Automated Analysis of TLS 1.3

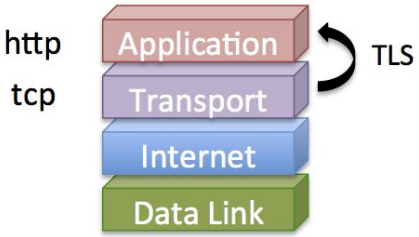
(How to Train Your Monkey)

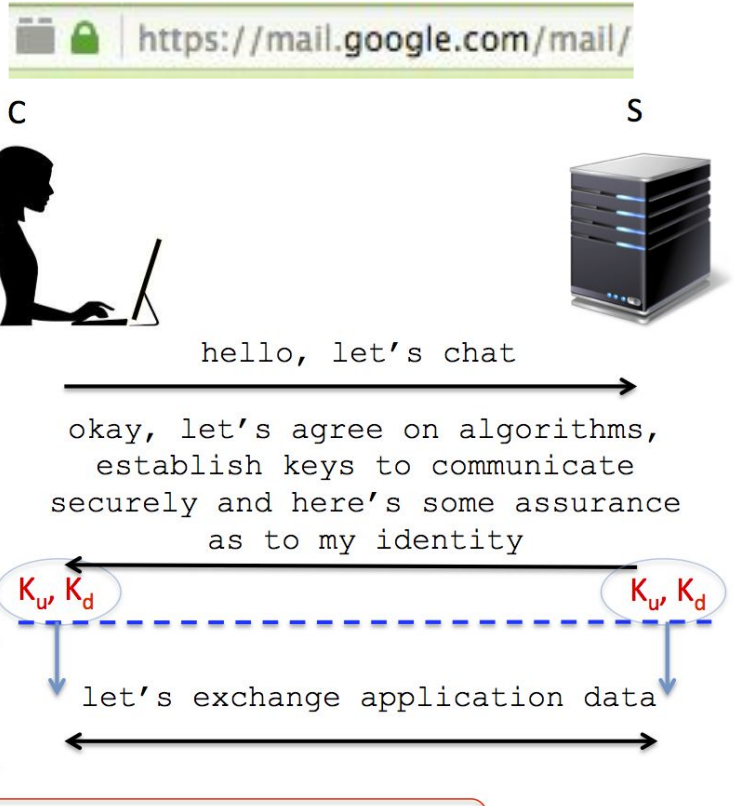
Thyla van der Merwe

2 February 2019

moz://a

What is TLS?


http
tcp
Application
Transport
Internet
Data Link
TLS


https://mail.google.com/mail/
C
S
hello, let's chat
okay, let's agree on algorithms, establish keys to communicate securely and here's some assurance as to my identity
K_u, K_d
K_u, K_d
let's exchange application data
Handshake protocol
Negotiate ciphersuite, authenticate entities and establish keys for record protocol
Record protocol
Provide confidentiality and authenticity of application data using keys established in the Handshake protocol

NETSCAPE

SSL 2.0 (1995) → SSL 3.0 (1996)

I E T F®

TLS 1.0 (1999) → TLS 1.1 (2006) → TLS 1.2 (2008)

NETSCAPE

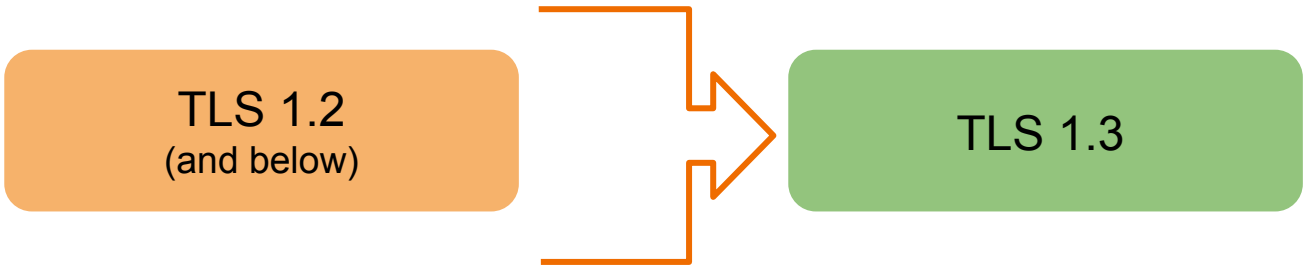
SSL 2.0 (1995) → SSL 3.0 (1996)

I E T F®

TLS 1.0 (1999) → TLS 1.1 (2006) → TLS 1.2 (2008)

(2018)

Why Something New?



Why Something New?

TLS 1.2
(and below)

TLS 1.3

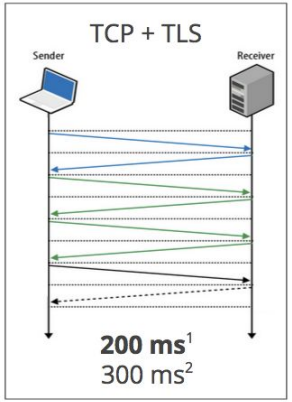
Why Something New?

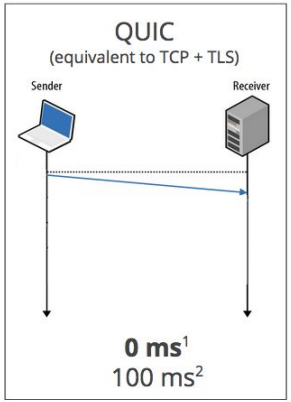
TLS 1.2
(and below)

weaknesses

TLS 1.3

Why Something New?



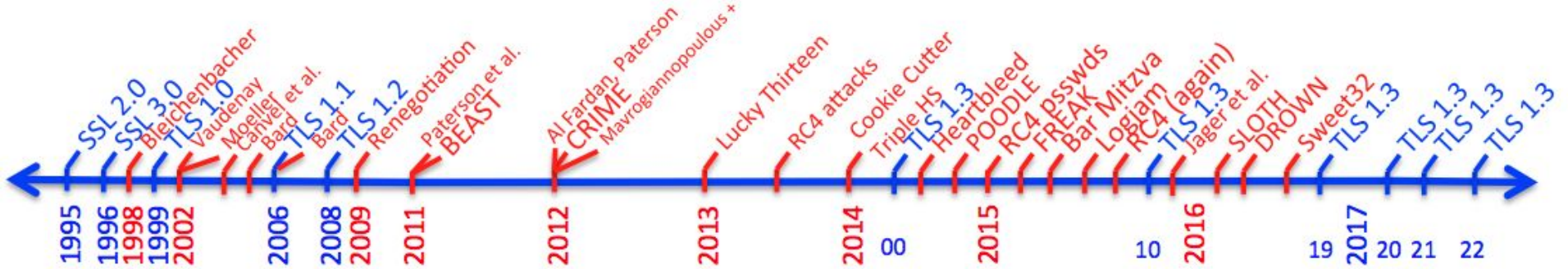


1. Repeat connection
2. Never talked to server before

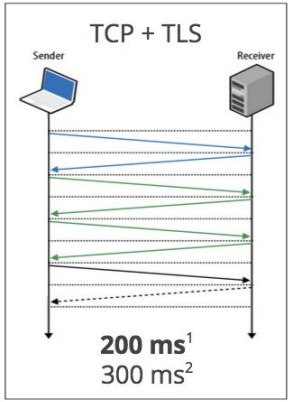
TLS 1.2
(and below)

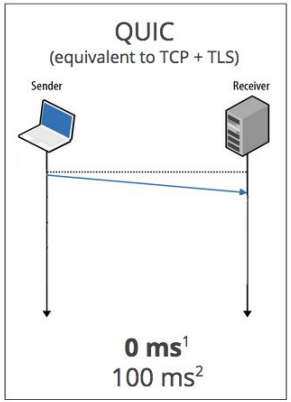
weaknesses

TLS 1.3



Why Something New?





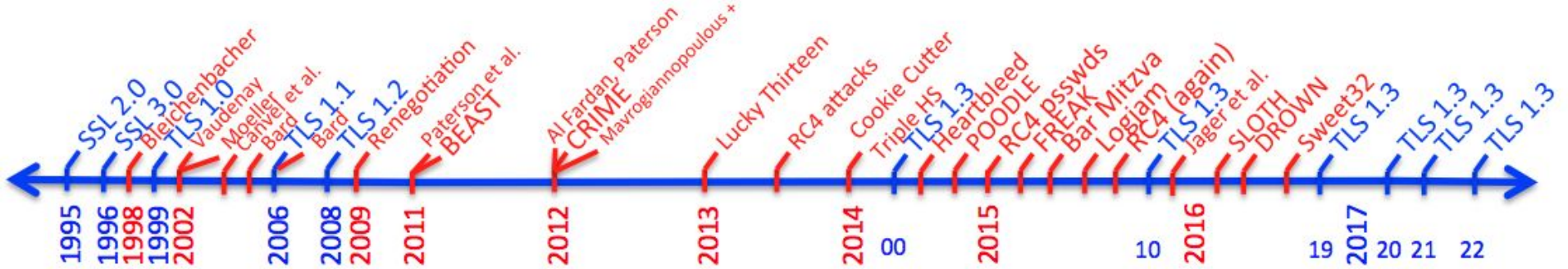
1. Repeat connection
2. Never talked to server before

improve efficiency

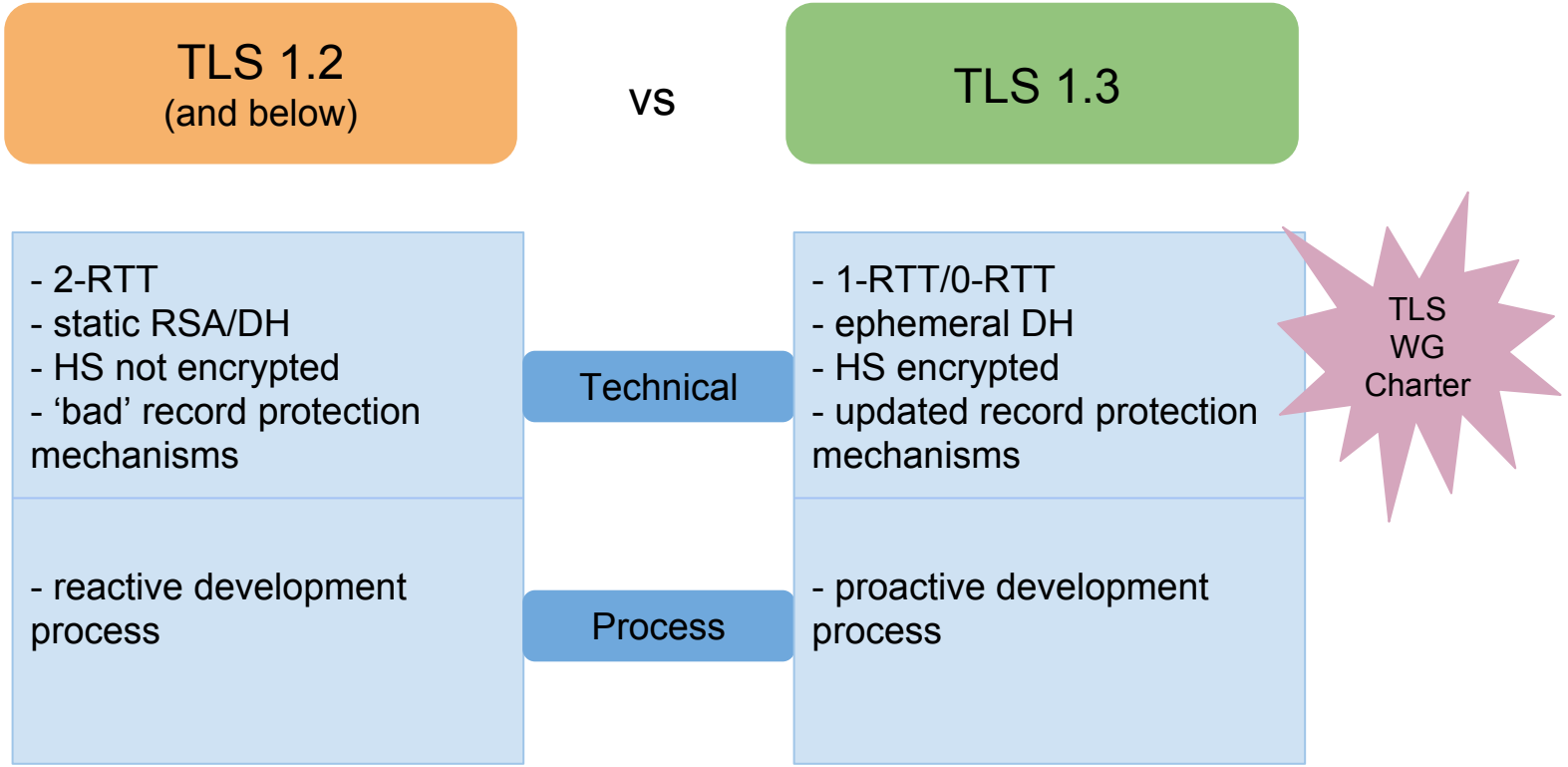
TLS 1.2
(and below)

TLS 1.3

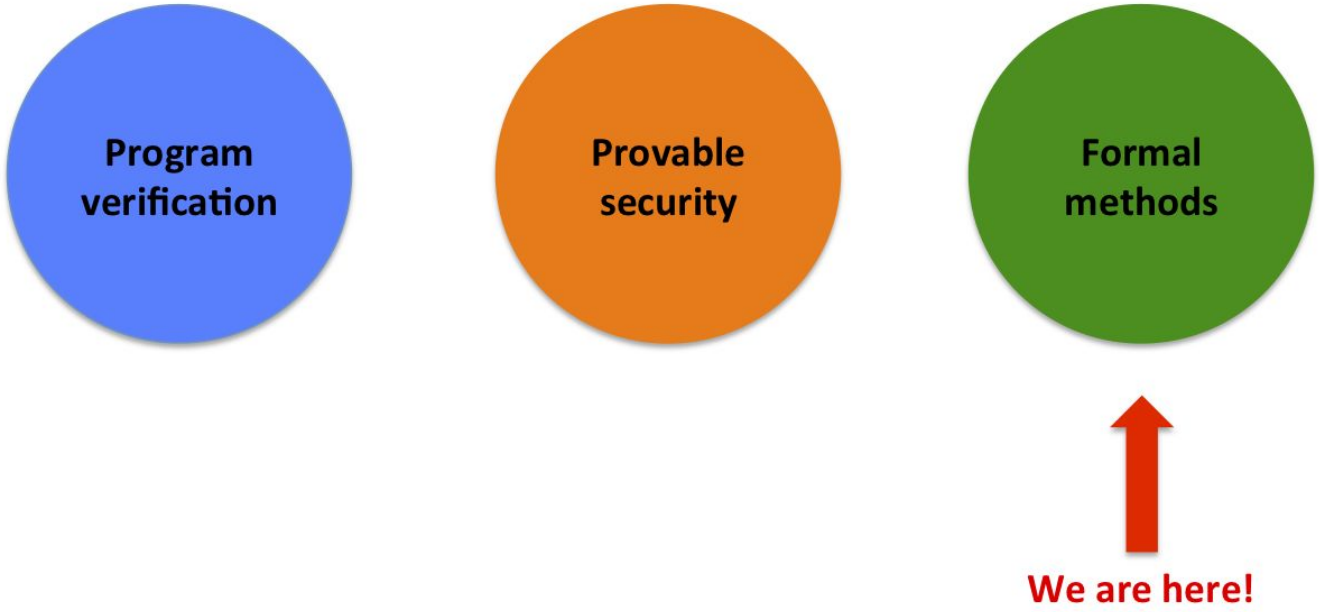
weaknesses



What's Changed?



Analysis Avenues for TLS 1.3



The Tamarin Prover

http://www.infsec.ethz.ch/research/software/tamarin.html

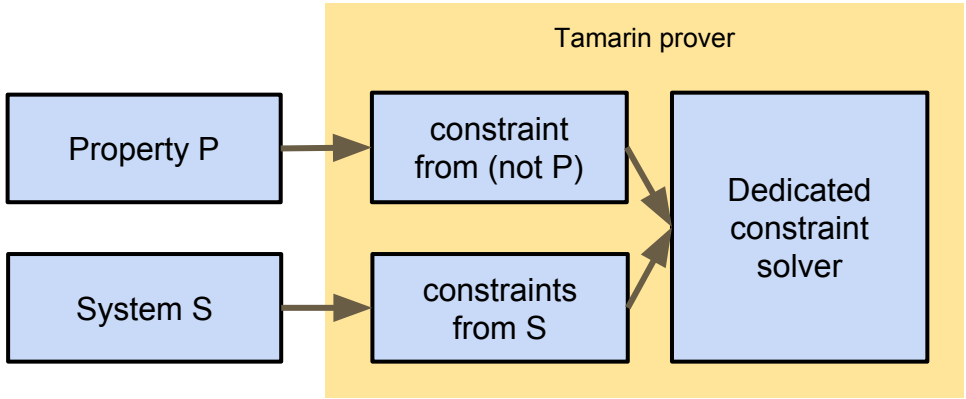
- What is it?
  - Automated tool for protocol analysis

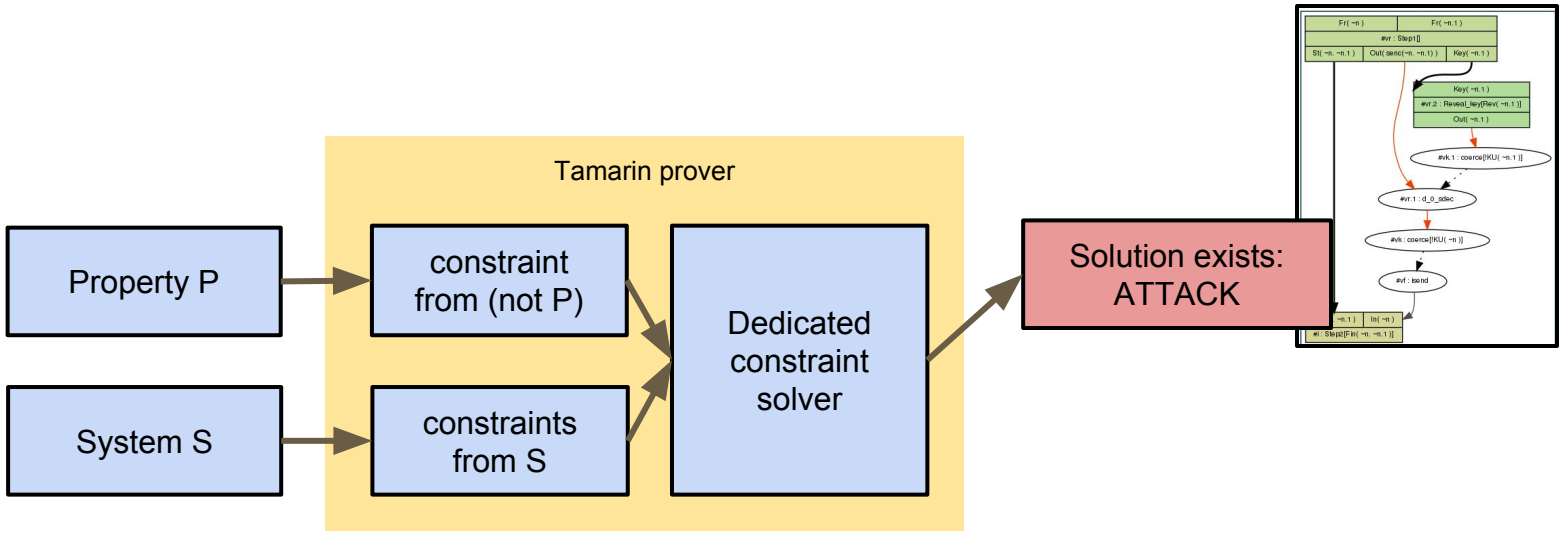
- How does it work?
  - For simple models/properties, prove automatically
  - Complex models require more interaction
  - A proof shows that a property holds in all possible combinations of honest actors and adversary behaviours!

