Automated analysis and the subtleties of authentication

Adventures in TLS 1.3

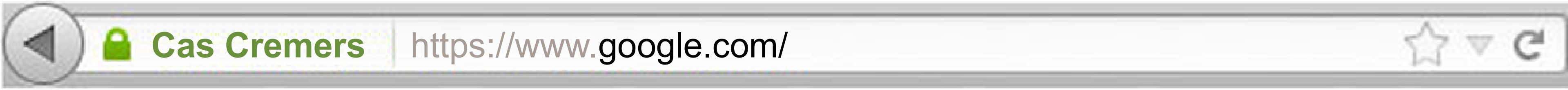
Cas Cremers

OAUTH Workshop, July 2017, Zurich, Switzerland

Based on joint work with Thyla van der Merwe, Sam Scott, Marko Horvat, and Jonathan Hoyland

TLS: **T**ransport **L**ayer **S**ecurity

The *purpose* of TLS:
To provide a secure channel to transfer messages



TLS is super secure!

TLS is super secure!

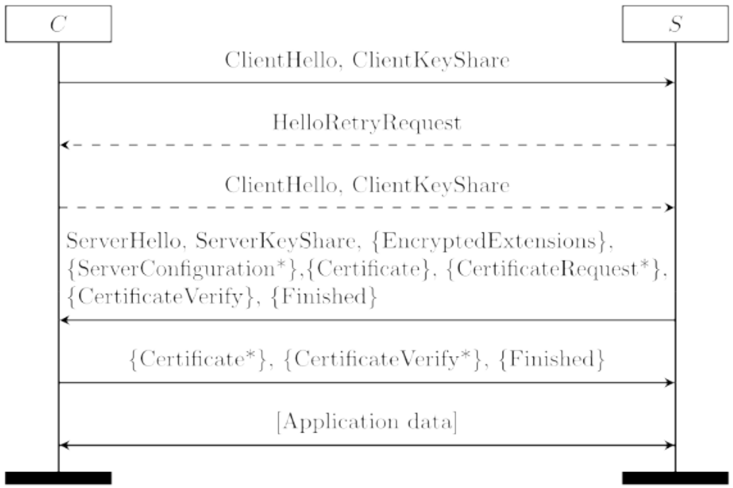
Currently under development: TLS 1.3

TLS 1.3 goals?

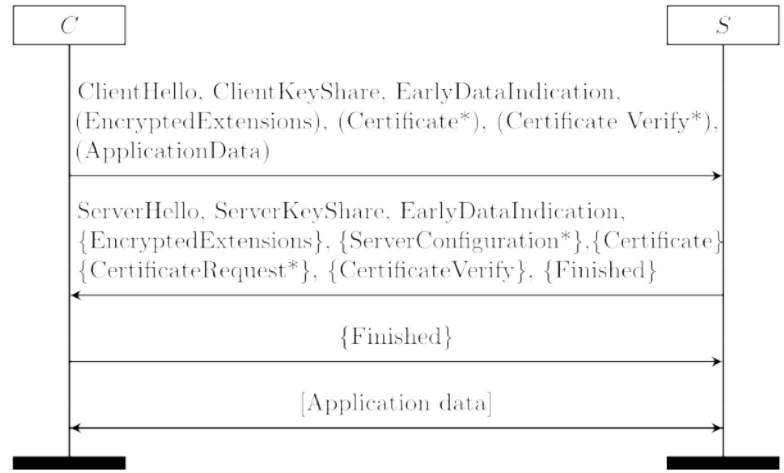
- Get rid of older features
  - Ciphersuites, non-PFS, ...
- Reduce initial communication cost
  - The 0-RTT minefield
- Clean up design
- Involve more specialists for more assurance

Could have been called TLS 2...

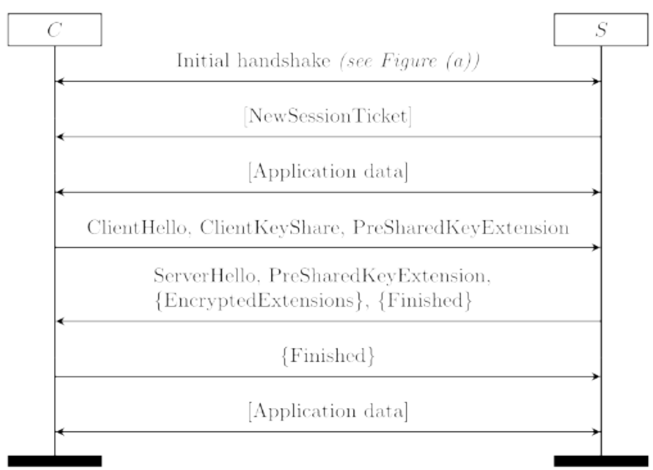
TLS 1.3



(a) Initial (EC)DHE handshake



(b) 0-RTT handshake



(c) PSK-resumption handshake (+PSK-DHE)

Post-handshake client authentication

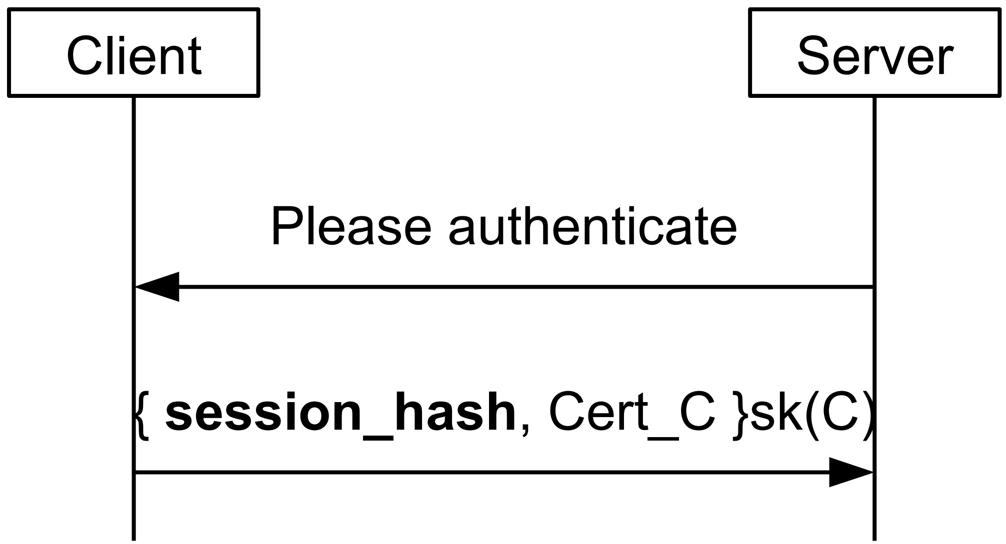
- Most common TLS use: **unilateral authentication**

- In some scenarios, we would like to later **upgrade a connection** to **mutually authenticated**

