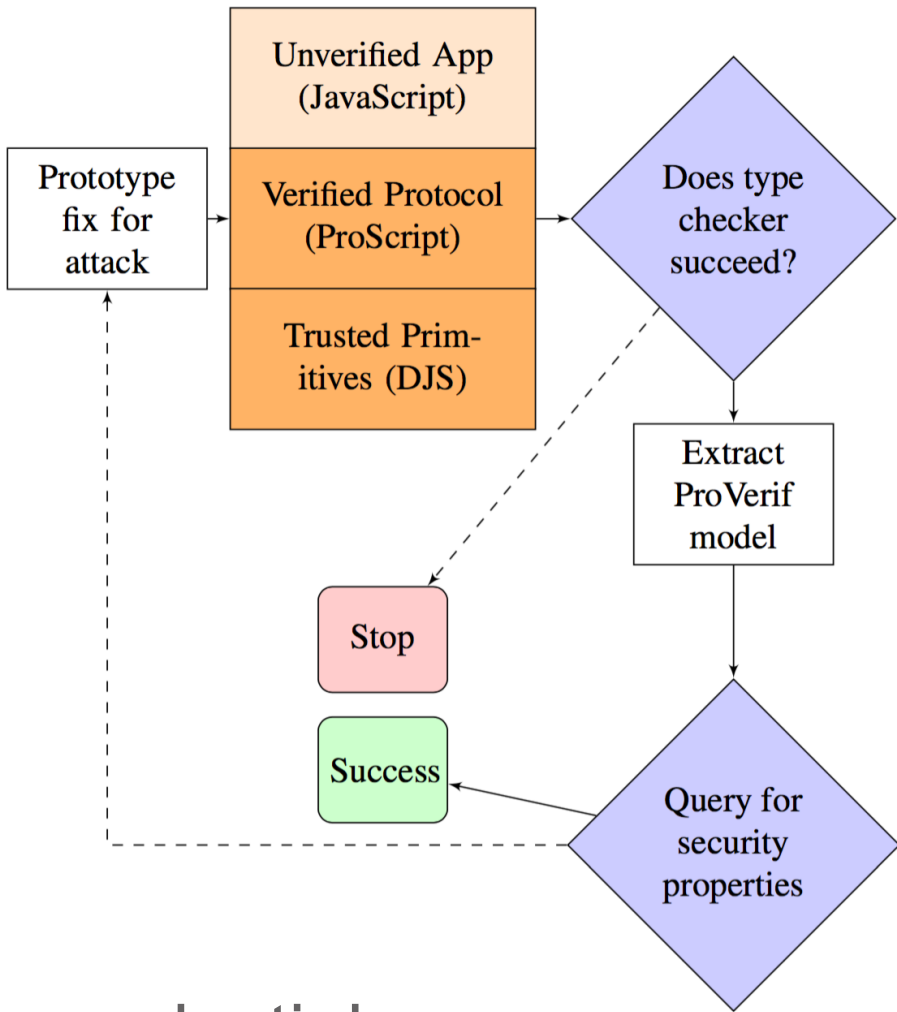
```
/* Test for the Skip-CCS attack by deleting the ServerCCS messsage */
const Skip_CCS_server_callbacks = {
  hs_recv_client_hello: server_callbacks.hs_recv_client_hello,
  hs_recv_client_ccs: server_callbacks.hs_recv_client_ccs,
  hs_recv_client_finished: function(msgs, cipherState) {
    let out_msgs = server_callbacks.hs_recv_client_finished(msgs,cipherS
    cipherState.require_alert = true;
    return [out_msgs[1]];
  }
}

/* Test for the SLOTH attack by modifying the ServerKeyExchange messag
const Sloth_server_callbacks = {
  hs_recv_client_hello: function(msgs, cipherState) {
    let out_msgs = server_callbacks.hs_recv_client_hello(msgs,cipherSta
    let ske = out_msgs[2];
    ske.sig.hash_alg = formats.HA.md5;
    ske.sign(cipherState.cr, cipherState.sr, cipherState.pv);
    return out_msgs
  },
  hs_recv_client_ccs: server_callbacks.hs_recv_client_ccs,
  hs_recv_client_finished: server_callbacks.hs_recv_client_finished,
}
```