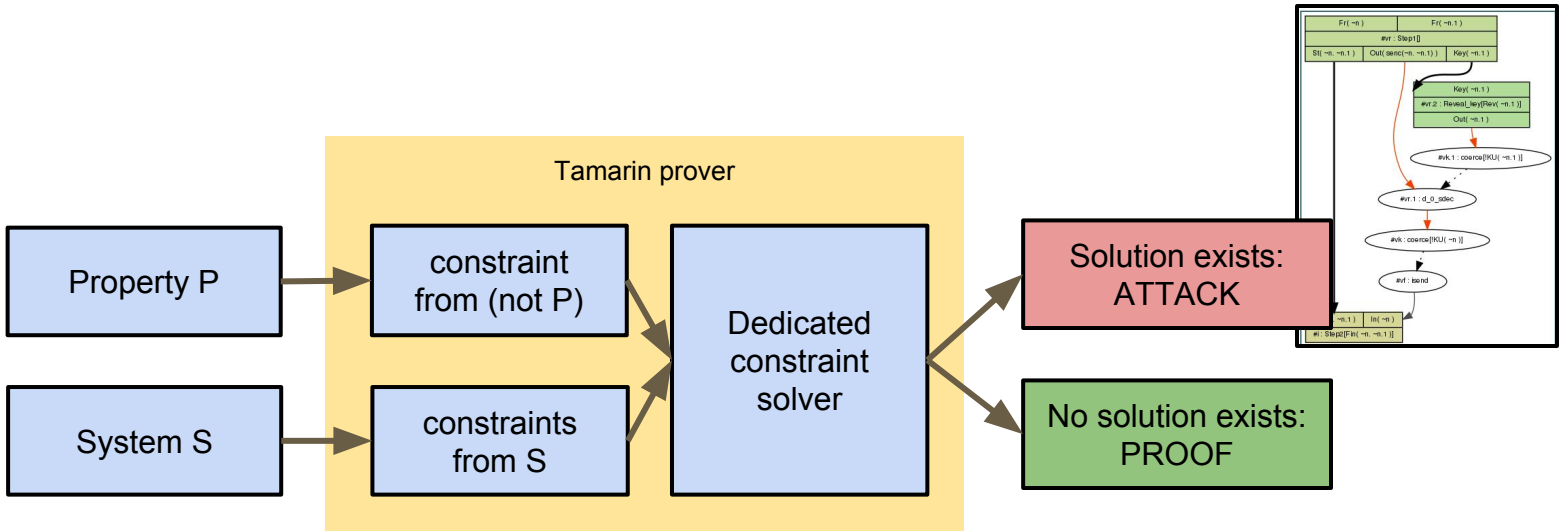
Constraint solver

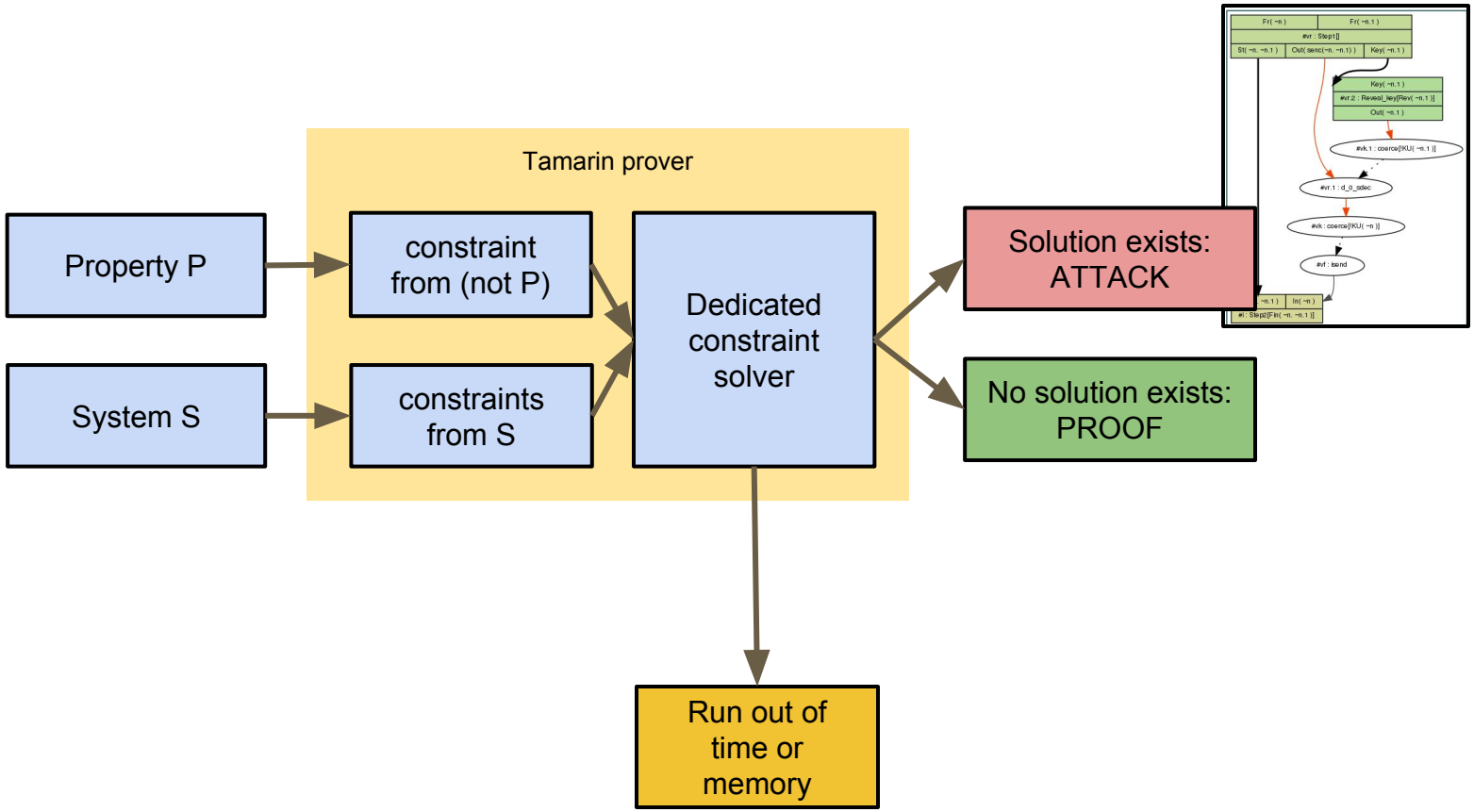
Theorem Prover

Constraint solver


Tamarin prover
Property P
System S
constraint from (not P)
constraints from S
Dedicated constraint solver


Tamarin prover
Property P
System S
constraint from (not P)
constraints from S
Dedicated constraint solver
Solution exists: ATTACK


Tamarin prover
Property P
System S
constraint from (not P)
constraints from S
Dedicated constraint solver
Solution exists: ATTACK
No solution exists: PROOF


Tamarin prover
Property P
System S
constraint from (not P)
constraints from S
Dedicated constraint solver
Solution exists: ATTACK
No solution exists: PROOF
Run out of time or memory

Tamarin prover

Property P

System S

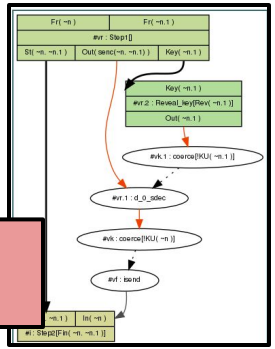
constraint from (not P)

constraints from S

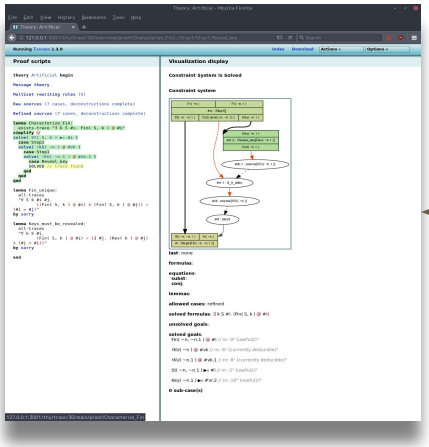
Dedicated constraint solver

Solution exists: ATTACK

No solution exists: PROOF



Run out of time or memory



Interactive mode
Inspect partial proof

Tamarin prover

Property P

System S

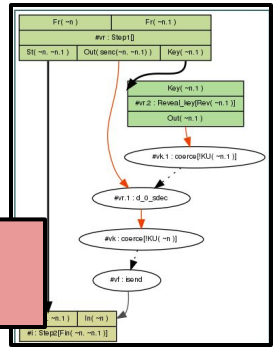
constraint from (not P)

constraints from S

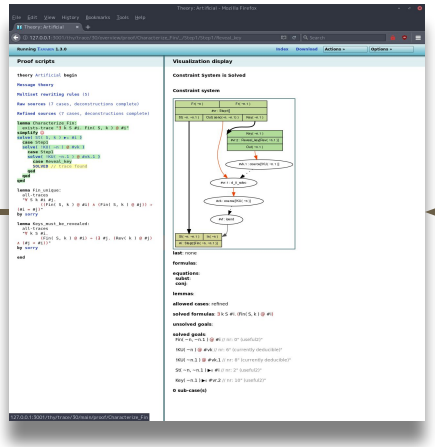
Dedicated constraint solver

Solution exists: ATTACK

No solution exists: PROOF



Run out of time or memory



Provide **hints** for the prover (e.g. invariants)

Interactive mode
Inspect partial proof

Specifying Protocols

Rewrite rules that specify transition system

rule name: LHS --[actions]-> RHS

rule name: LHS --[actions]-> RHS

```
rule my_protocol_step2:
   [ In( m1 ),  State( ThreadID, `state1`, previousData ) ]
      --[ Accepted( ThreadID, k) ]->
   [ Out( m2 ), State( ThreadID, `state2`, newData ) ]
```

premises (LHS)

rule name: LHS --[actions]-> RHS

```
rule my_protocol_step2:
   [ In( m1 ),  State( ThreadID, `state1`, previousData ) ]
      --[ Accepted( ThreadID, k) ]->
   [ Out( m2 ), State( ThreadID, `state2`, newData ) ]
```

premises (LHS)

conclusions (RHS)

rule name: LHS --[actions]-> RHS

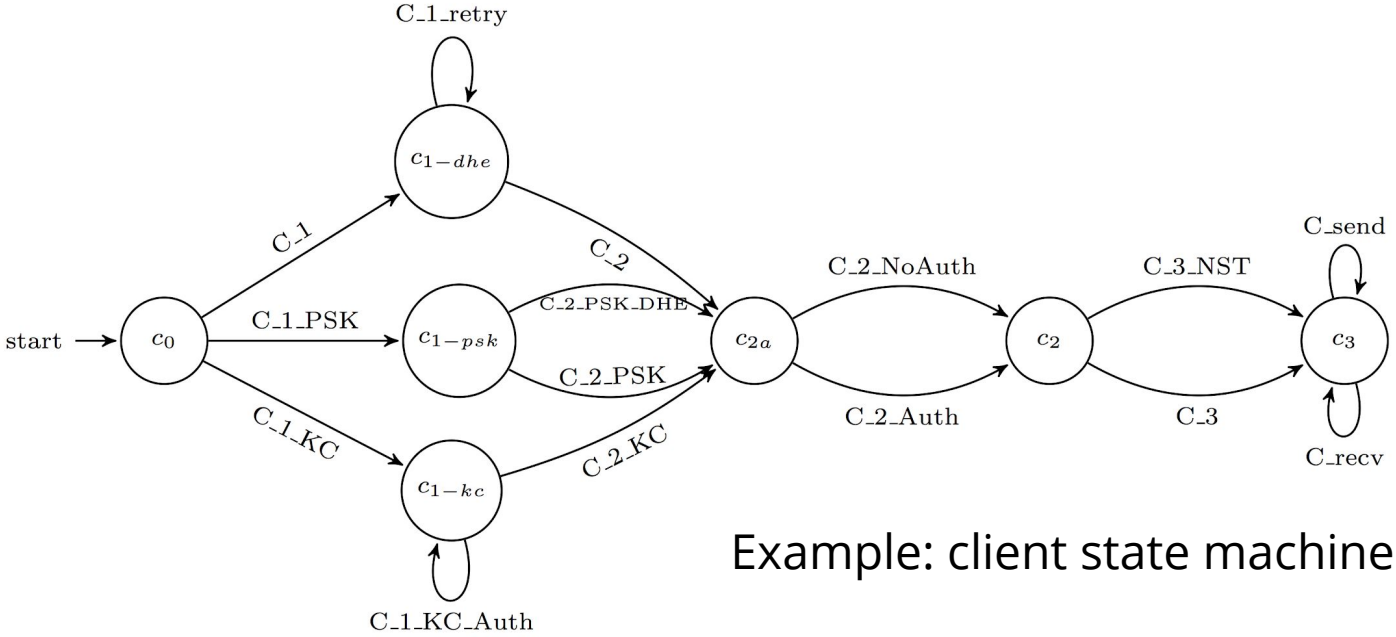
```
rule my_protocol_step2:
   [ In( m1 ),  State( ThreadID, `state1`, previousData ) ]
      --[ Accepted( ThreadID, k) ]->
   [ Out( m2 ), State( ThreadID, `state2`, newData ) ]
```

premises (LHS)

actions

conclusions (RHS)