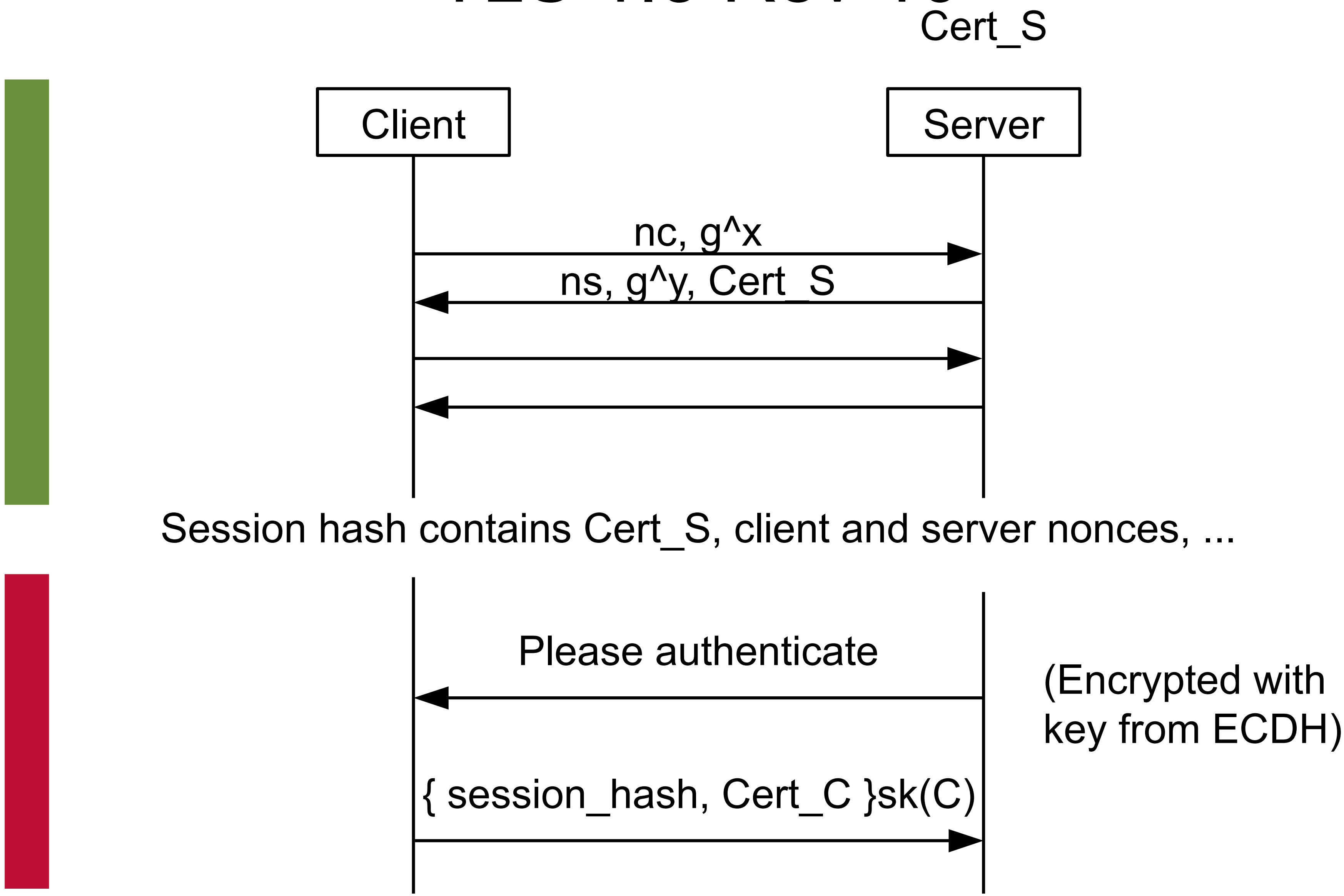
- TLS 1.2: Renegotiation

- TLS 1.3: **Post-handshake client authentication**
  - A.k.a. delayed authentication

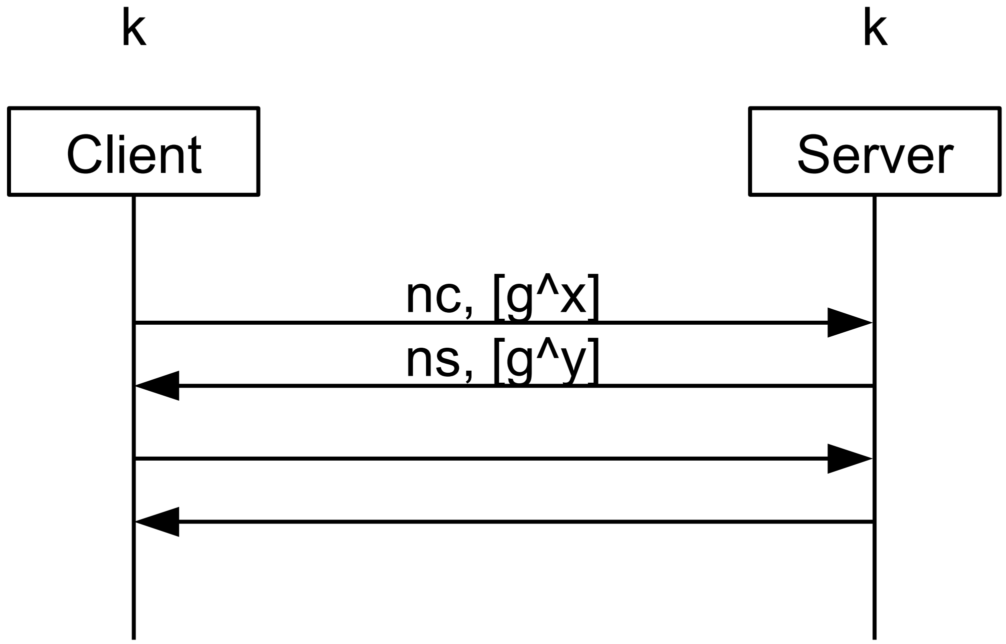
Post-handshake client authentication: TLS 1.3 Rev 10



Session hash from **ECDH handshake** TLS 1.3 Rev 10



Session hash from **PSK[-DHE] mode** TLS 1.3 Rev 10



Session hash contains ~~Cert_S,~~ client and server nonces, ...

Formal analysis using the Tamarin Prover?