TLS 1.3 in ProVerif

Deconstruct TLS source

- **Trusted Typed DJS:** Crypto, message formats
- **Verified ProScript:** Core protocol code (1000 / 4500 loc total)
- **Untrusted JavaScript:** Application, network, connection handling



Extract, verify in ProVerif

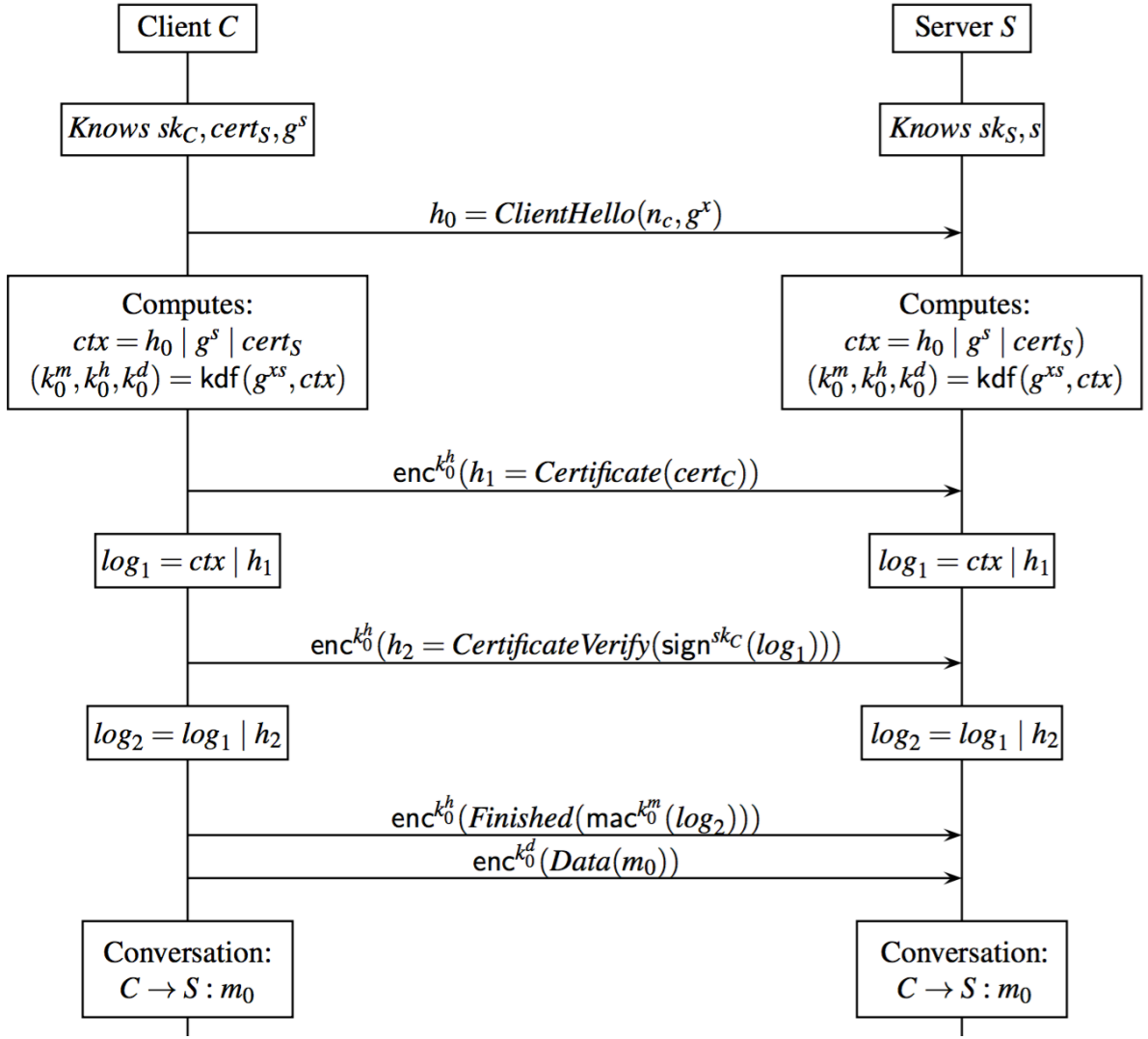
- add attacker model, security goals, tinker, …