rules --> state machine



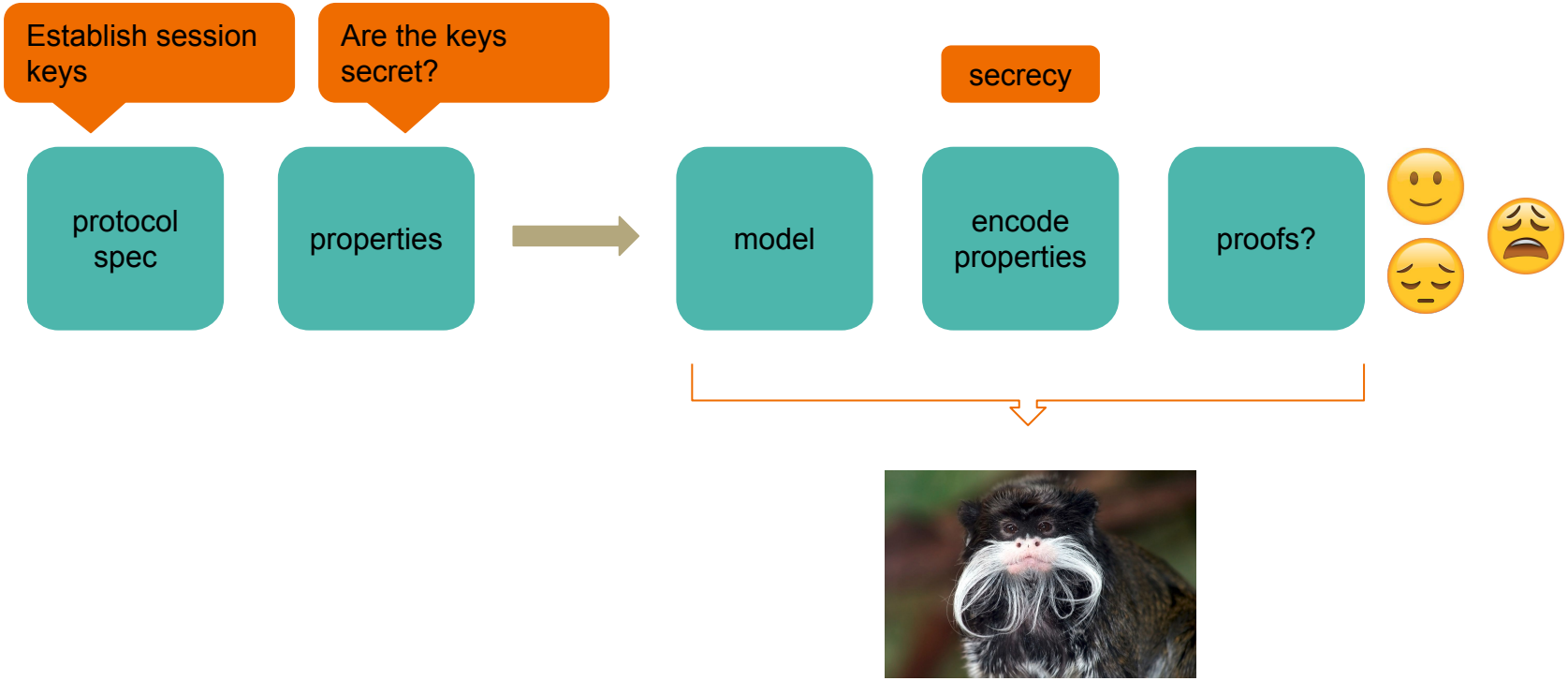
Example: client state machine

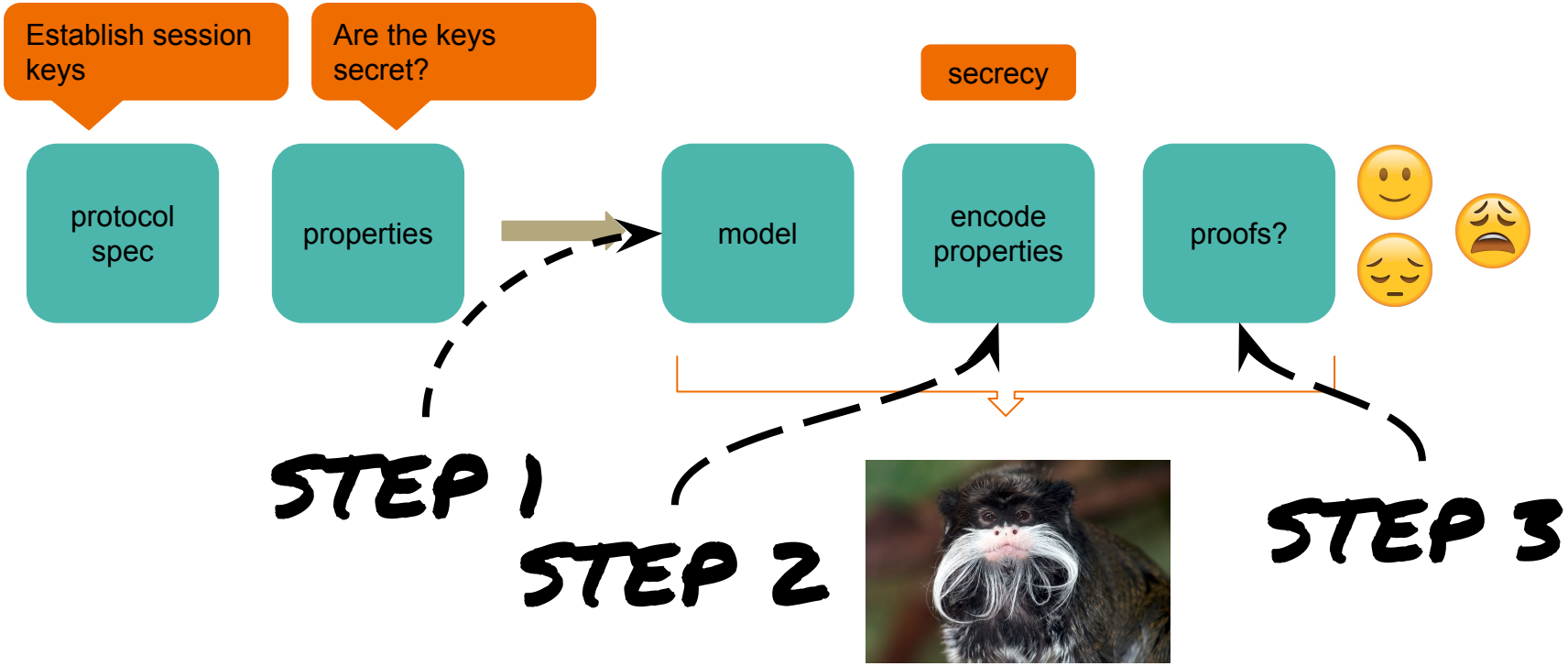
Rules correspond to edges

Analysing TLS 1.3

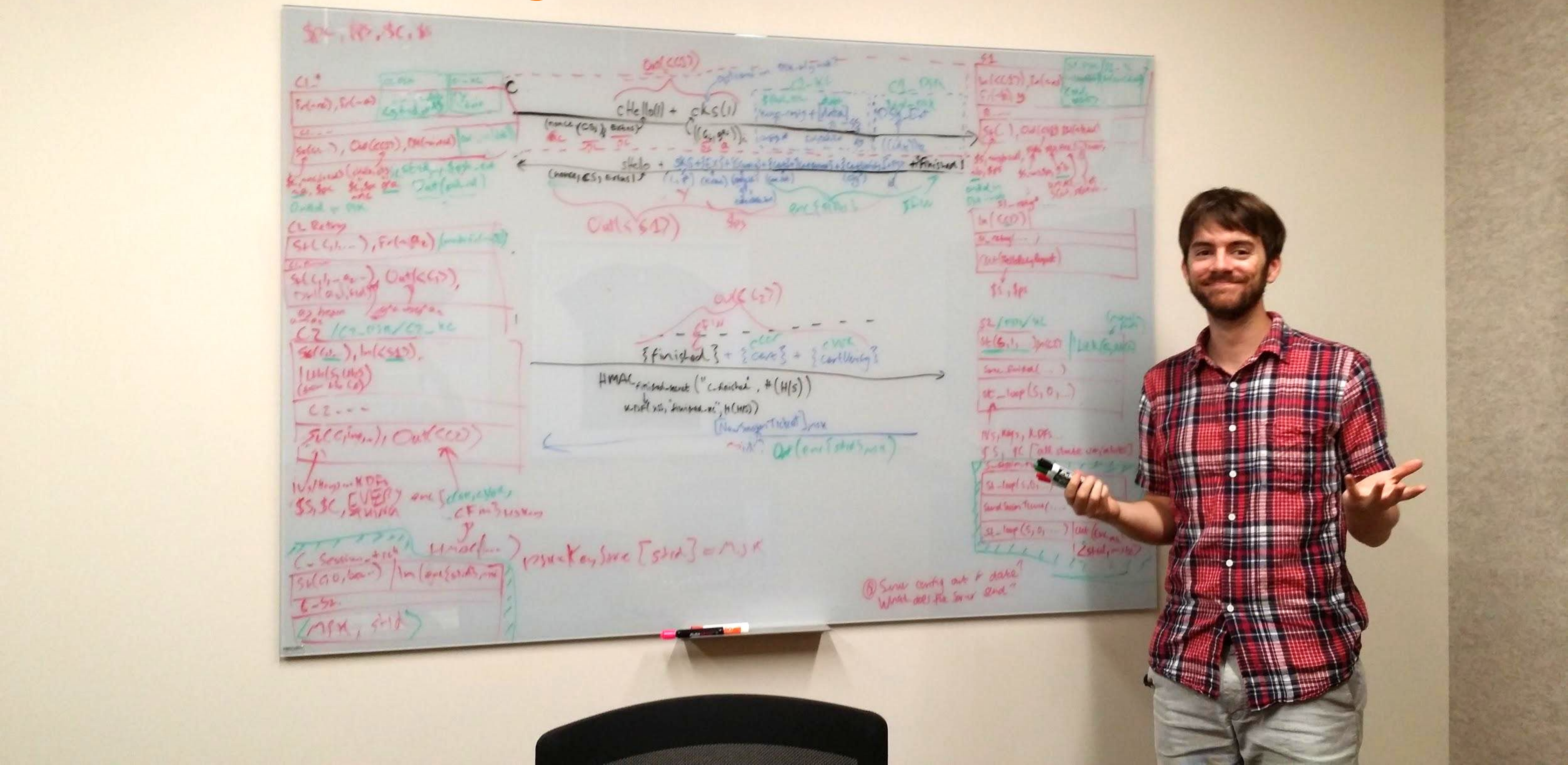
- We built a model of the TLS 1.3 specification, drafts 10 and 21
- We wanted to verify the main security properties of the TLS 1.3
  - secrecy of session keys
  - unilateral and mutual authentication
- We found an attack against draft 10+ → provided feedback to the IETF and we helped to fix the specification
- Our modeling of draft 21 shows that the logical core of TLS 1.3 looks sound!

Joint work with Cas Cremers, Marko Horvat, Jonathan Hoyland and Sam Scott.


Establish session keys
Are the keys secret?
secrecy
protocol spec
properties
model
encode properties
proofs?

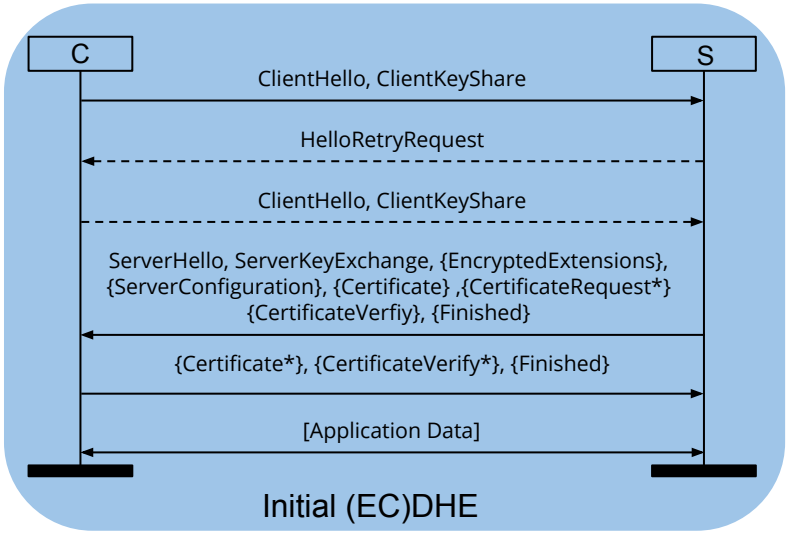

Establish session keys
Are the keys secret?
secrecy
protocol spec
properties
model
encode properties
proofs?
STEP 1
STEP 2
STEP 3

STEP 1: Building the Model

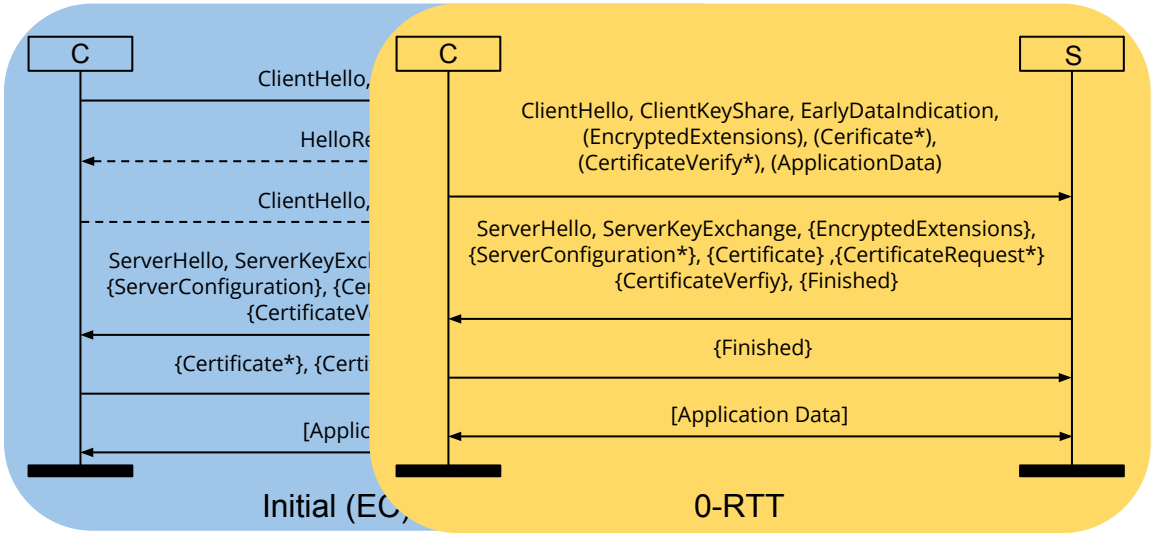


- Encode honest party and adversary actions as Tamarin rules
- Honest client and server rules correspond to flights of messages
- Rules transition protocol from one state to the next

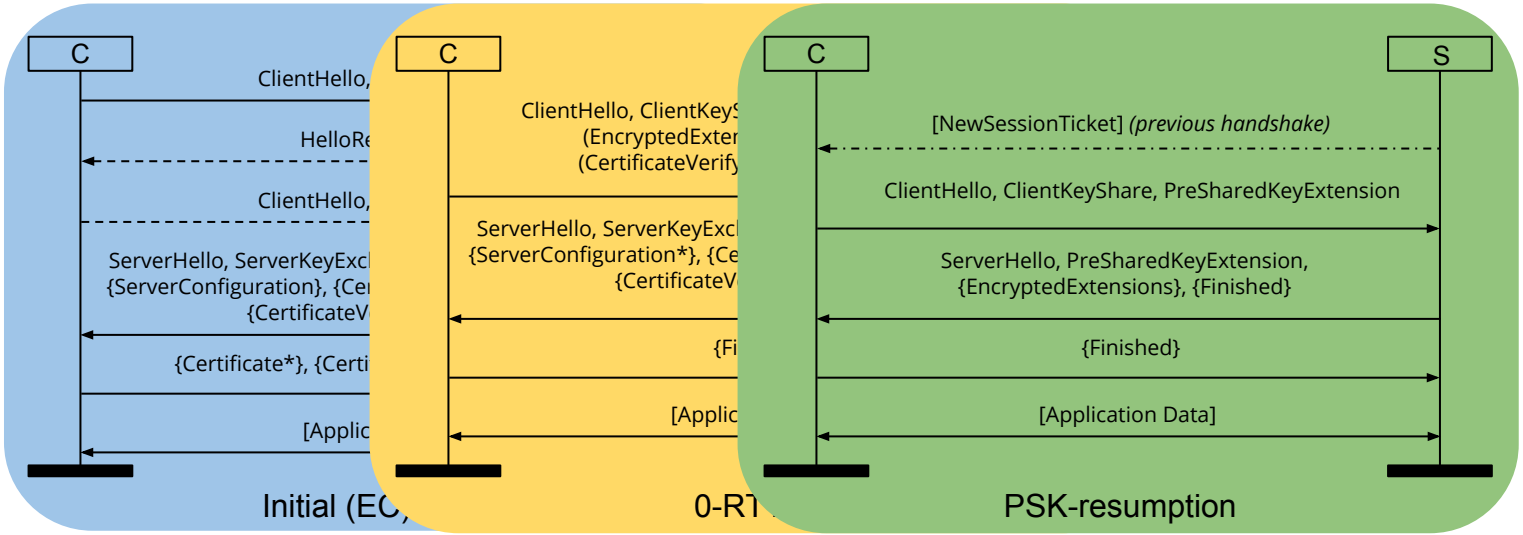
- Encode honest party and adversary actions as Tamarin rules
- Honest client and server rules correspond to flights of messages
- Rules transition protocol from one state to the next



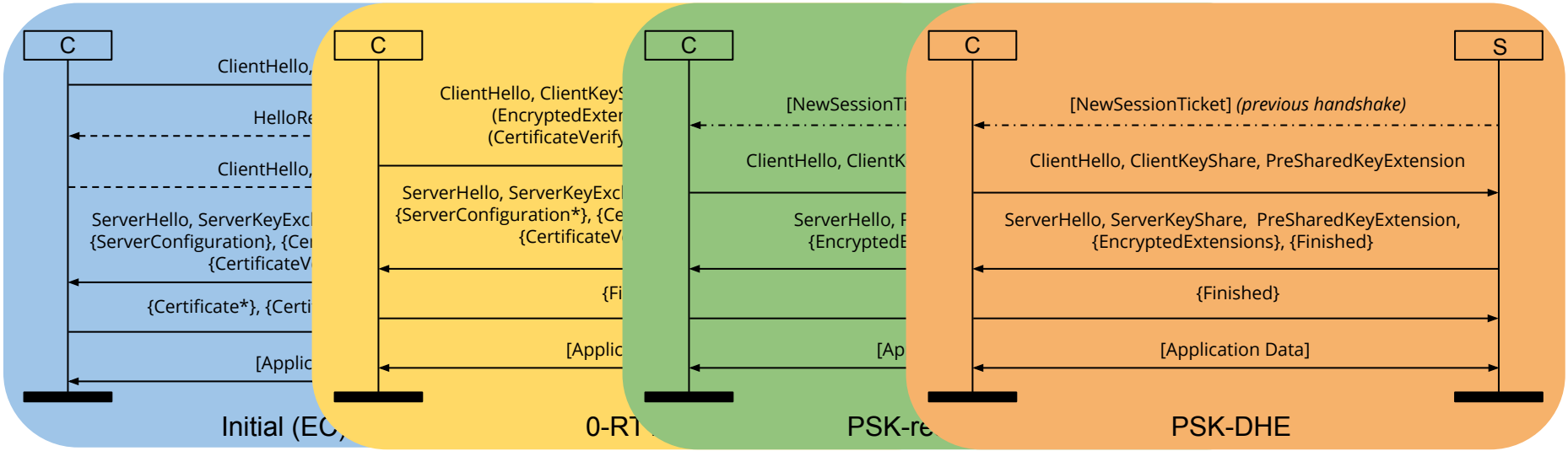
- Encode honest party and adversary actions as Tamarin rules
- Honest client and server rules correspond to flights of messages
- Rules transition protocol from one state to the next



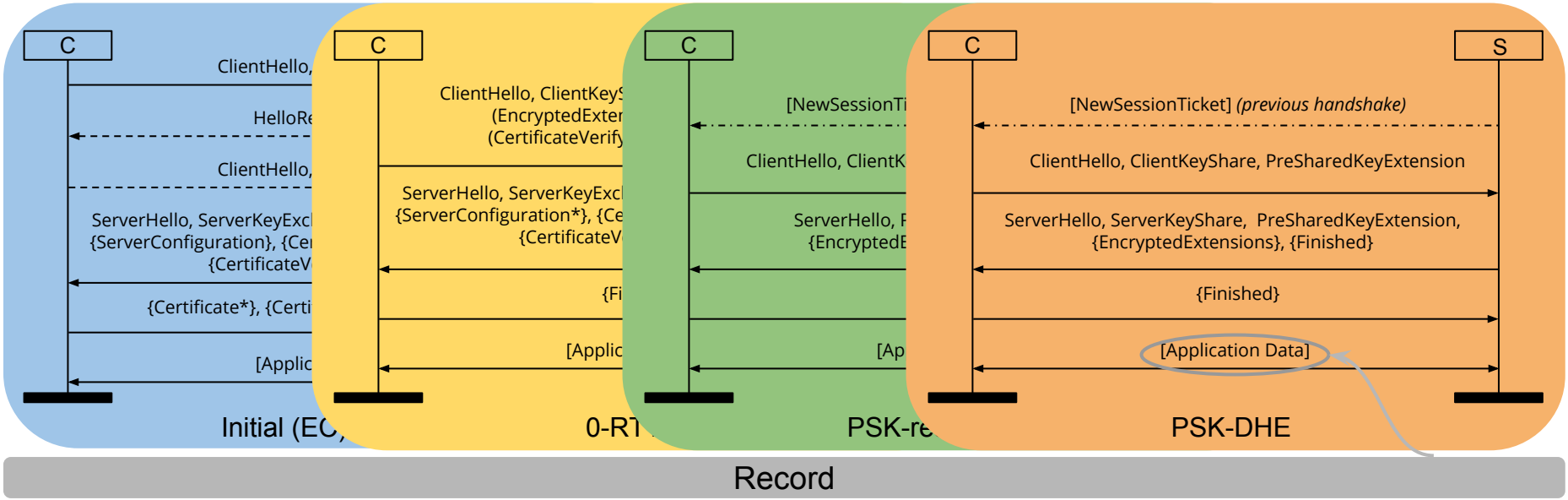
- Encode honest party and adversary actions as Tamarin rules
- Honest client and server rules correspond to flights of messages
- Rules transition protocol from one state to the next