Scyther vs Tamarin

Tamarin contributors

Simon Meier

Benedikt Schmidt

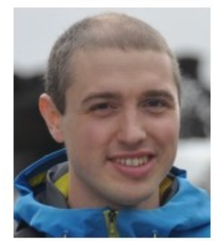
Cas Cremers

David Basin

Robert Kunneman

Steve Kremer

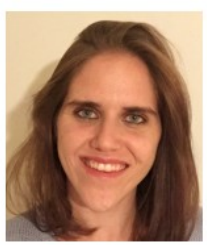
Ralf Sasse



Jannik Dreier

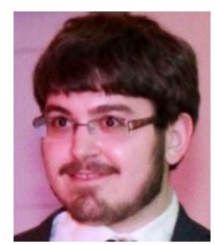
Cedric Staub

Sasa Radomirovic



Lara Schmid

Charles Dumenil



Kevin Milner

and more soon!

Tamarin prover: main ingredients

- More **expressive input language**
  - Models with loops, branching
  - Property specification in a fragment of first-order logic with quantification over timepoints
- More **powerful analysis techniques**
  - Constraint-solving backend
  - User can inspect (partial) proofs
  - User can provide invariants ("hints") to the prover

Running Tamarin 1.1.0 | Index | Download | Actions » | Options »

Proof scripts

```
theory FirstExample begin

Message theory

Multiset rewriting rules (8)

Untyped case distinctions (10 cases, all chains solved)

Typed case distinctions (10 cases, all chains solved)

lemma Client_session_key_secrecy:
  all-traces
  "¬(∃ S k #i #j.
          ((SessKeyC( S, k ) @ #i) ∧ (K( k ) @ #j)) ∧
          (¬(∃ #r. LtkReveal( S ) @ #r)))"
simplify
solve( Client_1( S, k ) ▶₀ #i )
  case Client_1
  solve( !KU( ~k ) @ #vk.1 )
    case Client_1
    solve( !KU( ~ltk ) @ #vk.2 )
      case Reveal_ltk
      by contradiction /* from formulas */
    qed
  qed
qed

lemma Client_auth:
  all-traces
  "∀ S k #i.
         (SessKeyC( S, k ) @ #i) ⇒
         ((∃ #a. AnswerRequest( S, k ) @ #a) ∨
          (∃ #r. (LtkReveal( S ) @ #r) ∧ (#r < #i)))"
by sorry

lemma Client_auth_injective:
  all-traces
  "∀ S k #i.
         (SessKeyC( S, k ) @ #i) ⇒
         ((∃ #a.
            (AnswerRequest( S, k ) @ #a) ∧
            (∀ #j. (SessKeyC( S, k ) @ #j) ⇒ (#i = #j))) ∨
          (∃ #r. (LtkReveal( S ) @ #r) ∧ (#r < #i)))"
by sorry

lemma Client_session_key_honest_setup:
```

Case: Reveal_ltk

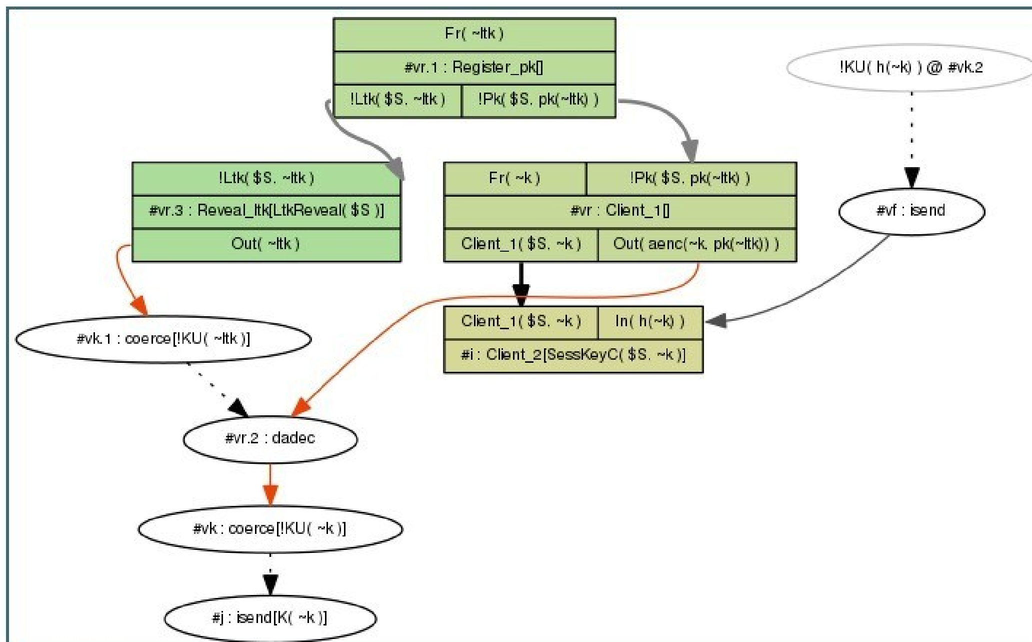
Applicable Proof Methods: Goals sorted according to the 'smart' heuristic (loop breakers delayed).

1. **contradiction** /* from formulas */

2. **solve(** !KU(h(~k)) @ #vk.2 **)** // nr. 4

a. **autoprove** (A. **for all solutions**)
b. **autoprove** (B. **for all solutions**) with proof-depth bound 5

Constraint system



last: none

formulas:

Selected case studies

- Key exchange
  - Naxos
  - Signed DH
  - KEA+
  - UM
  - Tsx
- Group protocols
  - GDH
  - TAK
  - (Sig)Joux
  - STR
- ID-based AKE
  - RYY
  - Scott
  - Chen-Kudla
- Loops
  - TESLA1 & 2
- Non-monotonic global state
  - Keyserver
  - Envelope
  - Exclusive secrets
  - Contract signing
  - **Security device/HSMs**
  - YubiKey
  - YubiHSM
  - **Vehicle-to-vehicle/automotive**
- PKI with strong guarantees
  - ARPKI (also global state)
- Transparency
  - DECIM (also global state)
- **Etc etc.**