Analysis:
Weaknesses in 0-RTT Client Auth

Client Auth + 0-RTT Data

Security Goals:

- **Secrecy:** Data m_0 known only to C and S
- **Forward secrecy:** even if sk_S leaks even if s leaks?
- **Authentication:** C and S agree on sender pk_C, receiver pk_S, and data m_0



Known Weaknesses in 0-RTT

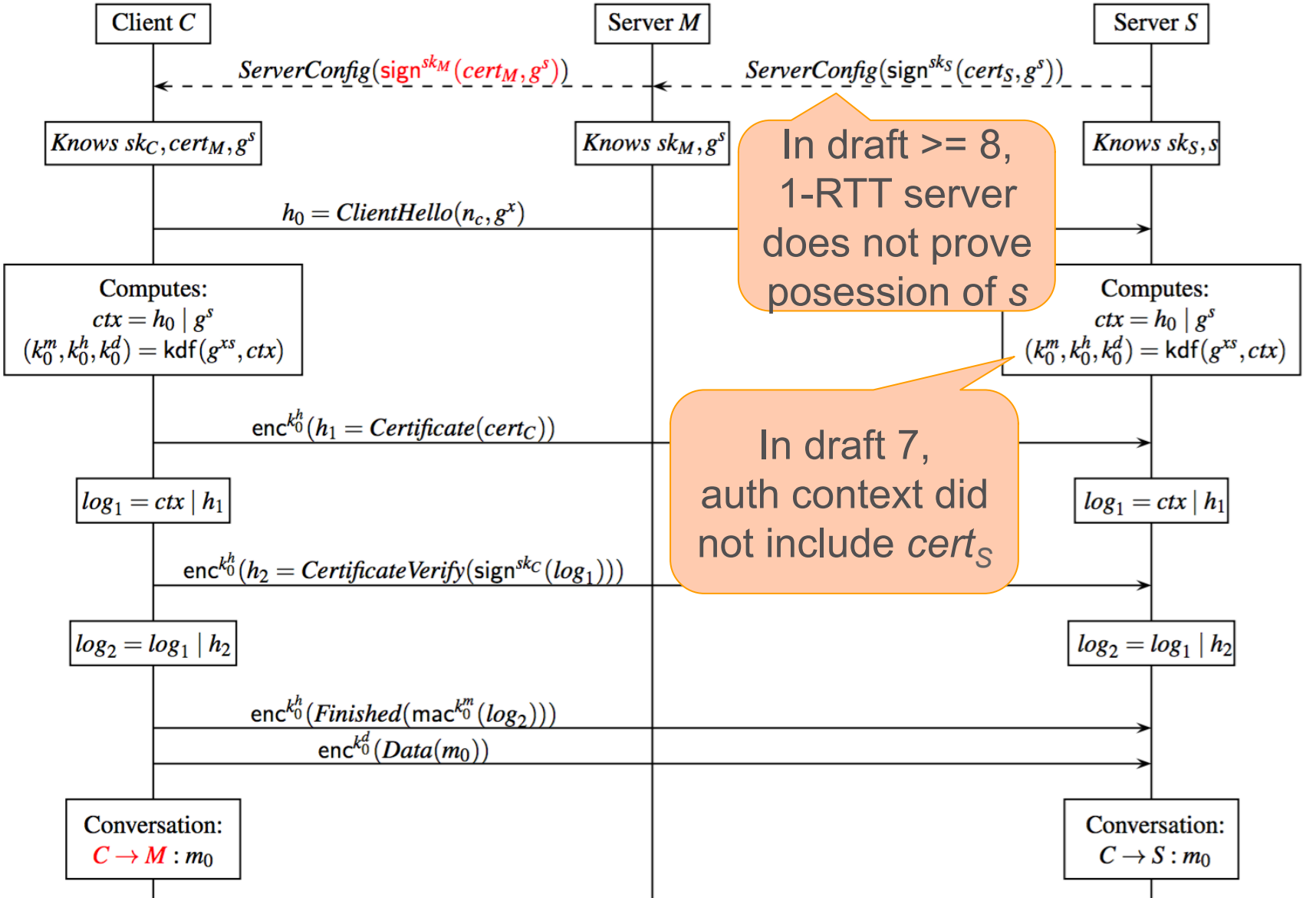
0-RTT data is not forward secret if *s* is leaked