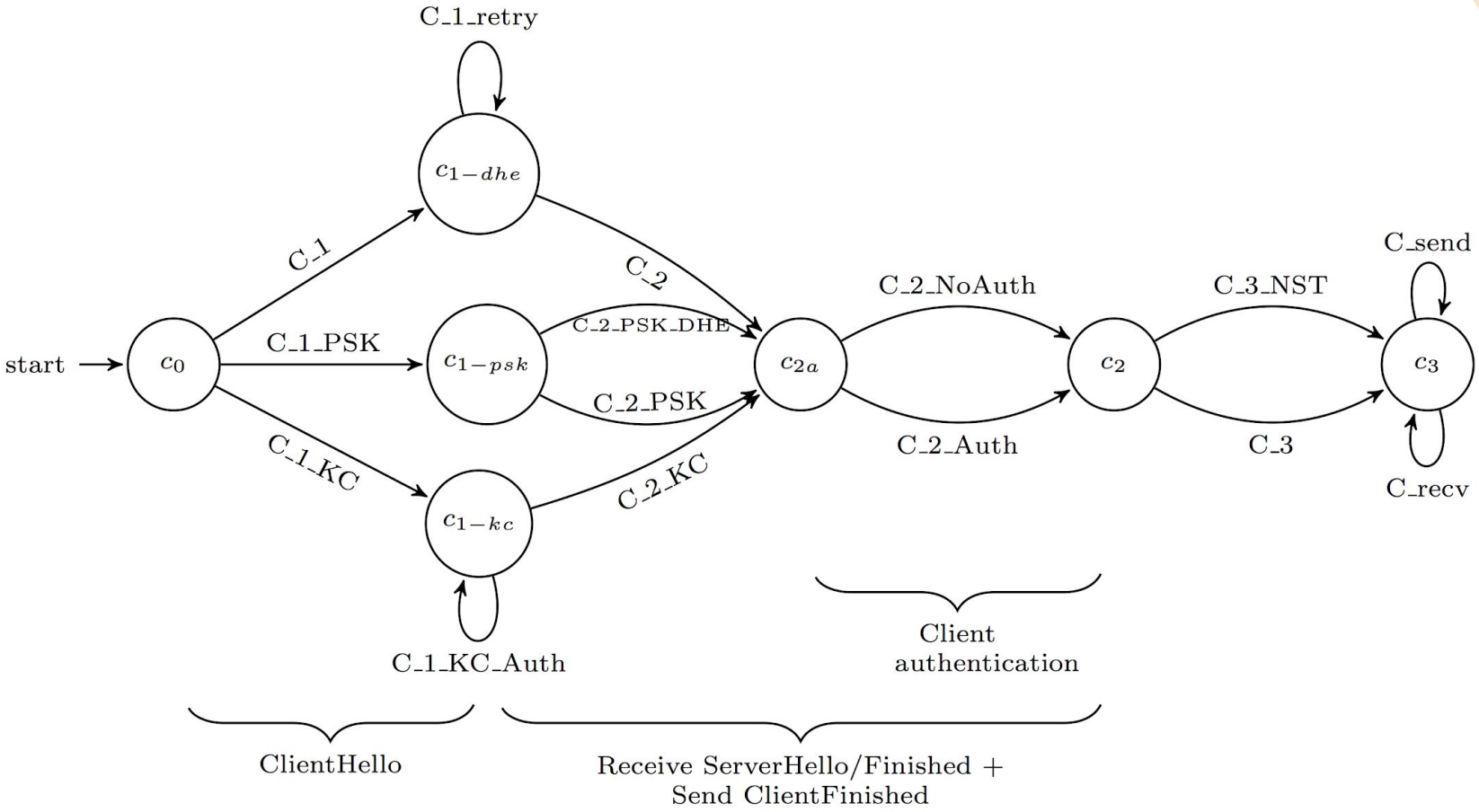
- Encode honest party and adversary actions as Tamarin rules
- Honest client and server rules correspond to flights of messages
- Rules transition protocol from one state to the next

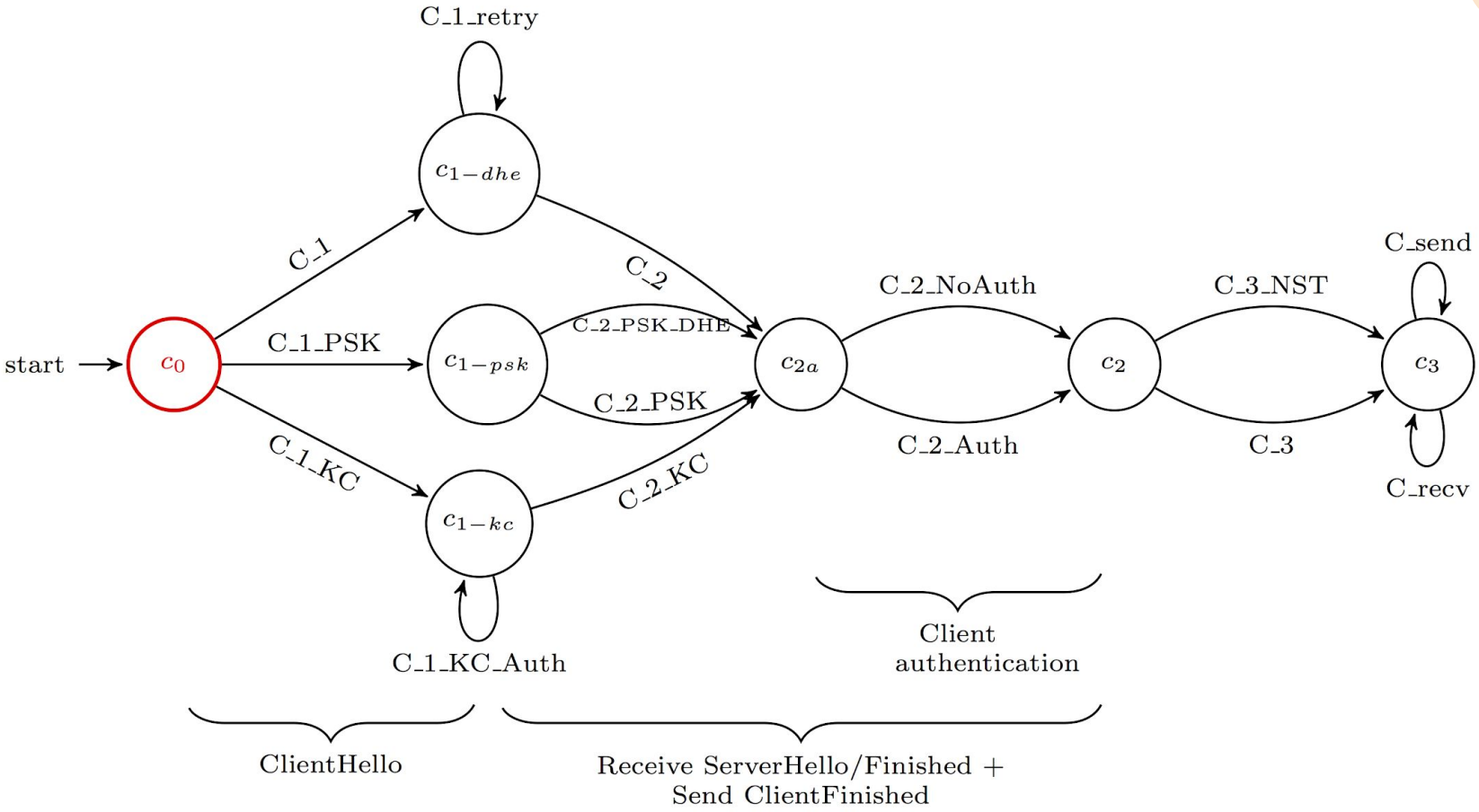


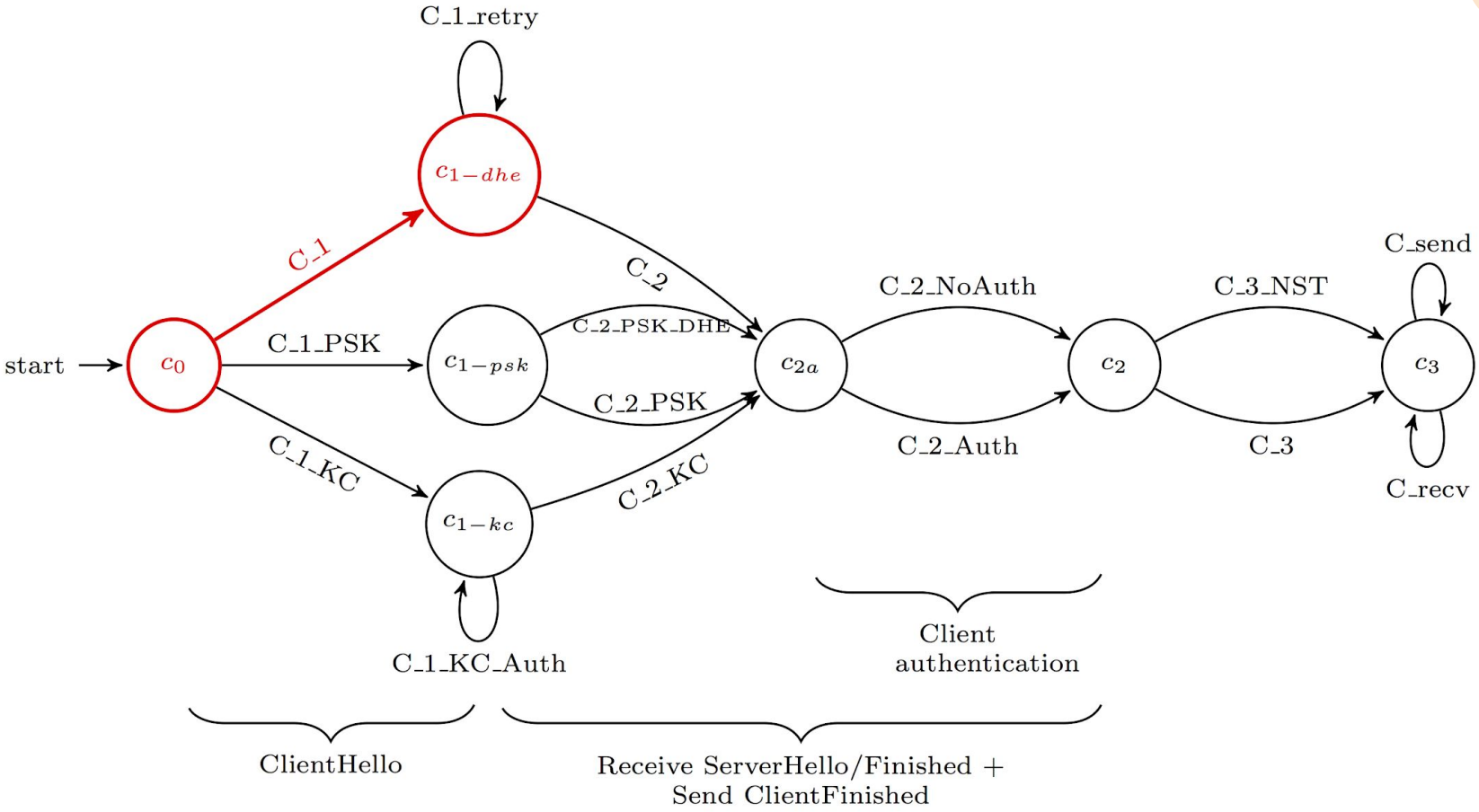
- Encode honest party and adversary actions as Tamarin rules
- Honest client and server rules correspond to flights of messages
- Rules transition protocol from one state to the next

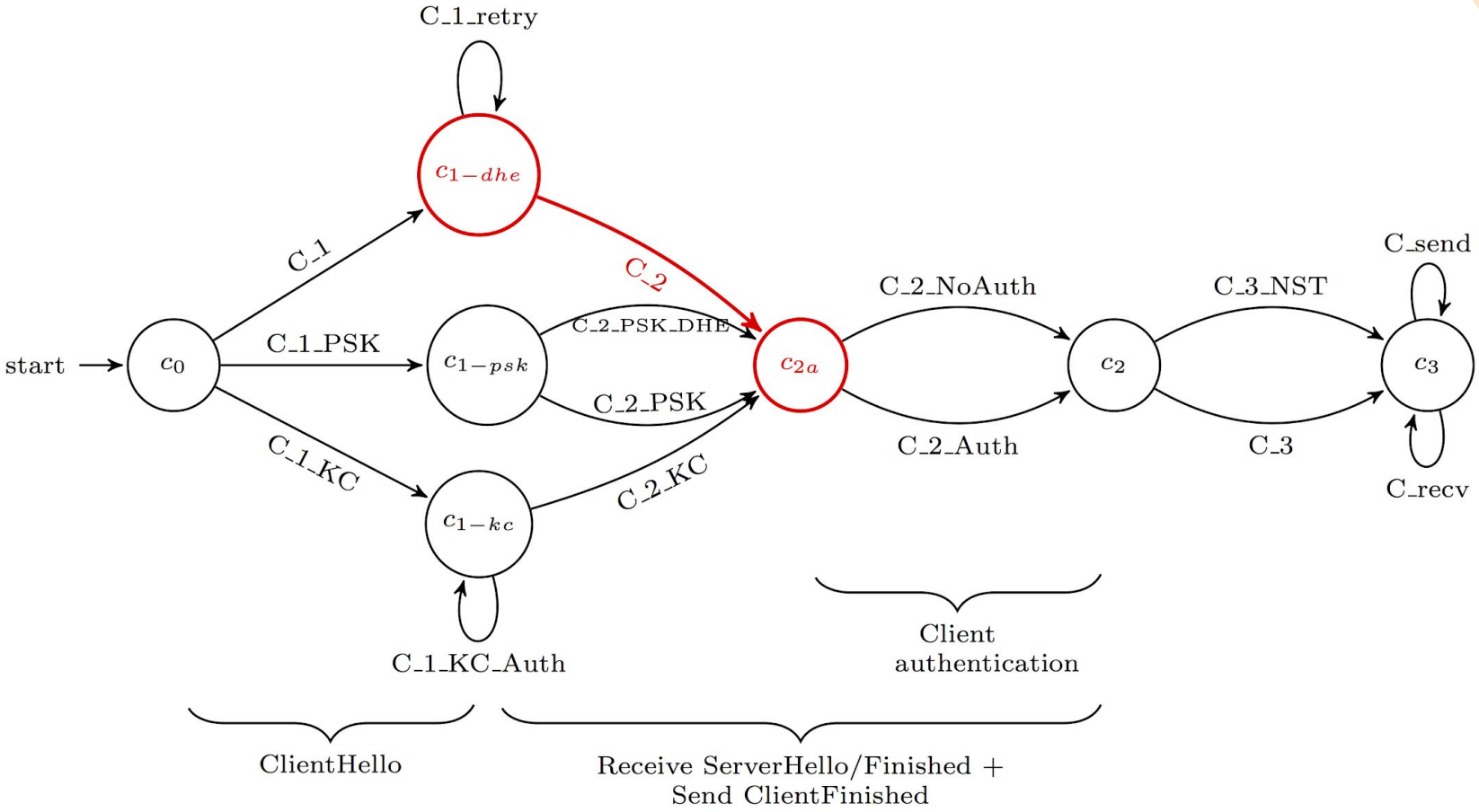


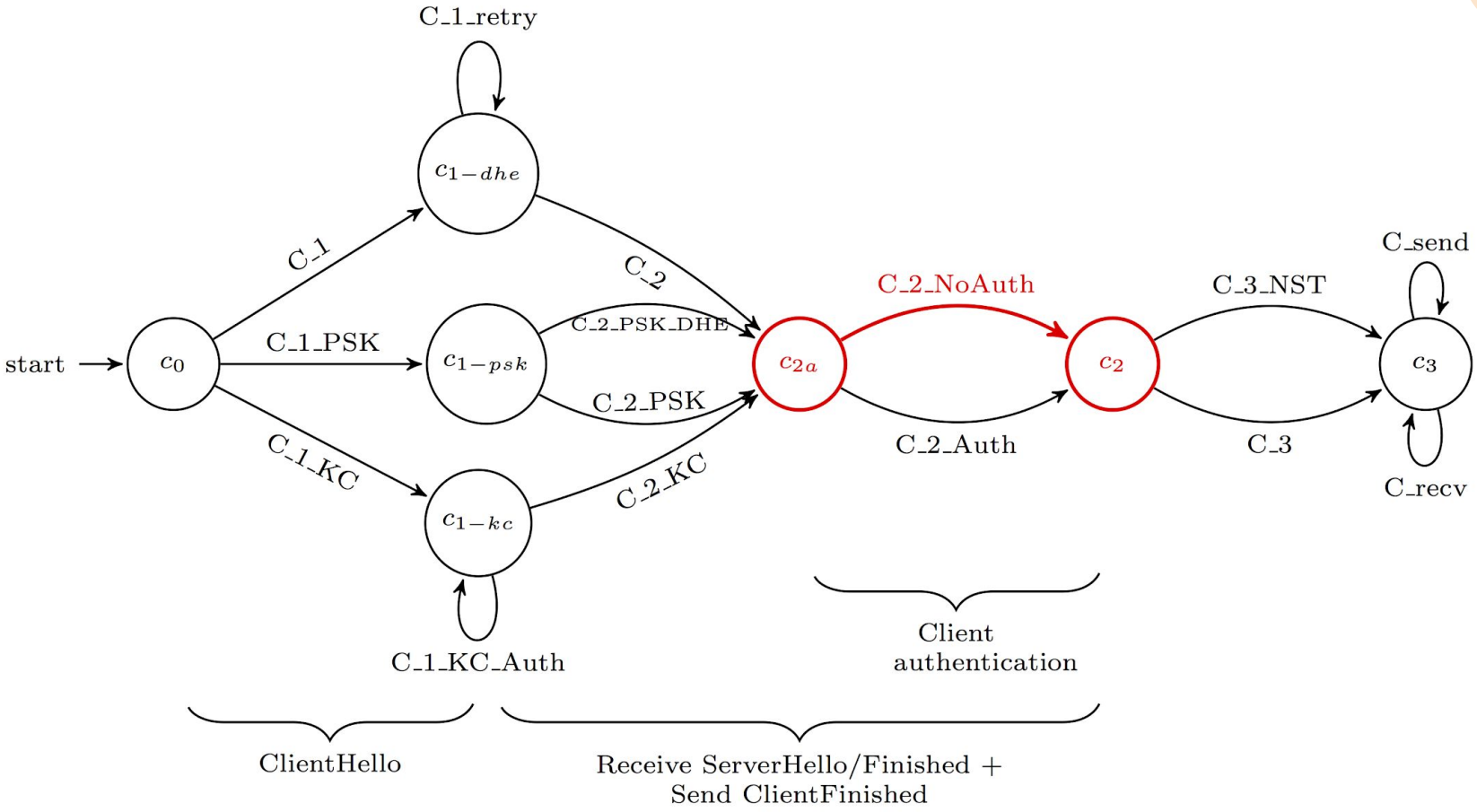
```
rule C_1:
let
    // Default C1 values
    tid = ~nc

    // Client Hello
    C   = $C
    nc  = ~nc
    pc  = $pc
    S   = $S

    // Client Key Share
    ga   = 'g'^~a

    messages = <C1_MSGS>

in
    [ Fr(nc)
    , Fr(~a)
    ]
  --[ C1(tid)
    , Start(tid, C, 'client')
    , Running(C, S, 'client', nc)
    , DH(C, ~a)
    ]->
    [ St_init(C,1, tid, C, nc, pc, S, ~a, messages, 'no_auth')
    , DHExp(C, ~a)
    , Out(<C,C1_MSGS>)
    ]
```