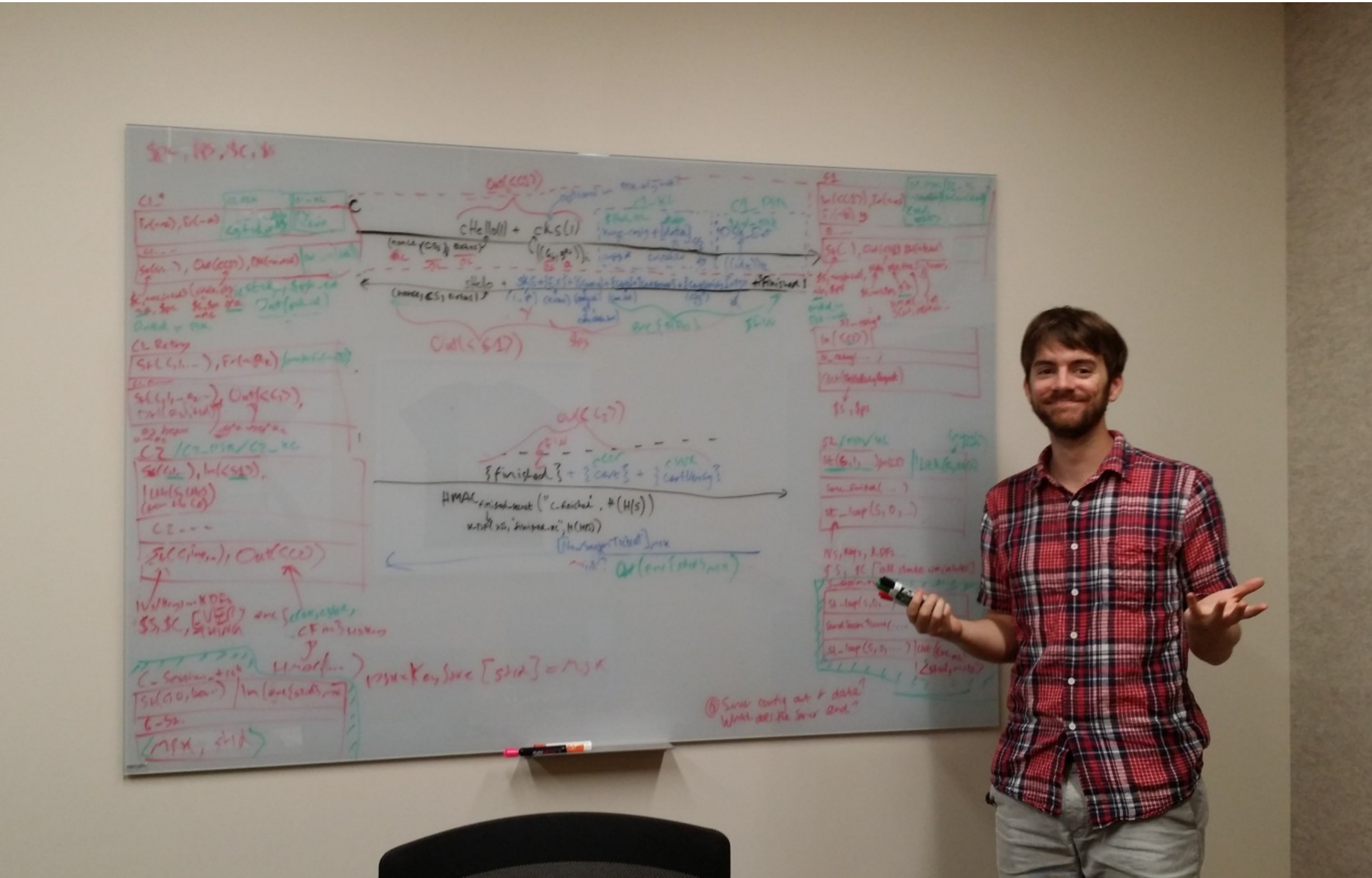
Formal analysis possible?

- Modeled the TLS 1.3 specification under development
  - at this time: draft 10
- Goal: verify the **core** security properties of TLS 1.3

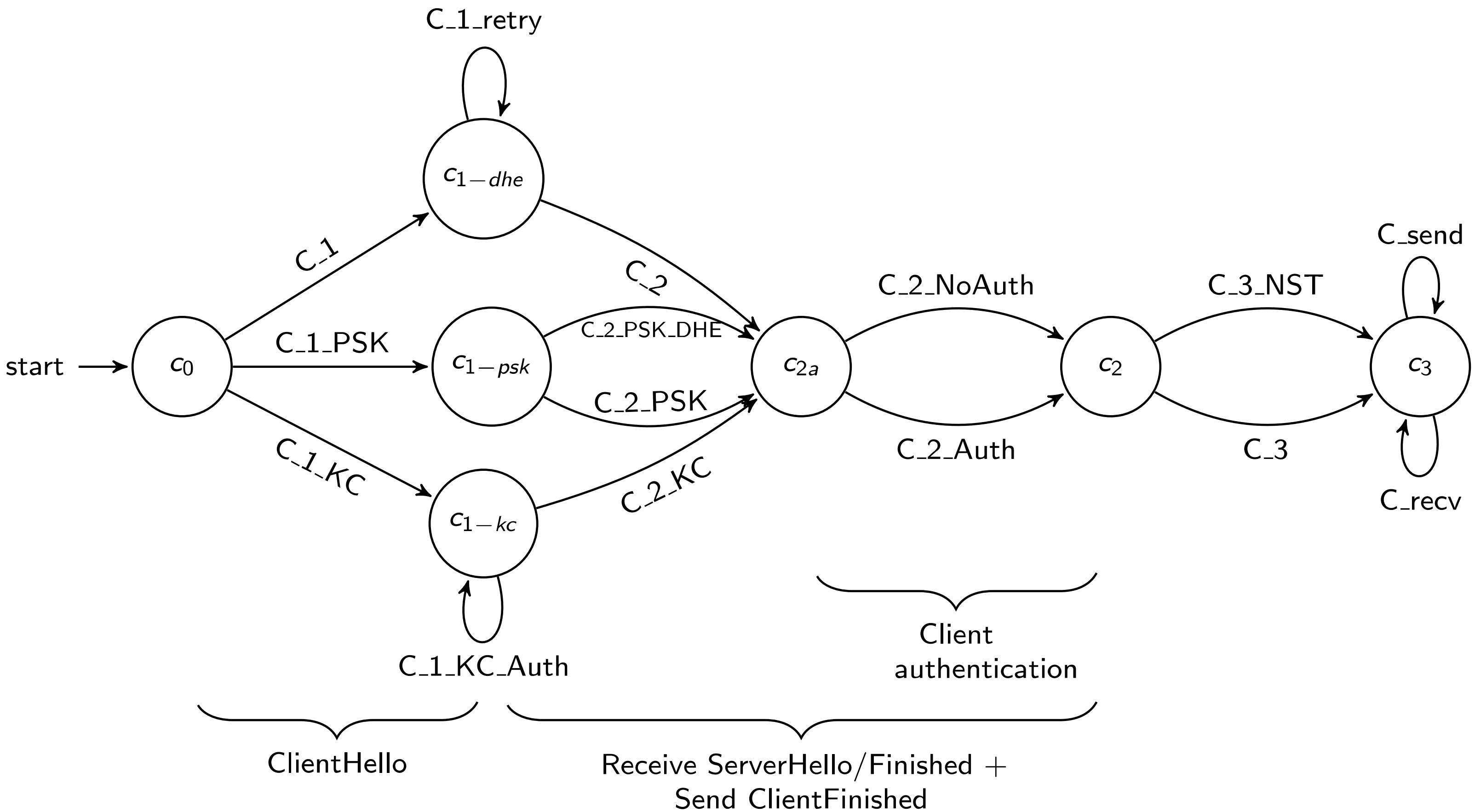
Thyla van der Merwe – Sam Scott – Marko Horvat – Jonathan Hoyland – Cas Cremers