- Similarly, 0-RTT PSK is not forward secret
- Appears in our model as an attack
- *Solution*: rotate semi-static keys and PSKs

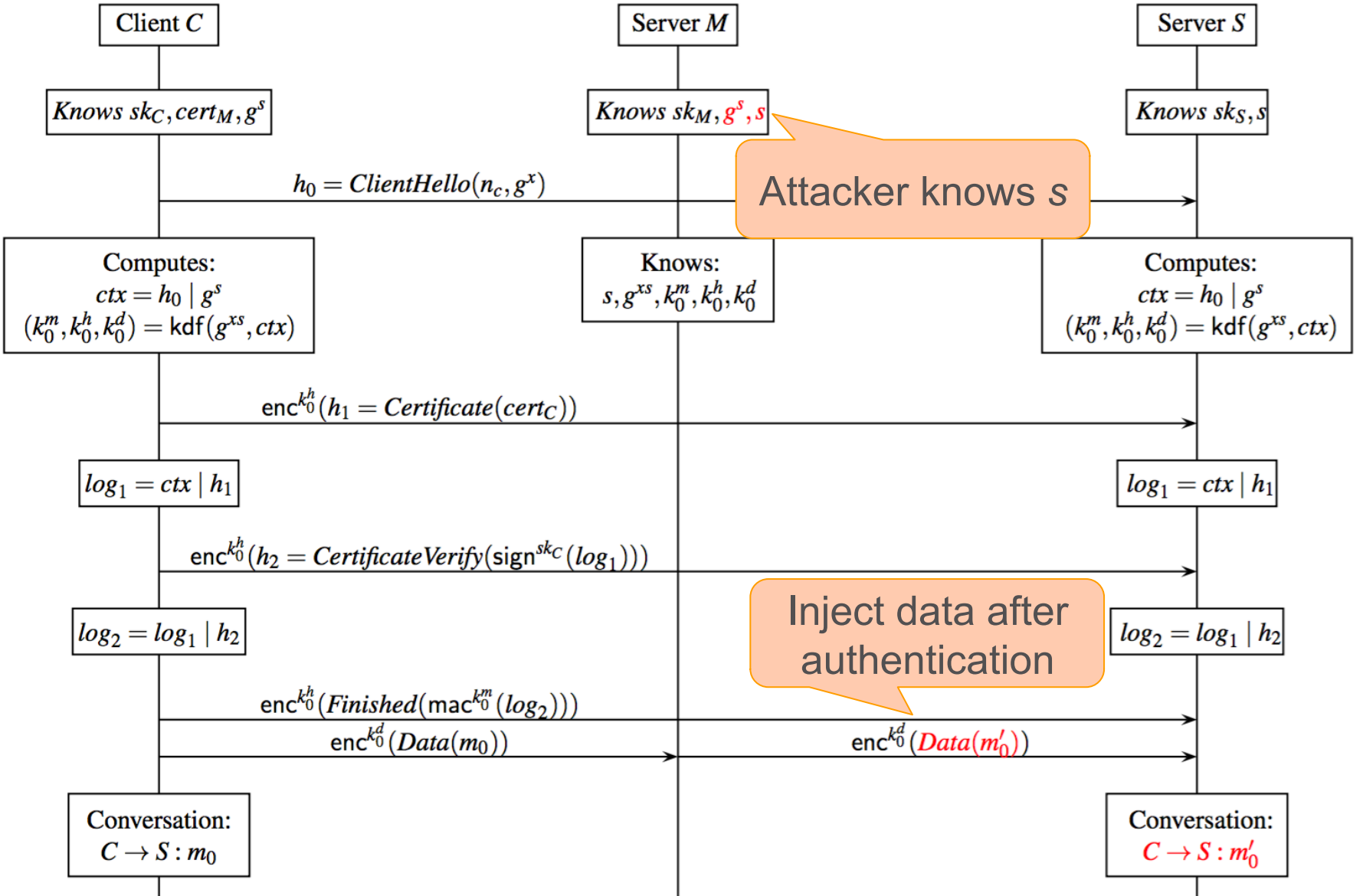
0-RTT data + client auth is replayable

- Authentication makes replay worse
- *Solution*: API needs to forbid 0-RTT POSTs e.g.
- **New query attack vector**: replayed 0-RTT requests may be responded with by 0.5-RTT response
- *Example*: authenticated GET whose response length is sensitive; even off-path attacks possible!

Unknown Key Share (Draft 7.5)