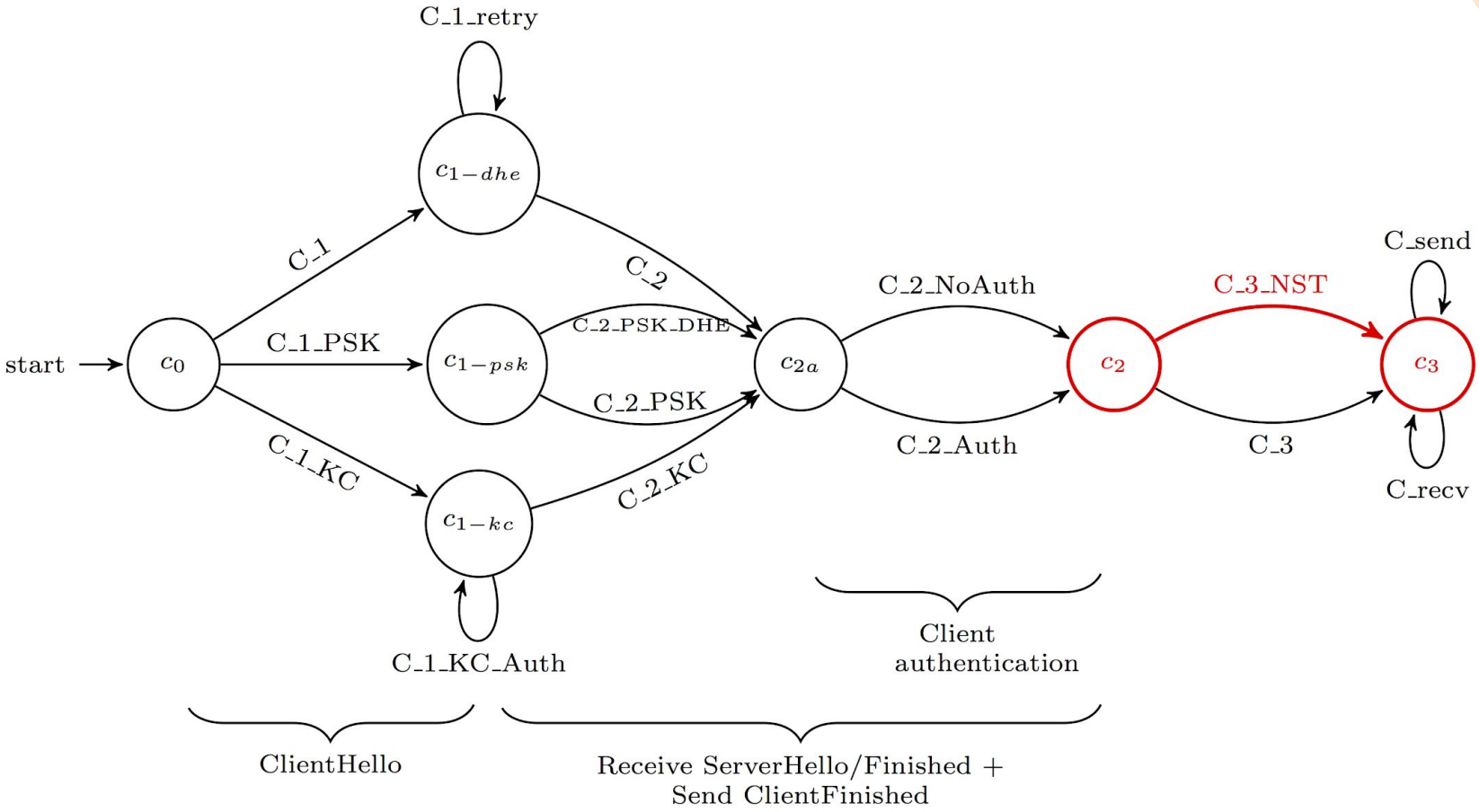

C_1_retry
c_{1-dhe}
C_1
C_2
C_2_NoAuth
C_3_NST
C_send
start
c_0
C_1_PSK
c_{1-psk}
C_2_PSK_DHE
C_2_PSK
c_{2a}
c_2
c_3
C_1_KC
c_{1-kc}
C_2_KC
C_2_Auth
C_3
C_recv
C_1_KC_Auth
Client
authentication
ClientHello
Receive ServerHello/Finished +
Send ClientFinished

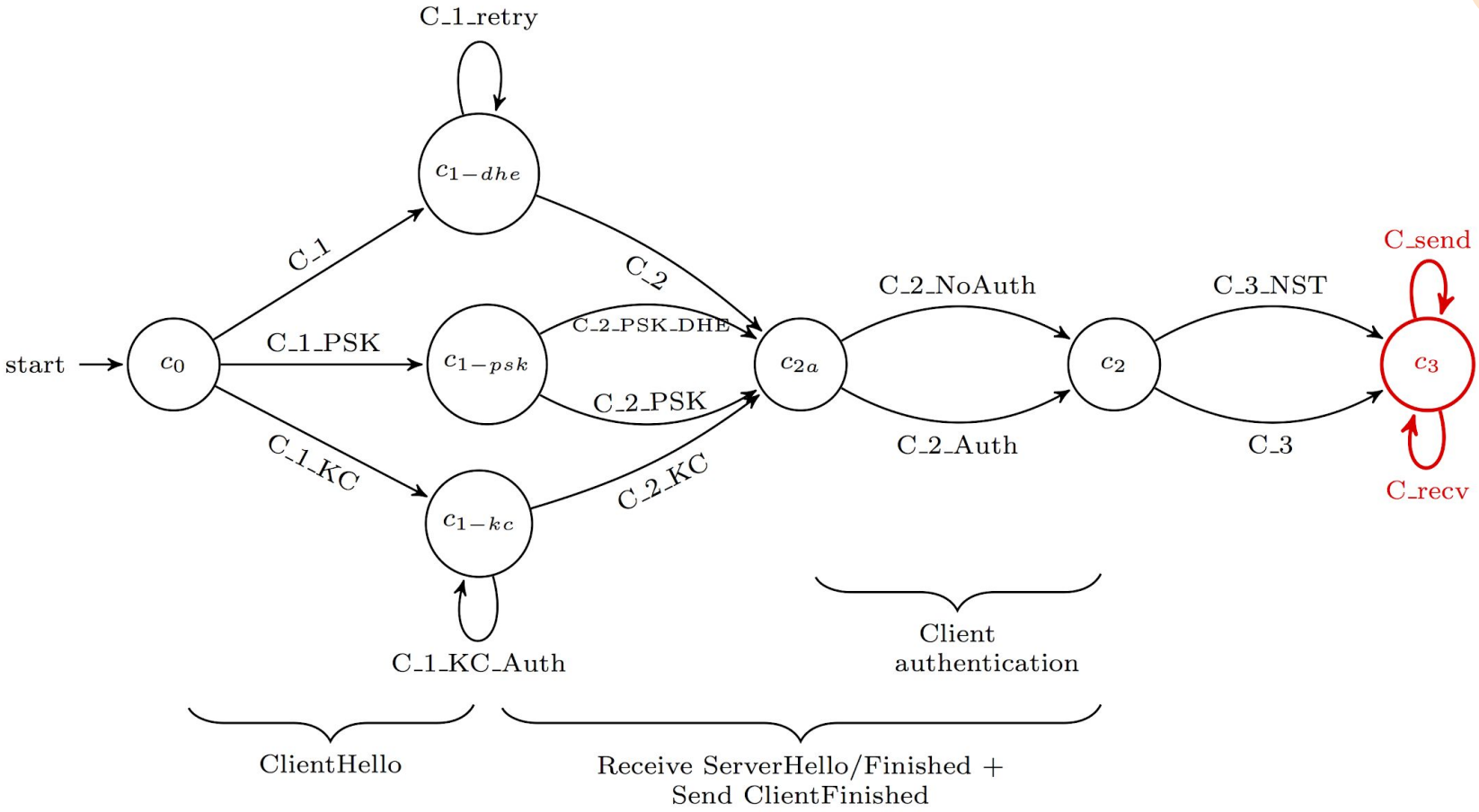

C_1_retry
c_{1-dhe}
C_1
C_2
start
c_0
C_1_PSK
c_{1-psk}
C_2_PSK_DHE
C_2_PSK
c_{2a}
C_2_NoAuth
C_2_Auth
c_2
C_3_NST
C_3
c_3
C_send
C_recv
C_1_KC
c_{1-kc}
C_2_KC
C_1_KC_Auth
Client
authentication
ClientHello
Receive ServerHello/Finished +
Send ClientFinished

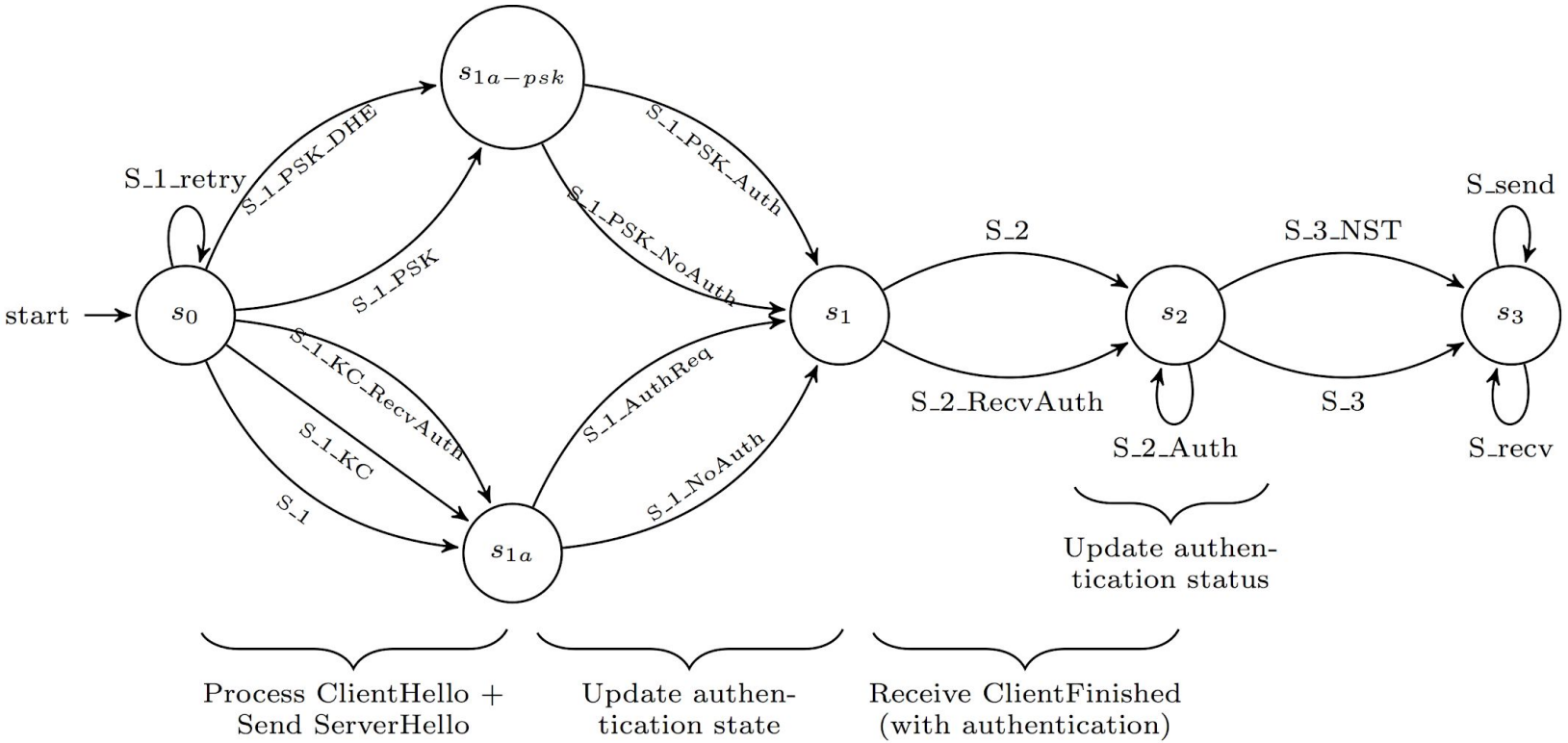

C_1_retry
c_{1-dhe}
C_1
C_2
C_2_NoAuth
C_3_NST
C_send
start
c_0
C_1_PSK
c_{1-psk}
C_2_PSK_DHE
C_2_PSK
c_{2a}
c_2
c_3
C_1_KC
c_{1-kc}
C_2_KC
C_2_Auth
C_3
C_recv
C_1_KC_Auth
Client
authentication
ClientHello
Receive ServerHello/Finished +
Send ClientFinished


C_1_retry
c_{1-dhe}
C_1
C_2
C_2_NoAuth
C_3_NST
C_send
start
c_0
C_1_PSK
c_{1-psk}
C_2_PSK_DHE
c_{2a}
c_2
c_3
C_2_PSK
C_2_Auth
C_3
C_recv
C_1_KC
C_2_KC
c_{1-kc}
C_1_KC_Auth
Client
authentication
ClientHello
Receive ServerHello/Finished +
Send ClientFinished


C_1_retry
c_{1-dhe}
C_1
C_2
C_2_NoAuth
C_3_NST
C_send
start
c_0
C_1_PSK
c_{1-psk}
C_2_PSK_DHE
C_2_PSK
c_{2a}
c_2
c_3
C_1_KC
c_{1-kc}
C_2_KC
C_2_Auth
C_3
C_recv
C_1_KC_Auth
Client
authentication
ClientHello
Receive ServerHello/Finished +
Send ClientFinished


C_1_retry
c_{1-dhe}
C_1
C_2
C_2_NoAuth
C_3_NST
C_send
start
c_0
C_1_PSK
c_{1-psk}
C_2_PSK_DHE
C_2_PSK
c_{2a}
c_2
c_3
C_1_KC
c_{1-kc}
C_2_KC
C_2_Auth
C_3
C_recv
C_1_KC_Auth
Client
authentication
ClientHello
Receive ServerHello/Finished +
Send ClientFinished


C_1_retry
c_{1-dhe}
C_1
C_2
C_2_NoAuth
C_3_NST
C_send
start
c_0
C_1_PSK
c_{1-psk}
C_2_PSK_DHE
C_2_PSK
c_{2a}
c_2
c_3
C_1_KC
c_{1-kc}
C_2_KC
C_2_Auth
C_3
C_recv
C_1_KC_Auth
Client
authentication
ClientHello
Receive ServerHello/Finished +
Send ClientFinished


start
s_0
s_{1a-psk}
s_{1a}
s_1
s_2
s_3
S_1_retry
S_1_PSK_DHE
S_1_PSK
S_1_KC_RecvAuth
S_1_KC
S_1
S_1_PSK_Auth
S_1_PSK_NoAuth
S_1_AuthReq
S_1_NoAuth
S_2
S_2_RecvAuth
S_2_Auth
S_3_NST
S_3
S_send
S_recv
Update authentication status
Process ClientHello + Send ServerHello
Update authentication state
Receive ClientFinished (with authentication)

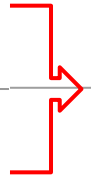
Adversarial Capabilities

- In addition to what Tamarin includes, we need to capture additional adversarial capabilities - for meaningful security notions

```
/*
  Reveal Ltk
  ----------------

  The adversarial capability to reveal long-term keys of parties.

  Premises:
    !Ltk($A, ~ltkA) - the long-term key to compromise

  Actions:
    RevLtk($A) - adversary has revealed the key of $A.

  Conclusions:
    Out(~ltkA) - provides the adversary with the long-term key
*/
rule Reveal_Ltk:
  [ !Ltk($A, ~ltkA) ] --[ RevLtk($A) ]-> [ Out(~ltkA) ]
```

Step 2: Encoding Properties

Security Property	Source
Unilateral authentication (server)	D.1.1
Mutual authentication	D.1.1
Confidentiality of ephemeral secret	D.1.1
Confidentiality of static secret	D.1.1
Perfect forward secrecy	D.1.1.1
Integrity of handshake messages	D.1.3