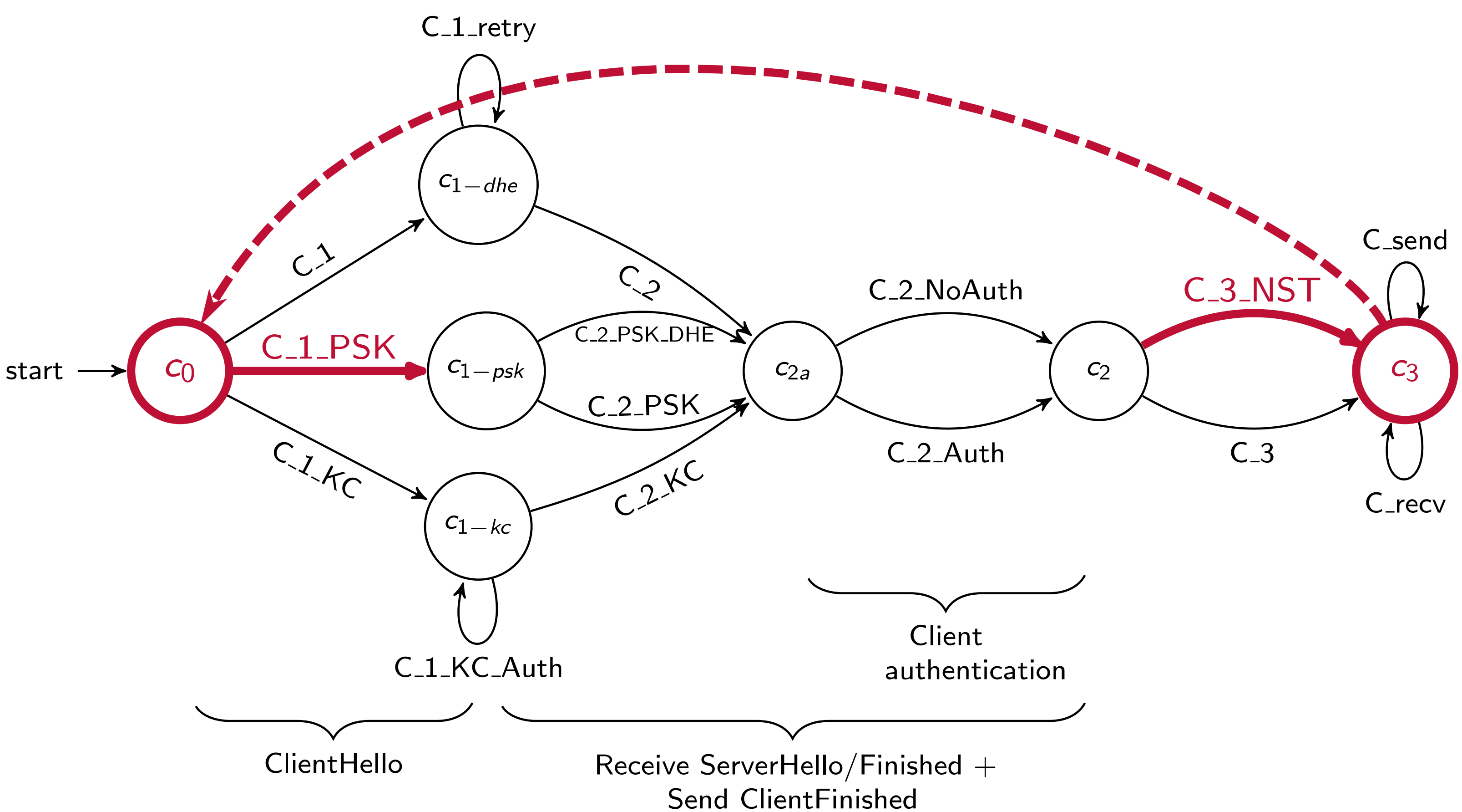
Step 1: Building a model



Step 1: Building a model



Step 1: Building a model



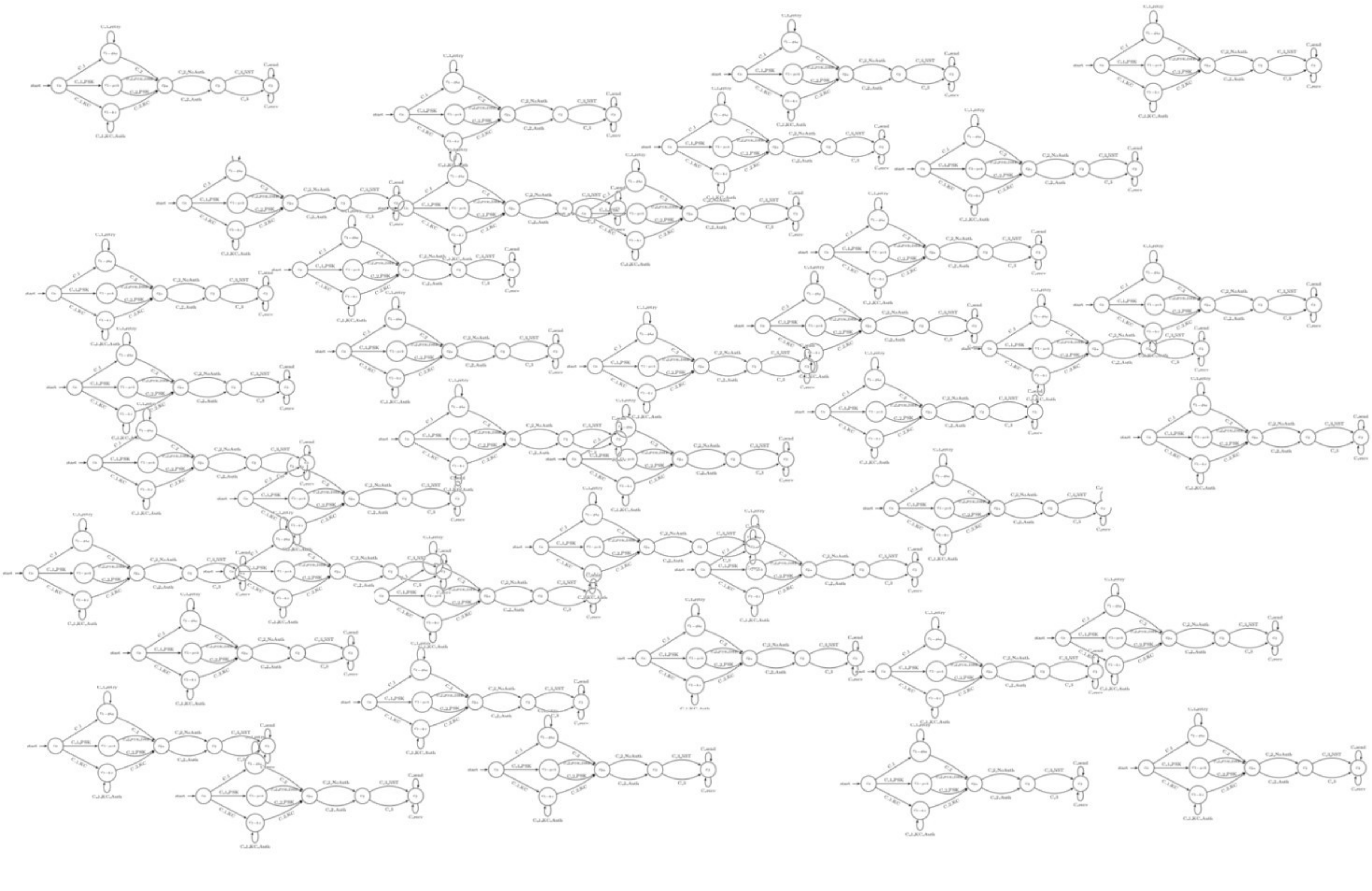
Step 2: Encoding security properties

secret_session_keys:
(1) „All actor peer role k #i.
(2) **SessionKey**(actor, peer, role, <k, 'authenticated'>)@i
(3) & not ((Ex #r. **RevLtk**(peer)@r & #r < #i)
| (Ex #r. **RevLtk**(actor)@r & #r < #i))
(4) ==> not Ex #j. **K**(k)@j“