Key Compromise Impersonation

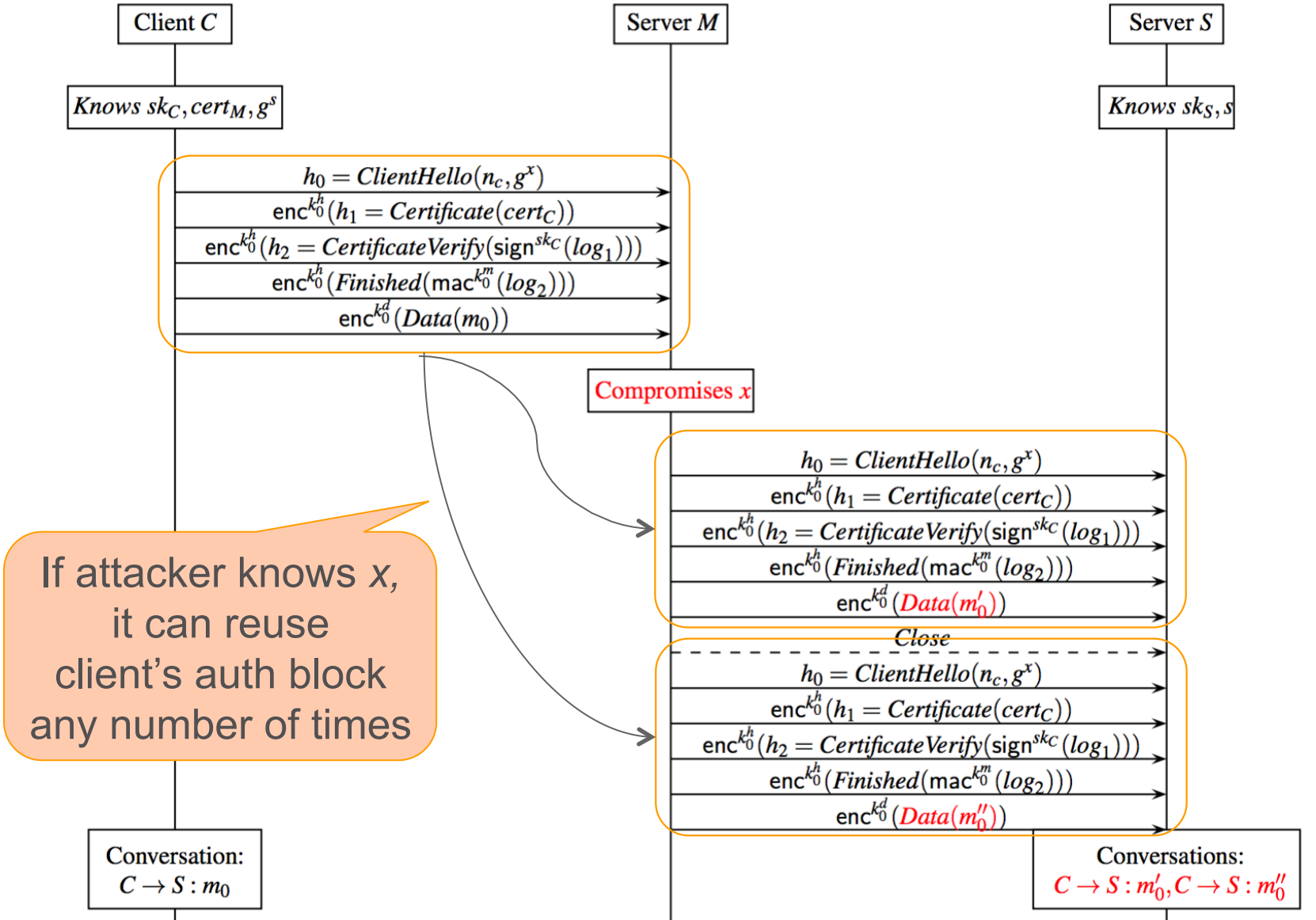


Long-Term Client Impersonation

If client ephemeral *x* is leaked,
attacker can forward 0-RTT client auth forever

- Result of replay + ephemeral compromise
- 0-RTT client auth unintentionally creates a long-term delegated credential
- *Example:* suppose client certificate is on smartcard, but ephemeral is on a public machine and can leak.
- *Solution:* Replay detection?

Long-Term Client Impersonation



Summary: 0-RTT Client Auth

- 0-RTT Auth is replayable
  and amplifies attacks on 0.5-RTT responses
- 0-RTT Auth is not forward-secure if *s* leaks
- 0-RTT Auth is vulnerable to KCI if *s* leaks
- 0-RTT Auth leaks signature capability if *x* leaks

Question:
Shall we get rid of certificate-based auth in 0-RTT?

Many of these problems do not seem to occur with PSK.