Confidentiality of session keys

```
 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
```

```
 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 variables on the first line (1),
- if the 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).

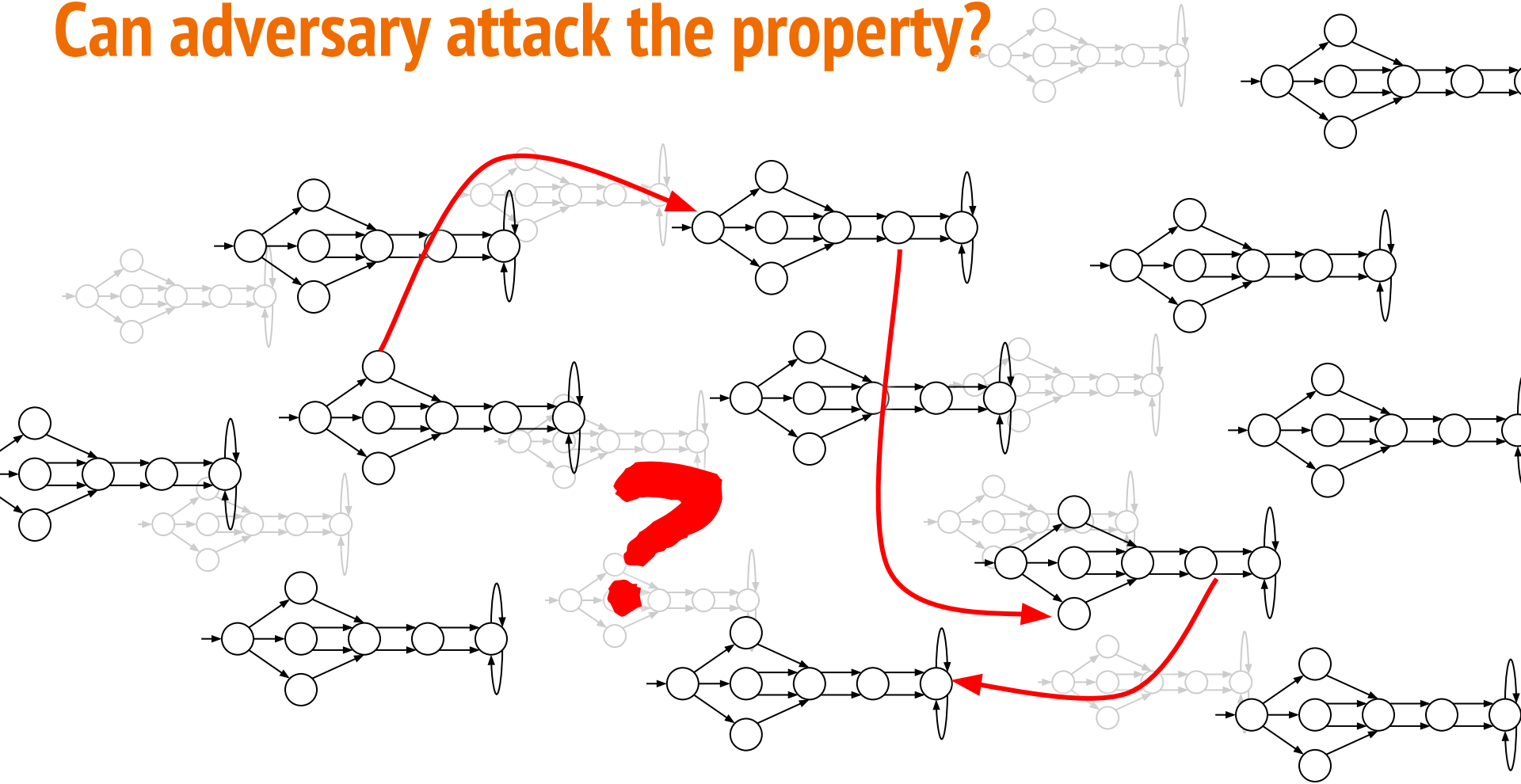
```
 secret_session_keys:
(1)   All actor peer role k #i.
(2)   SessionKey(actor, peer, role, <k, authenticated>)@i
(3)   & not ((Ex #r. RevLtk(peer)@r & #r < #i)
           | (Ex #r. RevLtk(actor@r & #r < #i))
(4)   ==> not Ex #j. K(k)@j
```

Aim to show that this holds in possible combinations of client, server and adversary behaviours!

Constructed Tamarin encodings for all of the main properties:

Security Property	
Unilateral authentication (server)	`entity_authentication` `mutual_entity_authentication`
Mutual authentication	
Confidentiality of ephemeral secret	`secret_early_data_keys` `secret_session_keys(with PFS)`
Confidentiality of static secret	
Perfect forward secrecy	
Integrity of handshake messages	`transcript_agreement` `mutual_transcript_agreement`

Can adversary attack the property?



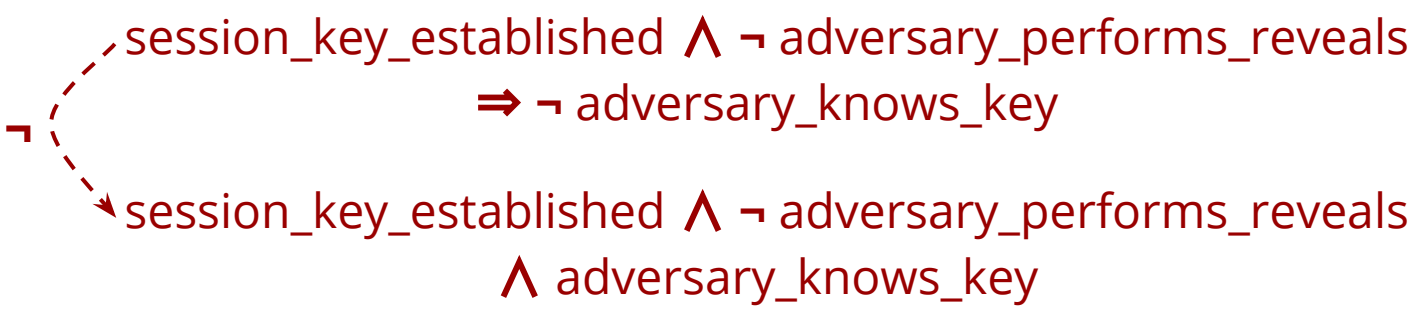
Step 3: Producing Proofs

- Let's simplify our `secret_session_keys` encoding:

$$\text{session\_key\_established} \land \neg\, \text{adversary\_performs\_reveals} \Rightarrow \neg\, \text{adversary\_knows\_key}$$

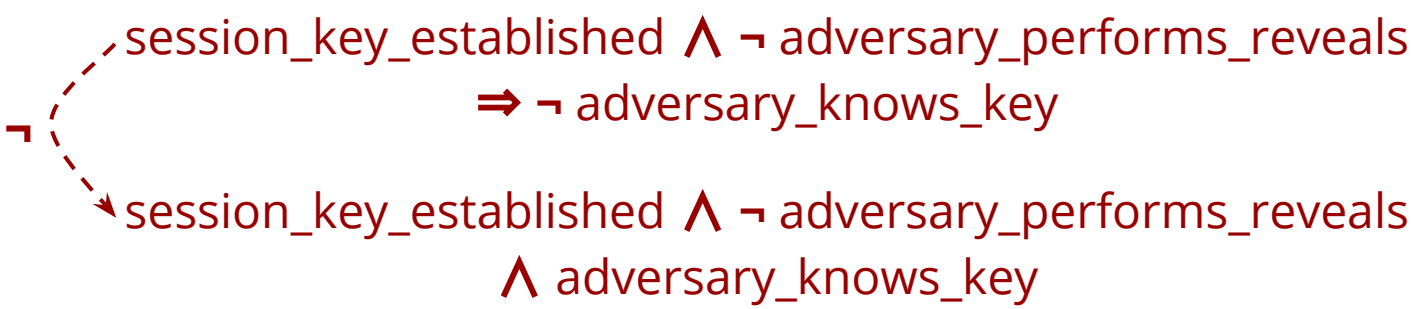
Step 3: Producing Proofs

- Let's simplify our `secret_session_keys` encoding:



Step 3: Producing Proofs

- Let's simplify our `secret_session_keys` encoding:



- Tamarin looks for a protocol execution that contains session_key_established and adversary_knows_key but that does not use adversary_performs_reveals

Step 3: Producing Proofs

- Let's simplify our `secret_session_keys` encoding:

¬ session_key_established ∧ ¬ adversary_performs_reveals
⇒ ¬ adversary_knows_key

session_key_established ∧ ¬ adversary_performs_reveals
∧ adversary_knows_key

- Tamarin looks for a protocol execution that contains session_key_established and adversary_knows_key but that <u>does not use</u> adversary_performs_reveals

{ } = property holds! {counterexample} = attack!

Step 3: Producing Proofs

- Tamarin translates the encoding into a constraint system - refines knowledge until it can determine that the encoding holds in all cases, or that a counterexample exists
- Tamarin uses a set of heuristics to determine what to do next
- ‘Autoprove’ or ‘Interactive’

Proof scripts

```
lemma secret_session_keys:
  all-traces
  "∀ actor peer role k #i.
        ((SessionKey( actor, peer, role, <k,
'authenticated'>
          ) @ #i) ∧
         (¬((∃ #r. (RevLtk( peer ) @ #r) ∧ (#r < #i))
∨
            (∃ #r. (RevLtk( actor ) @ #r) ∧ (#r <
#i))))) ⇒
        (¬(∃ #j. !KU( k ) @ #j))"
simplify
solve( SessionKey( actor, peer, role,
                   <k, 'authenticated'>
       ) @ #i )
  case C_2_Auth_case_1
  solve( F_St_C_2a_init( ~nc, $C, ~nc, $pc, $S, ~ns,
$ps, ss, es,
                         prev_messages, config_hash,
'no_auth'
        ) ▶₀ #i )
    case C_2_KC
    by contradiction /* from formulas */
  next
    case C_2_case_1
    by contradiction /* from formulas */
  next
    case C_2_case_2
    by contradiction /* from formulas */
  qed
next
  case C_2_Auth_case_2
  solve( F_St_C_2a_init( ~nc, $C, ~nc, $pc, $S, ~ns,
$ps, ss, es,
                         prev_messages, config_hash,
'no_auth'
        ) ▶₀ #i )
    case C_2_KC
```

Visualization display

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

1. simplify

2. induction

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

Constraint system

last: none

formulas:

∃ actor peer k #i.
(EarlyDataKey(actor, peer, 'client', k) @ #i)
∧
(∀ #r. (RevLtk(peer) @ #r) ⇒ ⊥) ∧ (∃ #j. (!KU(k) @ #j))

equations:

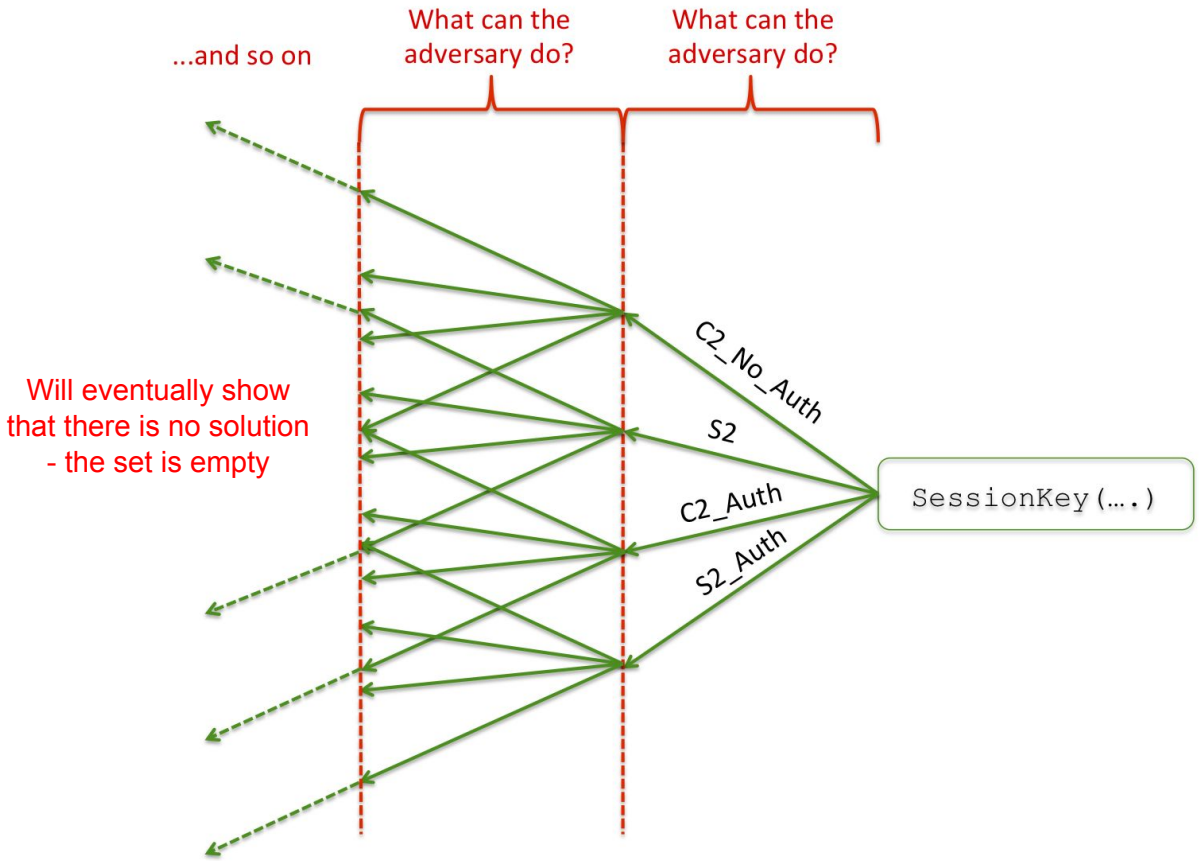
subst:

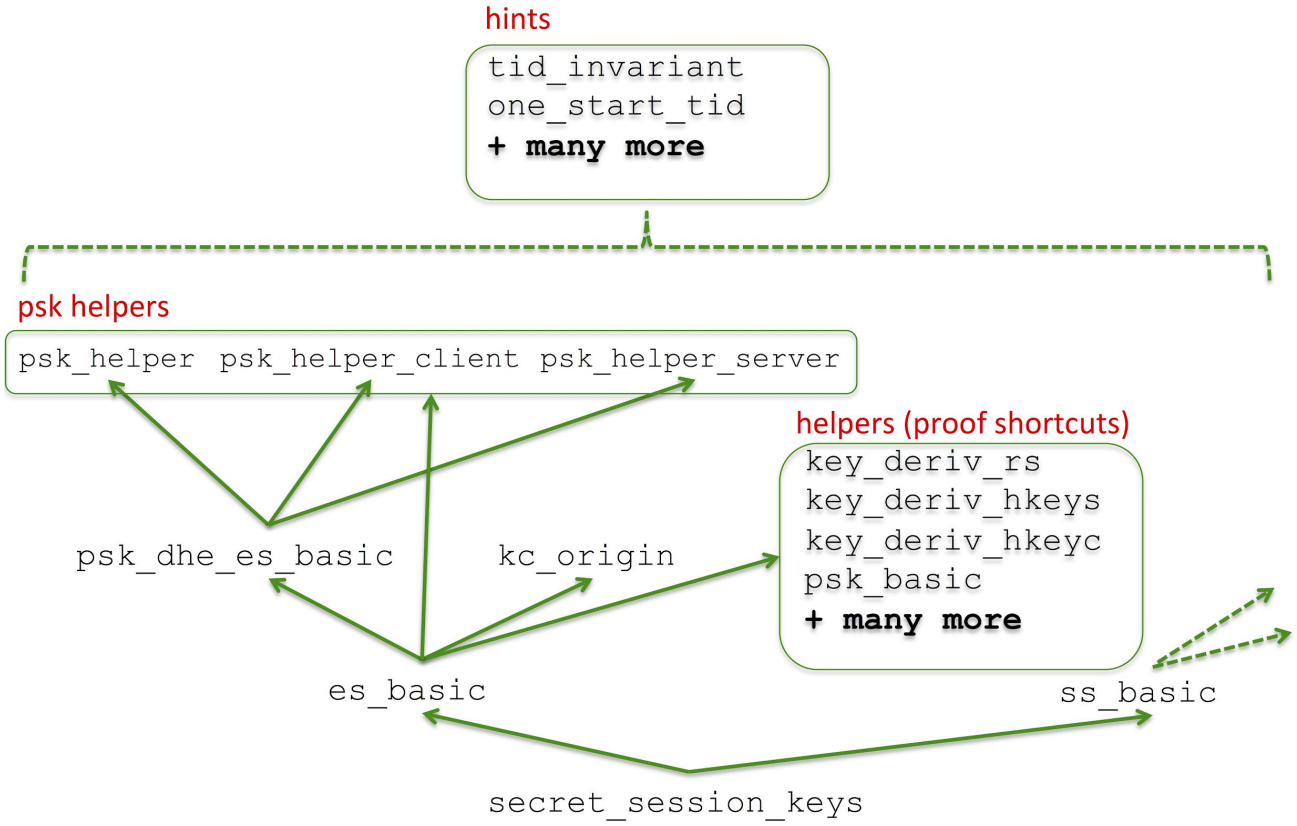
conj:

lemmas:

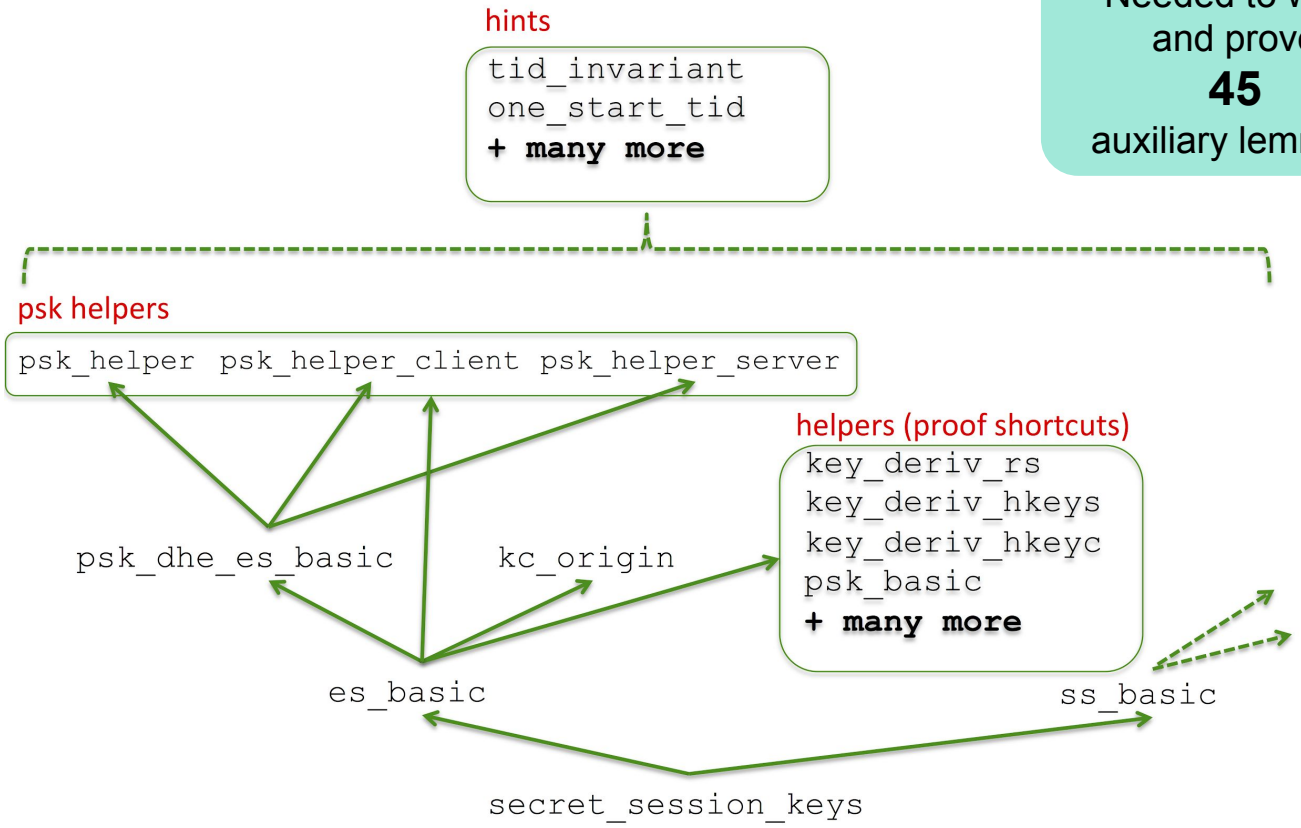
∀ A x y #i #j.
(GenLtk(A, x) @ #i) ∧ (GenLtk(A, y) @ #j) ⇒ #i = #j

∀ actor actor2 psk_id psk_id2 peer peer2 rs auth_status
auth_status2 #i #j.
(UsePSK(actor, psk_id, peer, rs, 'client', auth_status
) @ #i) ∧
(UsePSK(actor2, psk_id2, peer2, rs, 'server', auth_status2
) @ #j)


...and so on
What can the adversary do?
What can the adversary do?
Will eventually show that there is no solution - the set is empty
C2_No_Auth
S2
C2_Auth
S2_Auth
SessionKey(….)


hints
tid_invariant
one_start_tid
+ many more
psk helpers
psk_helper psk_helper_client psk_helper_server
helpers (proof shortcuts)
key_deriv_rs
key_deriv_hkeys
key_deriv_hkeyc
psk_basic
+ many more
psk_dhe_es_basic
kc_origin
es_basic
ss_basic
secret_session_keys

Needed to write and prove **45** auxiliary lemmas!