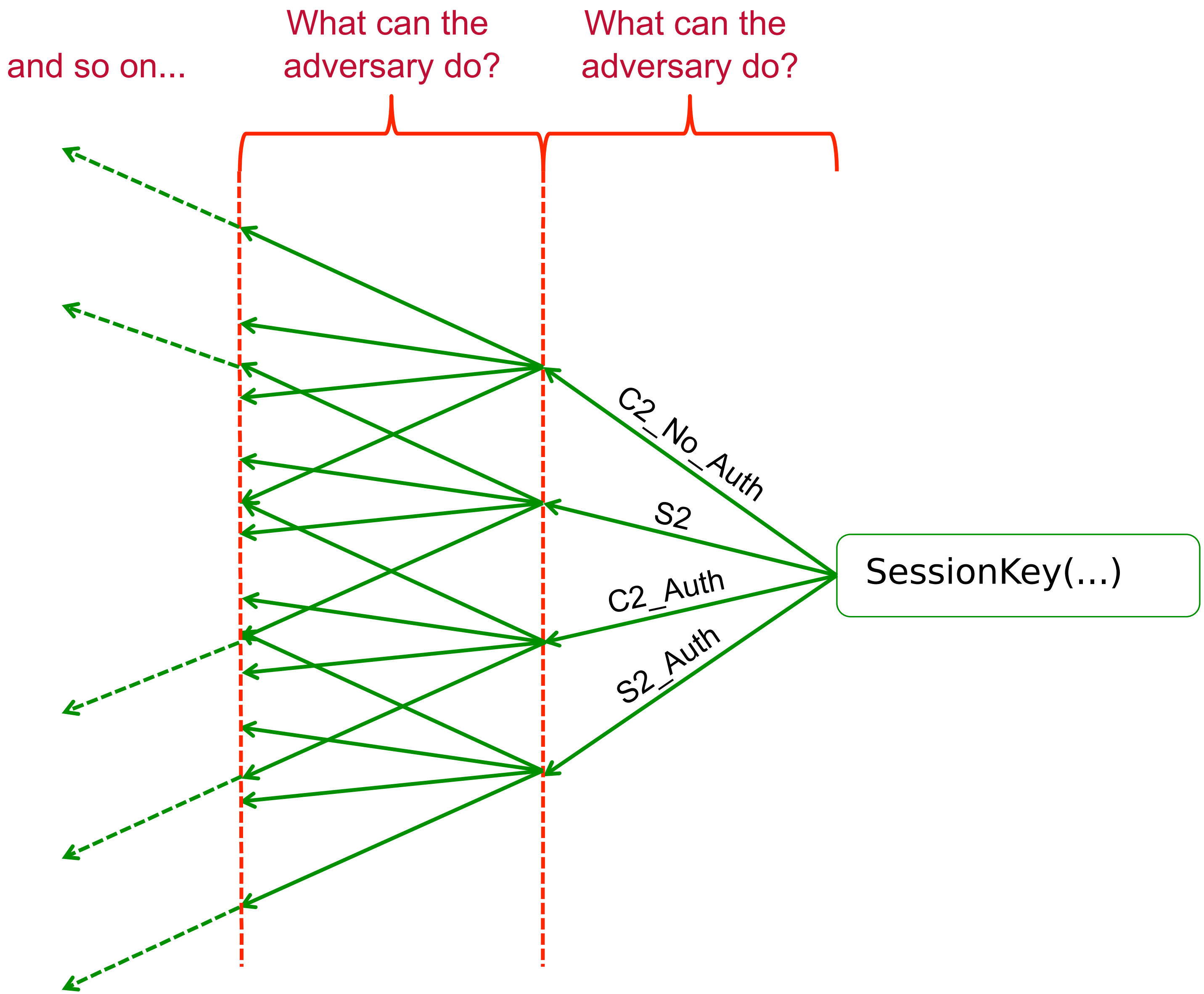
- This says…
  - For all possible values of variables on the first line (1)
  - if key k is accepted at time point i (2), and
  - the adversary has not revealed the long term keys of the actor or the peer before the key is accepted (3)
  - then the adversary cannot derive the key (4)

Want to show that this holds for all combinations of client, server, and adversary behaviours – ALL traces!