Analysis:
Mixing PSK with Signatures

PSK in Draft 11

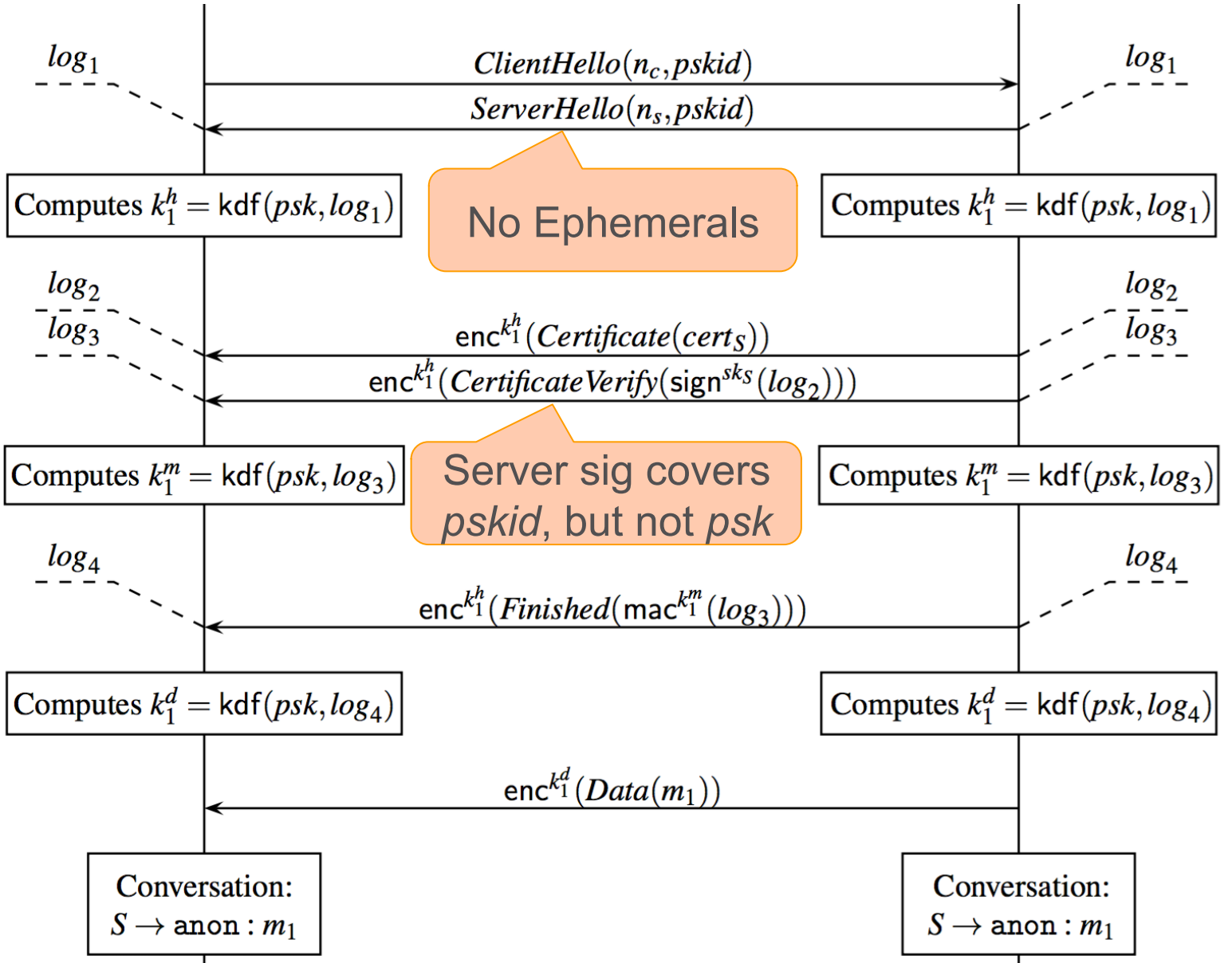
Multiple modes and key sources

- Static PSK vs. Resumption Master Secret
- PSK-DHE vs. Pure PSK
- 0-RTT and/or 1-RTT PSK ciphers

PSK + certificate-based authentication

- Allowed for 1-RTT client auth (Thyla's talk)
- Maybe allowed in 0-RTT, but underspecified
- Not allowed in 1-RTT server auth (but should be?)
- How to correctly compose PSK + signatures?