The adversary’s view

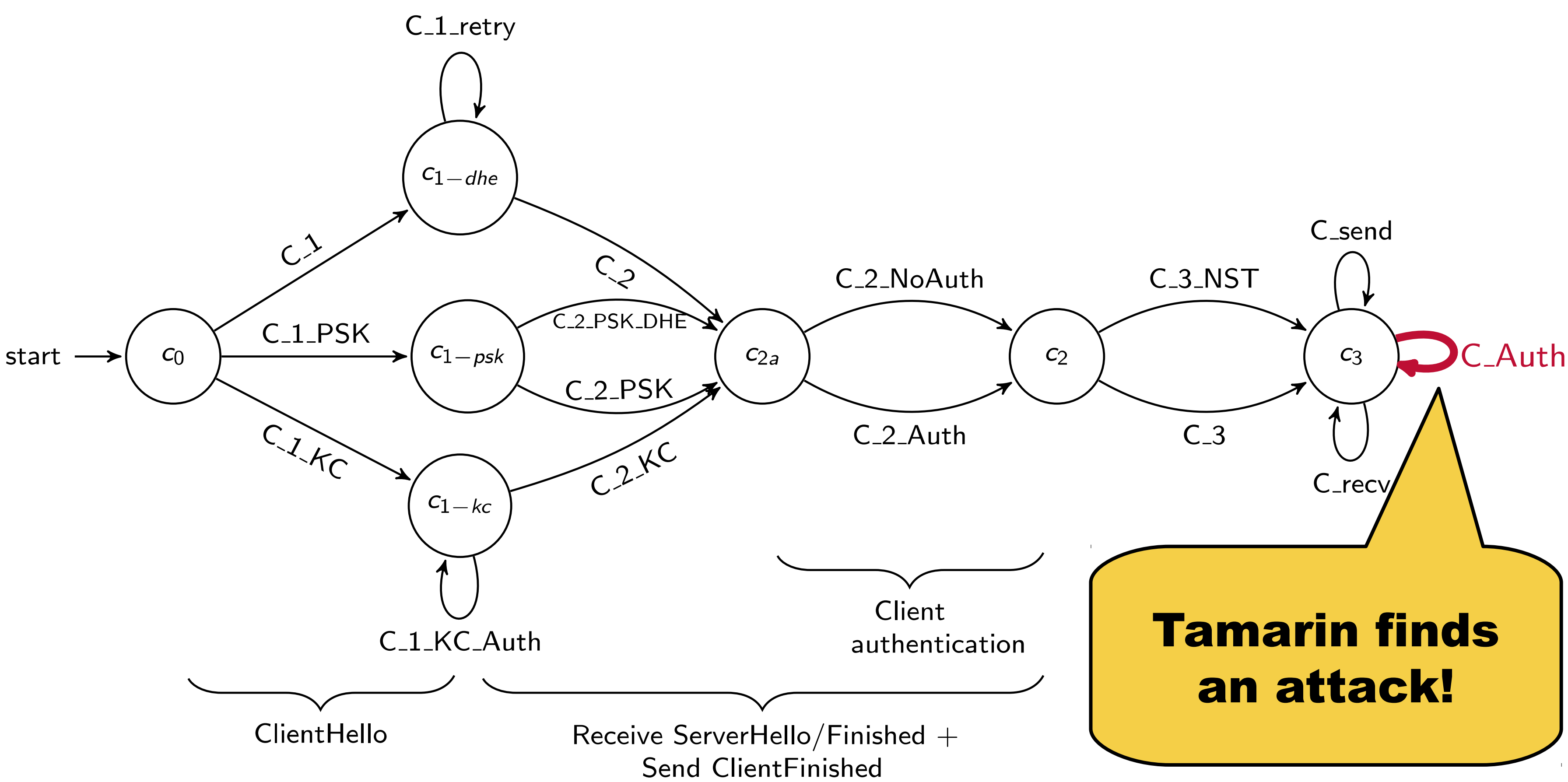


Step 3: Proving security properties



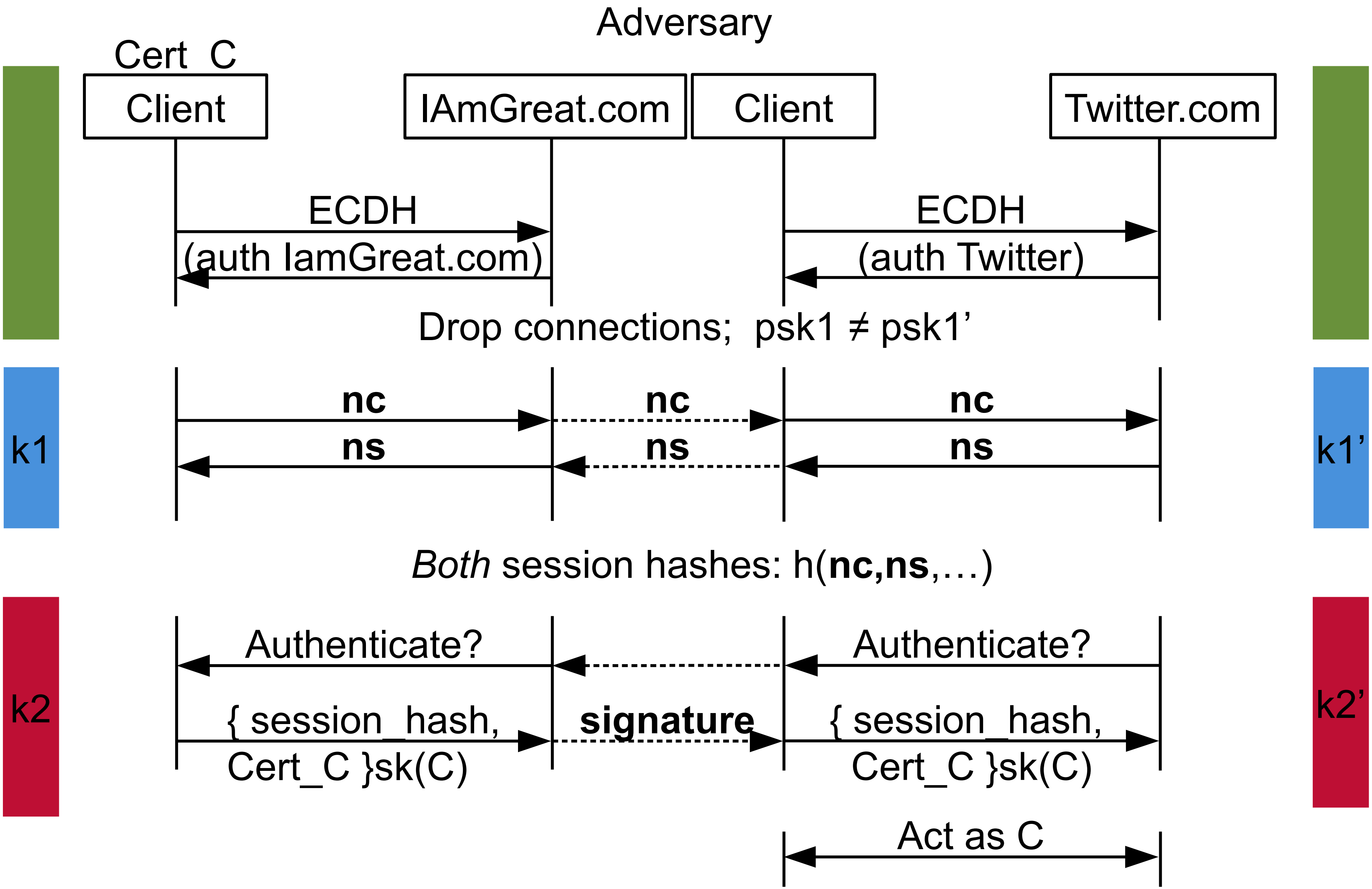
Revision 10 Initial results: looks good!