Pure PSK + Server Signature



Attacks on PSK + Signatures

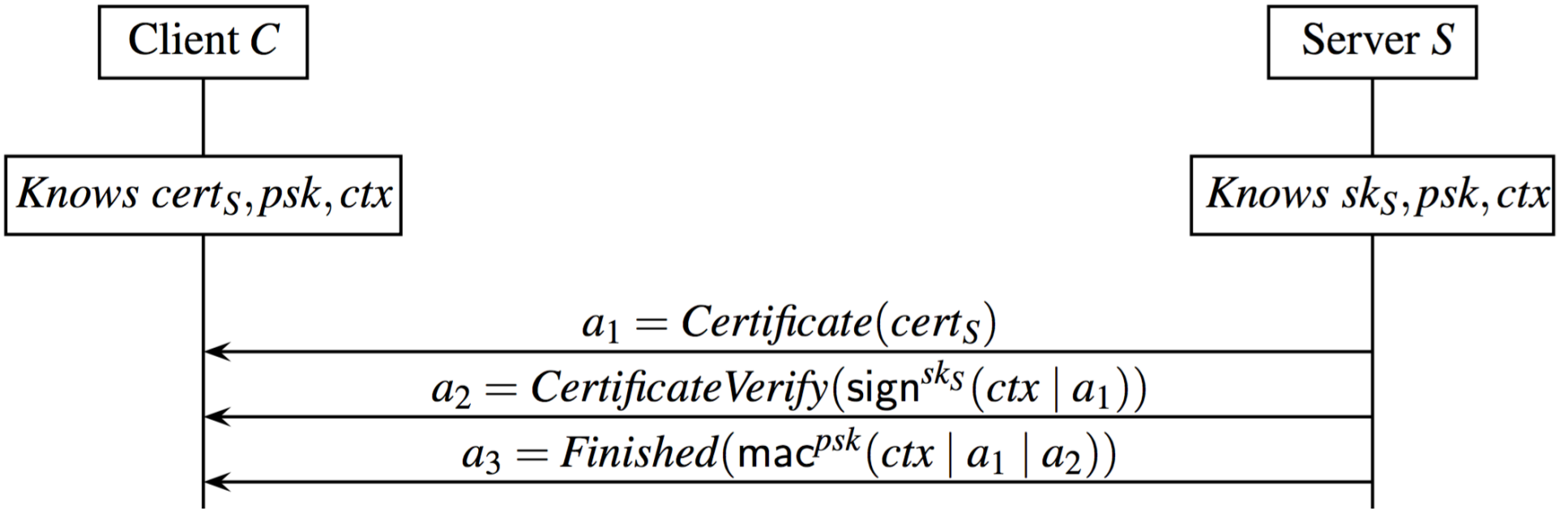
Impersonating Servers over Pure PSK

- Suppose C has a PSK with *pskid* at M
- Suppose attacker M has a PSK with same *pskid* at S
- M can forward S's signature to C
- C thinks it is talking to M, but is talking to S
- Similar attack to Cremers et al (Thyla's talk)
- No ServerFinished to save us here

Impersonating 0-RTT Clients over Static PSK

- Attacker M synchronizes *pskid* over C-M and C-S
- M forwards C's signature to S

Compound Authentication



Certificate does not authenticate PSK, although PSK does authenticate certificate

- In isolation, this sub-protocol does not guarantee compound authentication, resulting in attacks
- Handshake encryption under PSK does not help

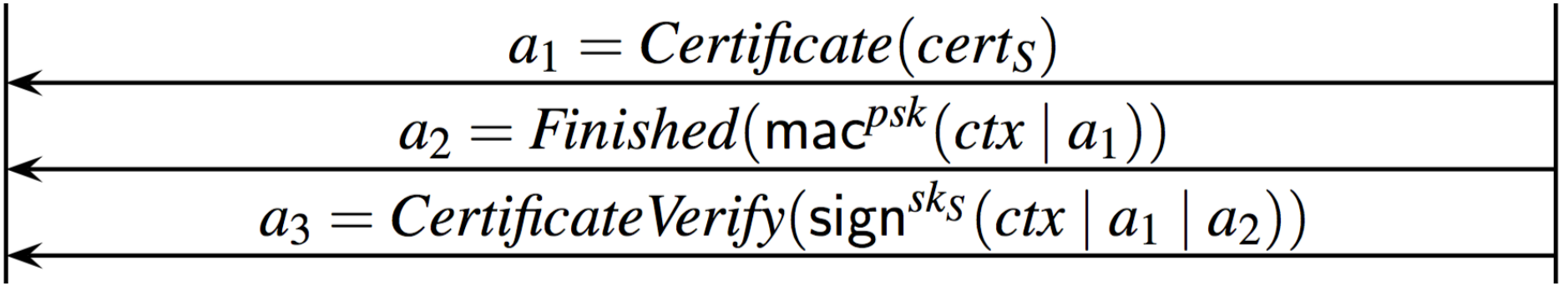
Compound Authentication

Solution: add more PSK-related info to context

- E.g. add resumption sesion hash to *pskid*
- E.g. add *cert* to *ctx*, enforce unique *pskid* per *cert*

Alternative: switch CertificateVerify and Finished

- MAC still covers certificate
- Signature now covers a PSK-based MAC
- Generic mitigation for authentication sub-protocol



Conclusions

Verifying models derived from code is effective

- Keeps us honest + enables quick formal feedback

0-RTT Client Auth is fragile against compromise

- Well-documented in spec; get rid of it?

PSK + signatures do not mix easily

- Protocol can be redesigned to compose them well

0-RTT replay is a source of headaches

- Replay detection for client randoms?

Key schedule can potentially be simplified

- Symbolic analysis does not detect new attacks

Questions?

- *Coming Soon*: ProScript TLS on GitHub