Attacking client authentication (revision 10+)



Problem 1

- Post-handshake client authentication looks good… until composed with everything else



Adversary
Cert_C
Client
IAmGreat.com
Client
Twitter.com
ECDH
(auth IamGreat.com)
ECDH
(auth Twitter)
Drop connections; psk1 ≠ psk1'
k1
nc
ns
nc
ns
nc
ns
k1'
Both session hashes: h(**nc**,**ns**,…)
k2
Authenticate?
Authenticate?
{ session_hash, Cert_C }sk(C)
signature
{ session_hash, Cert_C }sk(C)
k2'
Act as C

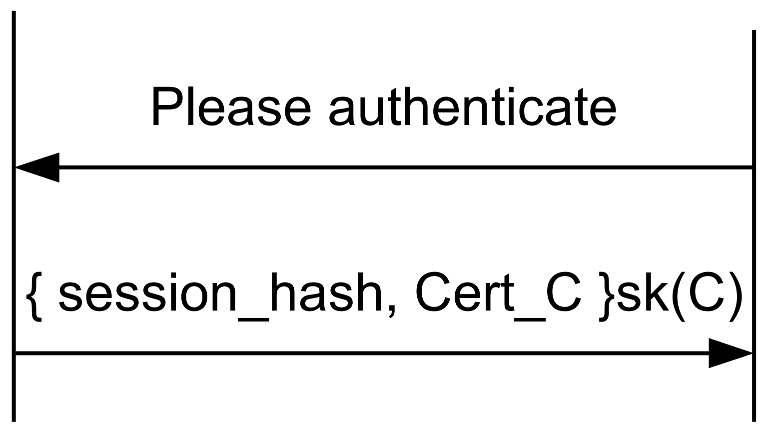
Observations

- Prime example of an attack that can arise because of the interaction of modes
- Very complex attack
  - requires **18 messages** to set up
  - involves 2 handshakes, 2 resumptions, 1 client authentication…
- Found by Tamarin
  - We didn't see it coming at all

Problem 2

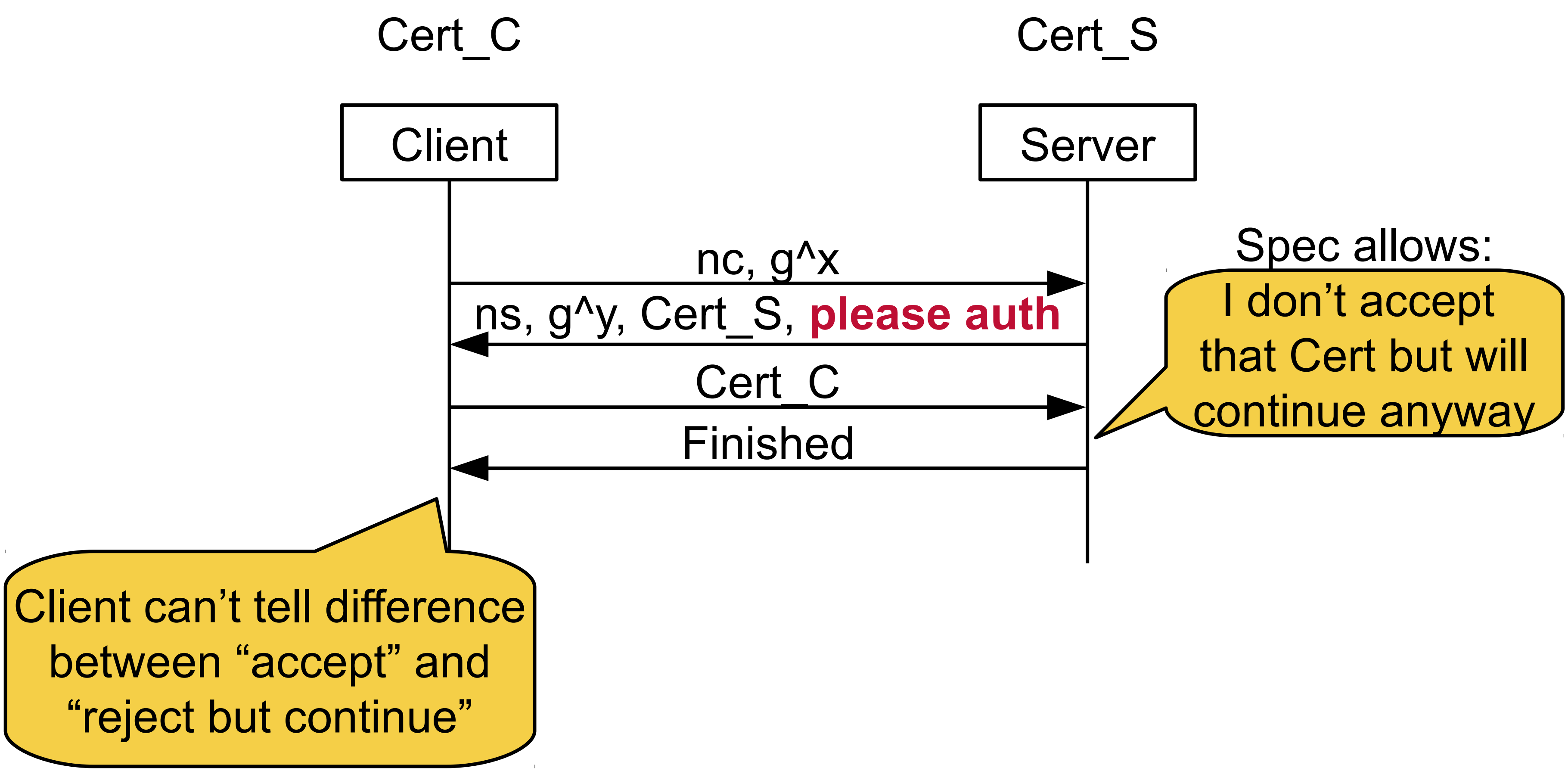
- Post-handshake client auth:
- While the server waits for a response, data can still go back and forth...

What does the client know?
Post-handshake client auth



Issue: server and client can still exchange data while server waits for post-handshake client authentication response.
At some point, server may receive the response and consider the connection to be mutually authenticated. However, the client has no way of knowing when or even if this happened.
(Needs to ask application layer!)

What does the client know? Mutual authentication mode



Awkward handshake

- Even if
  - The server asks for a certificate
  - The client provides it
  - The server sends/accepts subsequent traffic
- then the client can still not be sure that the server thinks the client is authenticated

Conclusions

- **TLS 1.3 and authentication**
  - Authentication is still complicated!
  - We became more involved in the process
    (we're now official contributors to the TLS 1.3 RFC)
- ***The future***
  - Real-world complexity remains challenging
  - Improving scope: scaling our algorithms
  - Improving accuracy: integrate more crypto insights

cas.cremers@cs.ox.ac.uk