```
lemma secret_session_keys:
  all-traces
  "∀ actor peer role k #i.
         ((SessionKey( actor, peer, role, <k,
'authenticated'>
           ) @ #i) ∧
          (¬((∃ #r. (RevLtk( peer ) @ #r) ∧ (#r < #i))
∨
             (∃ #r. (RevLtk( actor ) @ #r) ∧ (#r <
#i))))) ⇒
         (¬(∃ #j. !KU( k ) @ #j))"
```

```
lemma secret_session_keys:
  all-traces
  "∀
```

```
lemma entity_authentication [reuse]:
  all-traces
  "∀ actor peer nonces #i.
        ((CommitNonces( actor, peer, 'client', nonces
) @ #i) ∧
          (¬(∃ #r. RevLtk( peer ) @ #r))) ⇒
        (∃ #j peer2.
          (RunningNonces( peer, peer2, 'server',
nonces ) @ #j) ∧
          (#j < #i))"
```

+ mutual

```
'auth
∨
#i)))
```

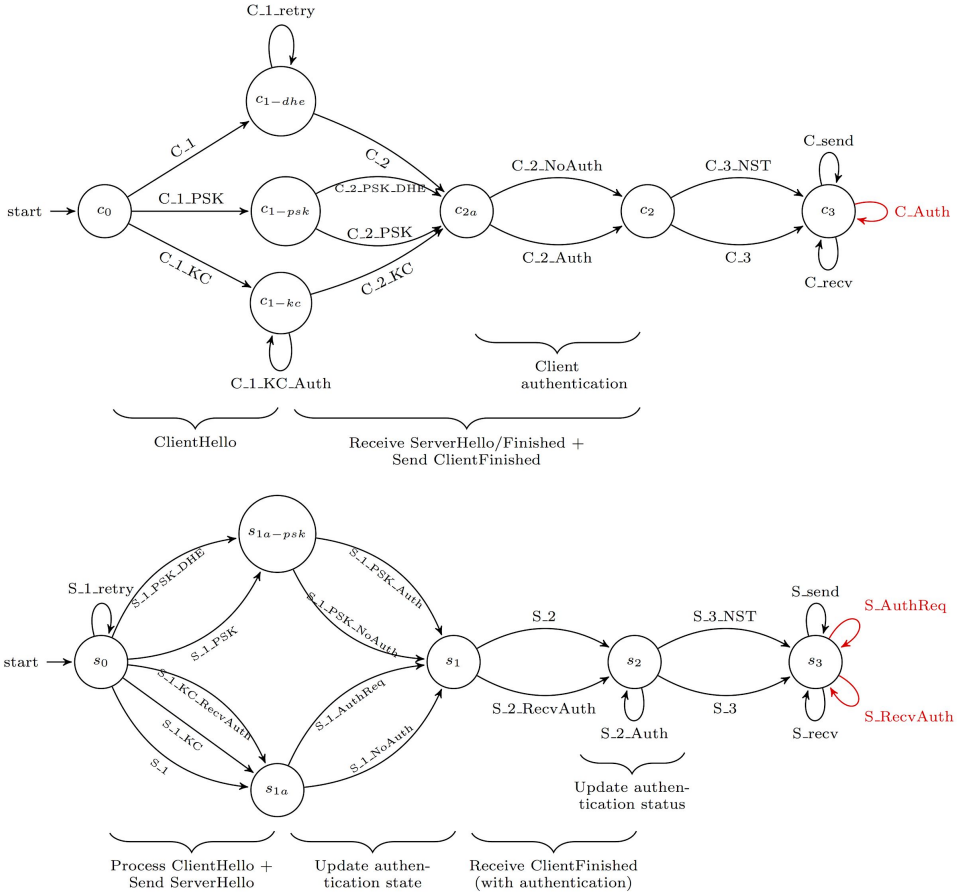
```
lemma secret_session_keys:
  all-traces
  "∀
        lemma entity_authentication [reuse]:
          all-traces
'auth     "∀
        lemma transcript_agreement [reuse]:
          all-traces
      ) @ #
          "∀ actor peer transcript #i.
∨
                ((CommitTranscript( actor, peer, 'client',
              transcript
#i)))
                  ) @ #i) ∧
      nonce
                 (¬(∃ #r. RevLtk( peer ) @ #r))) ⇒
                (∃ #j peer2.
                  (RunningTranscript( peer, peer2, 'server',
              transcript
                   ) @ #j) ∧
                  (#j < #i))"
```

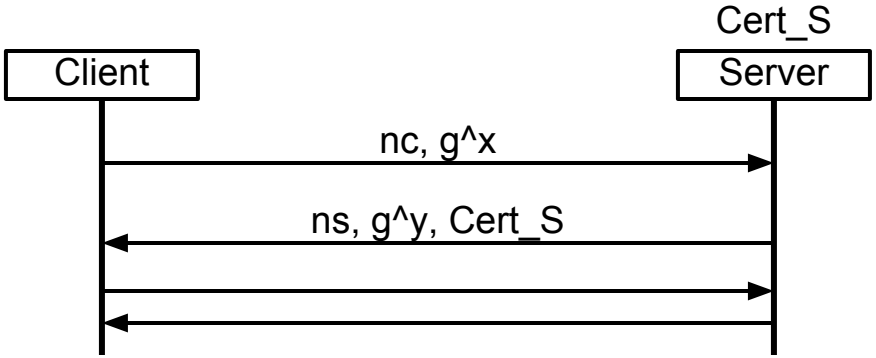
+ mutual

```
lemma secret_session_keys:
  all-traces
  "∀ lemma entity_authentication [reuse]:
      all-traces
'auth  "∀ lemma transcript_agreement [reuse]:
          all-traces
     ) @ #  "∀ actor peer transcript #i.
∨                  ((CommitTranscript( actor, peer, 'client',
          transcript
#i)))                    ) @ #i) ∧
     nonce
                  lemma secret_early_data_keys:
                    all-traces
                    "∀ actor peer k #i.
          transc          ((EarlyDataKey( actor, peer, 'client', k ) @
                  #i) ∧
                           (¬(∃ #r. RevLtk( peer ) @ #r))) ⇒
                          (¬(∃ #j. !KU( k ) @ #j))"
```

```
lemma secret_session_keys:
  all-traces
  "∀ lemma entity_authentication [reuse]:
      all-traces
'auth  "∀ lemma transcript_agreement [reuse]:
         all-traces
     ) @ #  "∀ actor peer transcript #i.
∨                  ((CommitTranscript( actor, peer, 'client',
           transcript
#i)))                    ) @ #i) ∧
     nonce
                lemma secret_early_data_keys:
                  all-traces
                  "∀ actor peer k #i.
           transc        ((EarlyDataKey( actor, peer, 'client', k ) @
                #i) ∧
                         (¬(∃ #r. RevLtk( peer ) @ #r))) ⇒
                        (¬(∃ #j. !KU( k ) @ #j))"
```

Finding An Attack

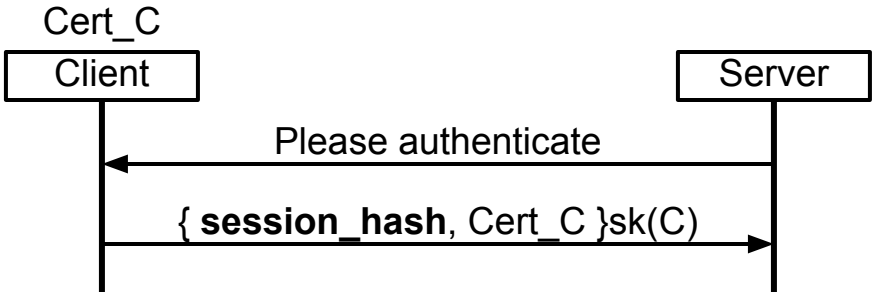




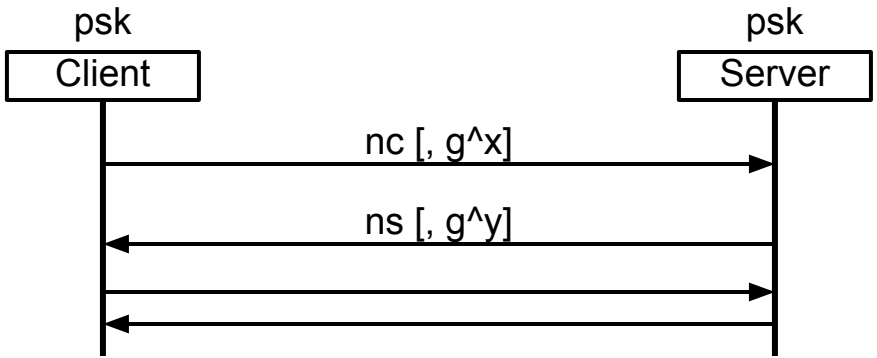
ECDH Handshake

(unilateral, only mentioning relevant items)

Compute ***session_hash*** that includes ***ns***, ***nc***, ***Cert_S***

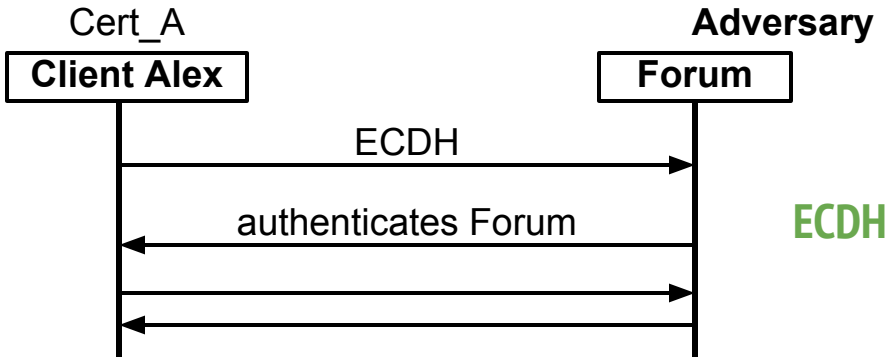


Post-handshake Client authentication



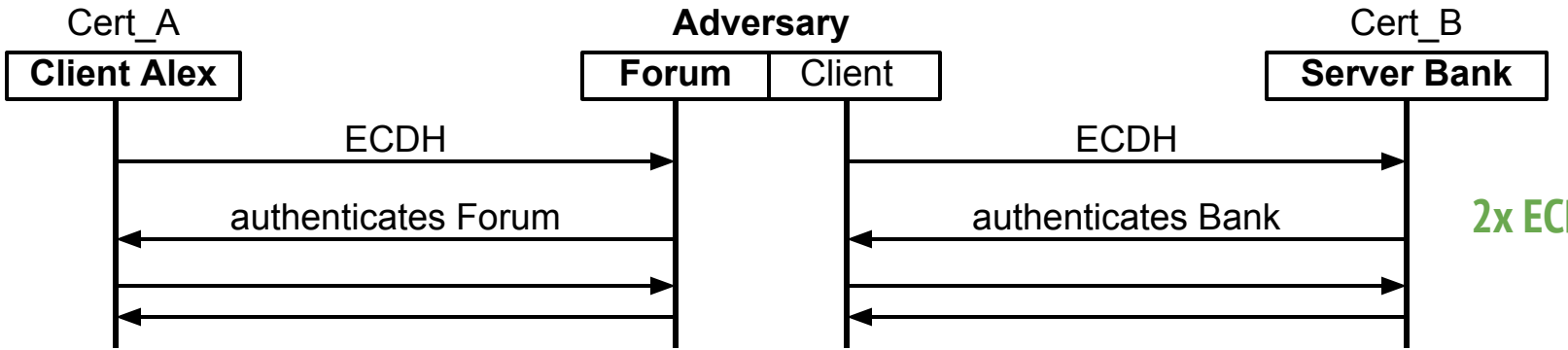
PSK [-DHE]

Compute ***session_hash*** that includes ***ns***, ***nc***

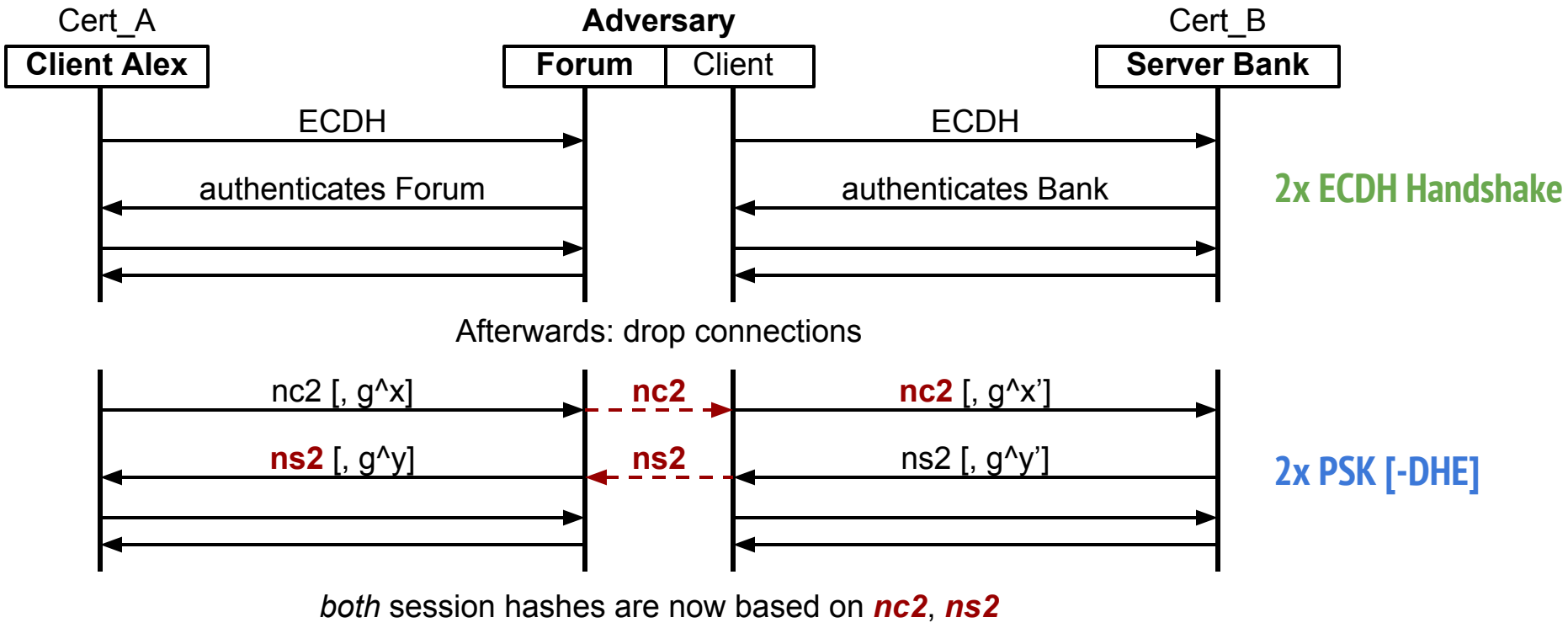


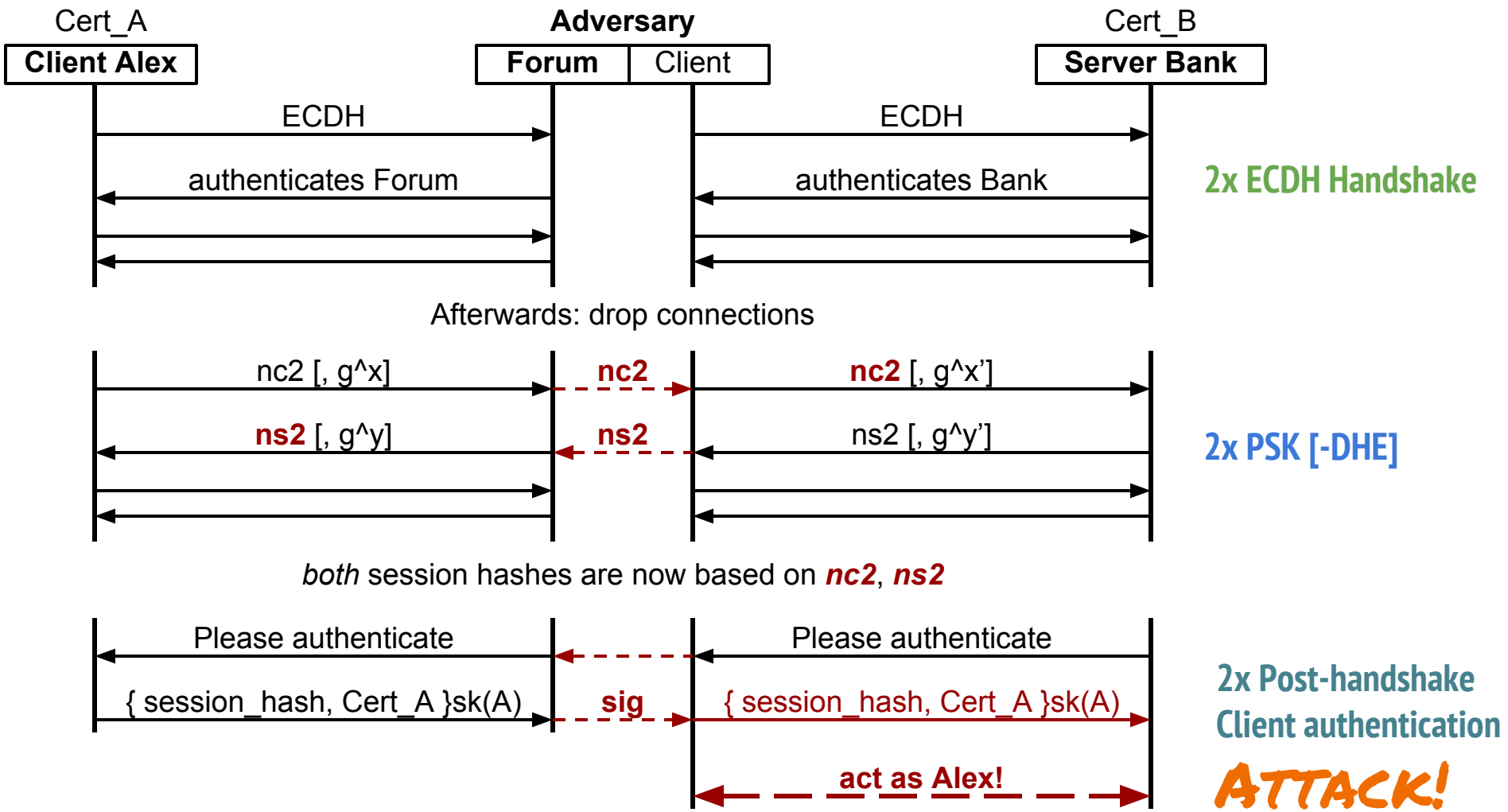
ECDH Handshake

ATTACK SETUP!



2x ECDH Handshake


Cert_A
Client Alex
Adversary
Forum
Client
Cert_B
Server Bank
ECDH
authenticates Forum
ECDH
authenticates Bank
2x ECDH Handshake
Afterwards: drop connections
nc2 [, g^x]
nc2
nc2 [, g^x']
ns2 [, g^y]
ns2
ns2 [, g^y']
2x PSK [-DHE]
both session hashes are now based on nc2, ns2


Cert_A
Client Alex
Adversary
Forum
Client
Cert_B
Server Bank
ECDH
authenticates Forum
ECDH
authenticates Bank
2x ECDH Handshake
Afterwards: drop connections
nc2 [, g^x]
nc2
nc2 [, g^x']
ns2 [, g^y]
ns2
ns2 [, g^y']
2x PSK [-DHE]
both session hashes are now based on nc2, ns2
Please authenticate
Please authenticate
{ session_hash, Cert_A }sk(A)
sig
{ session_hash, Cert_A }sk(A)
act as Alex!
2x Post-handshake
Client authentication
ATTACK!

Step 1: Building the Model

- TLS 1.3 was a rapidly moving target
- Draft 21 - a completely new protocol!
- We now modelled in a far more granular fashion
  - higher transparency - good for us, also good for everyone else!

TLS 1.3 Protocol Overview

---snip---

TLS supports three basic key exchange modes:

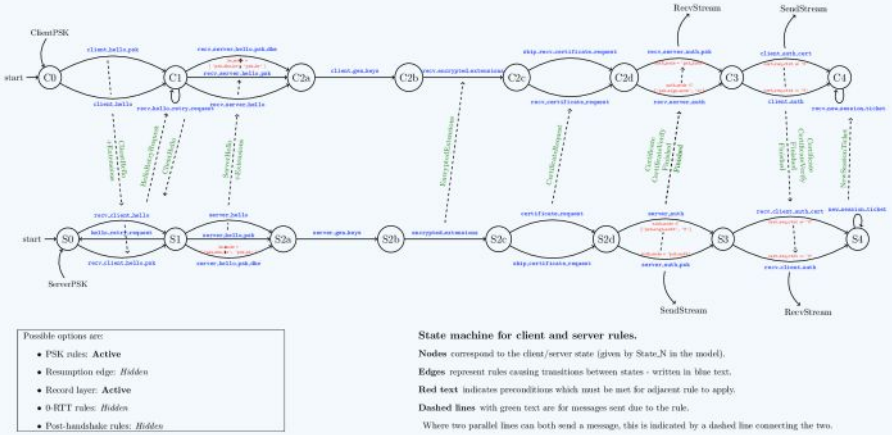
- (EC)DHE (Diffie-Hellman both the finite field and elliptic curve varieties),
- PSK-only, and
- PSK with (EC)DHE

below shows the basic full TLS handshake:

```
       Client                                           Server

Key  ^ ClientHello
Exch | + key_share*
     | + signature_algorithms*
     | + psk_key_exchange_modes*
     v + pre_shared_key*         -------->
                                                  ServerHello  ^ Key
                                                + key_share*   | Exch
                                           + pre_shared_key*   v
                                       {EncryptedExtensions}   ^  Server
                                       {CertificateRequest*}   v  Params
                                              {Certificate*}   ^
                                       {CertificateVerify*}    | Auth
                                                 {Finished}    v
                               <--------     [Application Data*]
     ^ {Certificate*}
Auth | {CertificateVerify*}
     v {Finished}              -------->
```

Tamarin model

We model the different phases, options and message flights through a series of rule invocations. The basic full handshake is captured by this state machine diagram:



Possible options are:

- PSK rules: **Active**
- Resumption edge: *Hidden*
- Record layer: **Active**
- 0-RTT rules: *Hidden*
- Post-handshake rules: *Hidden*

State machine for client and server rules.

Nodes correspond to the client/server state (given by State_N in the model).

Edges represent rules causing transitions between states - written in blue text.

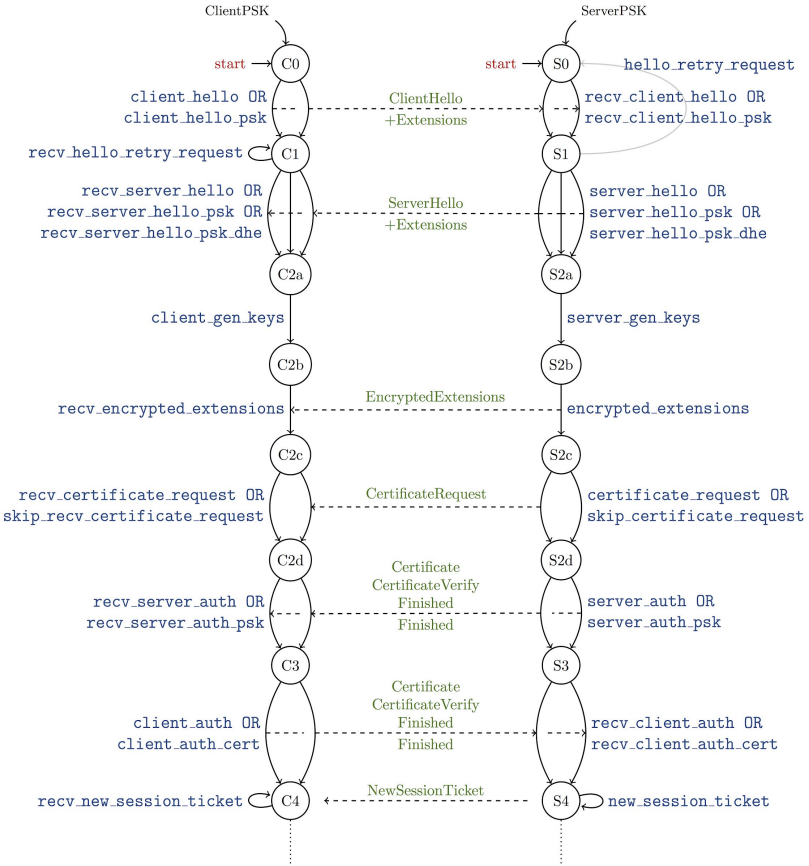
Red text indicates preconditions which must be met for adjacent rule to apply.

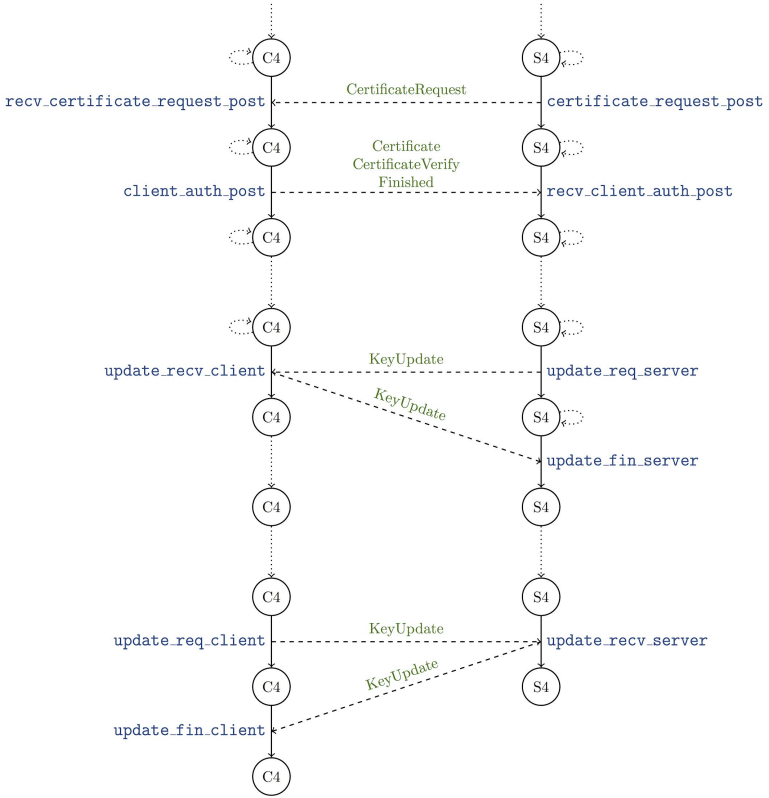
Dashed lines with green text are for messages sent due to the rule.

Where two parallel lines can both send a message, this is indicated by a dashed line connecting the two.

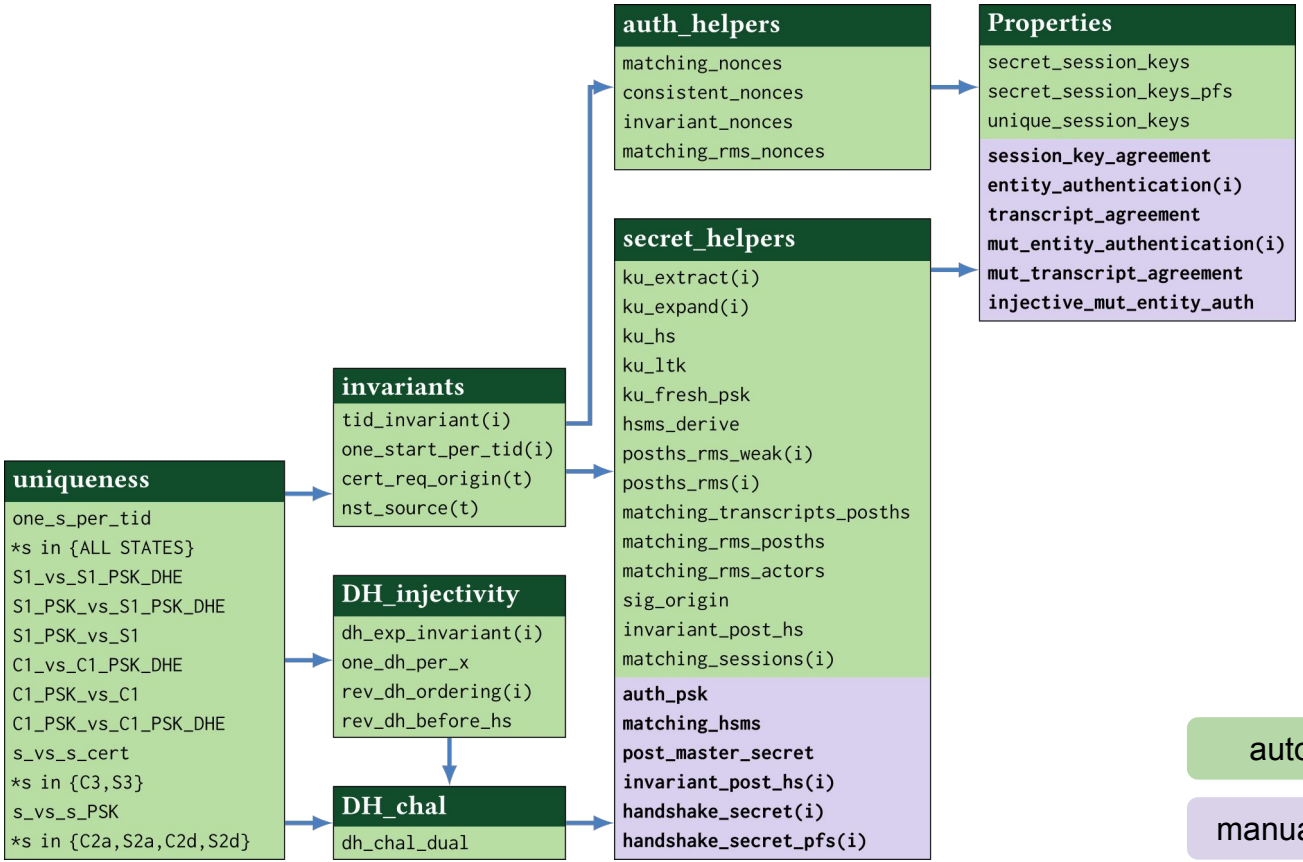
For example, we see that a PSK-only handshake is captured through the invocation of the rules `client_hello_psk -> recv_client_hello_psk -> server_hello_psk -> ...` for the PSK-DHE handshake, the rule `server_hello_psk_dhe` would be used instead.

We associate with each handshake *message* (i.e. not necessarily each flight) a distinct rule, to help separate concerns.


ClientPSK
ServerPSK
start
C0
S0
hello_retry_request
client_hello OR
client_hello_psk
ClientHello
+Extensions
recv_client_hello OR
recv_client_hello_psk
recv_hello_retry_request
C1
S1
recv_server_hello OR
recv_server_hello_psk OR
recv_server_hello_psk_dhe
ServerHello
+Extensions
server_hello OR
server_hello_psk OR
server_hello_psk_dhe
C2a
S2a
client_gen_keys
server_gen_keys
C2b
S2b
recv_encrypted_extensions
EncryptedExtensions
encrypted_extensions
C2c
S2c
recv_certificate_request OR
skip_recv_certificate_request
CertificateRequest
certificate_request OR
skip_certificate_request
C2d
S2d
recv_server_auth OR
recv_server_auth_psk
Certificate
CertificateVerify
Finished
Finished
server_auth OR
server_auth_psk
C3
S3
client_auth OR
client_auth_cert
Certificate
CertificateVerify
Finished
Finished
recv_client_auth OR
recv_client_auth_cert
recv_new_session_ticket
C4
NewSessionTicket
S4
new_session_ticket


C4
S4
CertificateRequest
recv_certificate_request_post
certificate_request_post
Certificate
CertificateVerify
Finished
client_auth_post
recv_client_auth_post
KeyUpdate
update_recv_client
update_req_server
KeyUpdate
update_fin_server
KeyUpdate
update_req_client
update_recv_server
KeyUpdate
update_fin_client

Step 2: Encoding Security Properties



Step 3: Producing Proofs

Security Property	
Establishing the same session keys	✓
Secret session keys	✓
Peer authentication	✓
Uniqueness of session keys	✓
Downgrade protection (within 1.3)	✓
Perfect forward secrecy	✓
Key Compromise Impersonation (KCI) resistance	✓

Security Property	
Establishing the same session keys	✓
Secret session keys	✓
Peer authentication	✓
Uniqueness of session keys	✓
Downgrade protection (within 1.3)	✓
Perfect forward secrecy	✓
Key Compromise Impersonation (KCI) resistance	✓

More fine-grained model → more computational power required

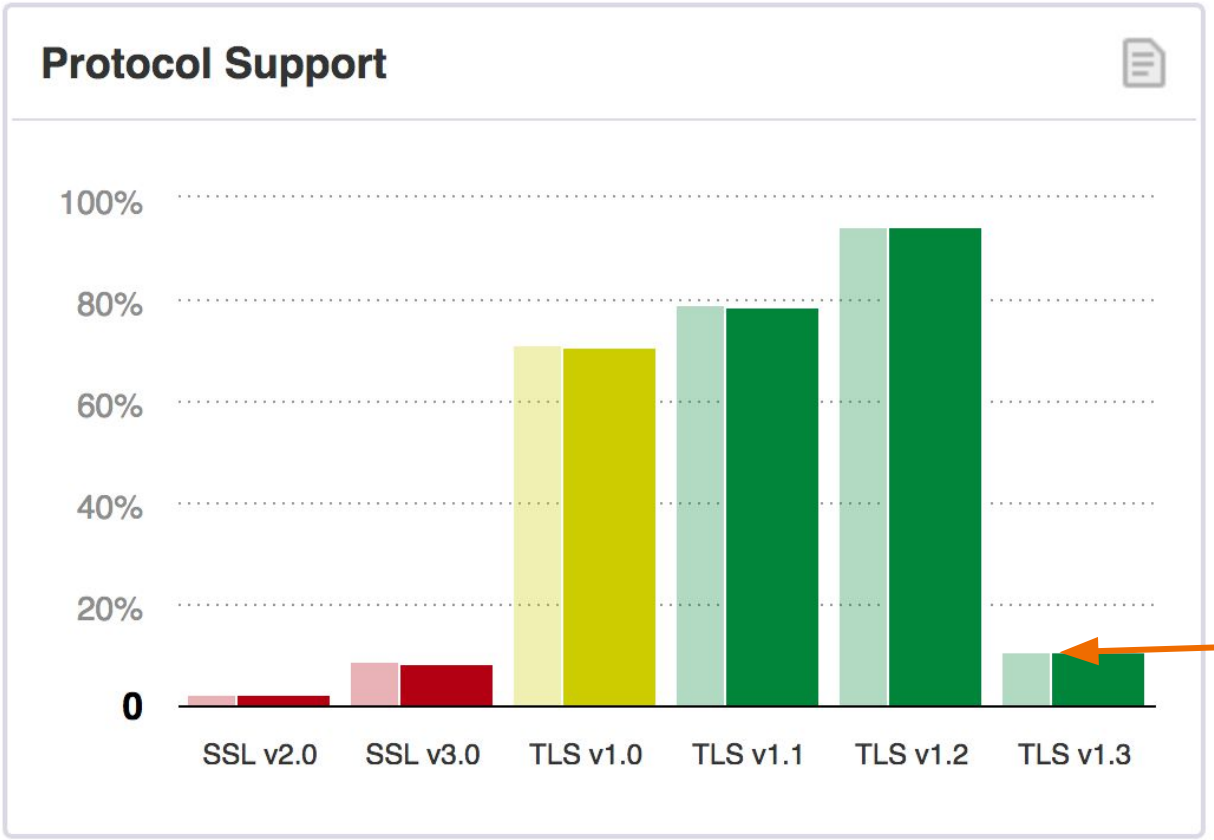
- 48-core machine, 512GB of RAM
- 10GB RAM to load, can consume 100GB RAM for a proof
- 1 week to prove entire model
- 3 person-months of modelling

Security Property	
Establishing the same session keys	✓
Secret session keys	✓
Peer authentication	✓ See [CHHMS17]
Uniqueness of session keys	✓
Downgrade protection (within 1.3)	✓
Perfect forward secrecy	✓
Key Compromise Impersonation (KCI) resistance	✓

More fine-grained model → more computational power required

- 48-core machine, 512GB of RAM
- 10GB RAM to load, can consume 100GB RAM for a proof
- 1 week to prove entire model
- 3 person-months of modelling

TLS 1.3 is Out There!



8% of connections

50% of traffic

January 2019 - **10.7%** of sites support TLS 1.3!

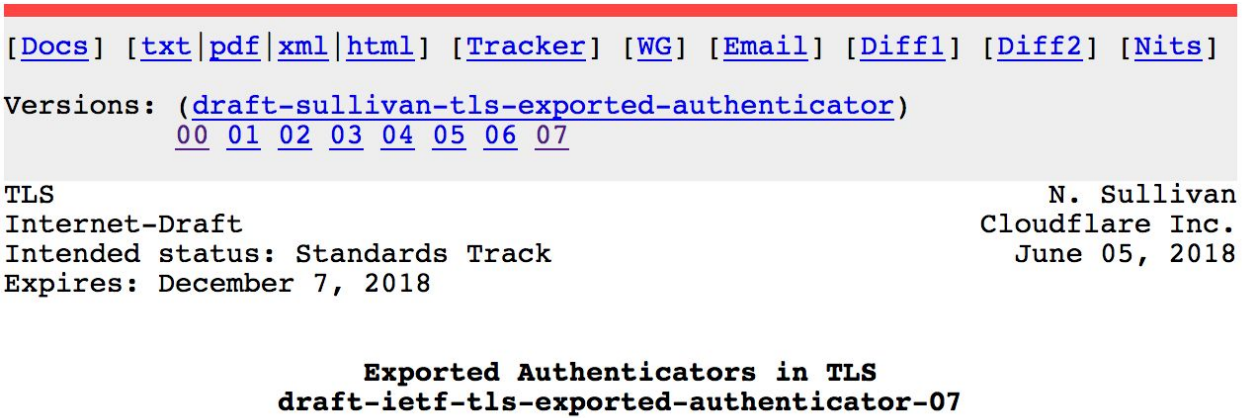
https://www.ssllabs.com/ssl-pulse/

What Lies Ahead?

- Feedback loop - modelling complex protocols is making Tamarin better
  - Improved precision (granularity) of modelling
  - Improve automation

What Lies Ahead?

- Feedback loop - modelling complex protocols is making tools better
  - Improved precision (granularity) of modelling
  - Improve automation
- TLS 1.3 extensions



[Docs] [txt|pdf|xml|html] [Tracker] [WG] [Email] [Diff1] [Diff2] [Nits]

Versions: (draft-sullivan-tls-exported-authenticator)
00 01 02 03 04 05 06 07

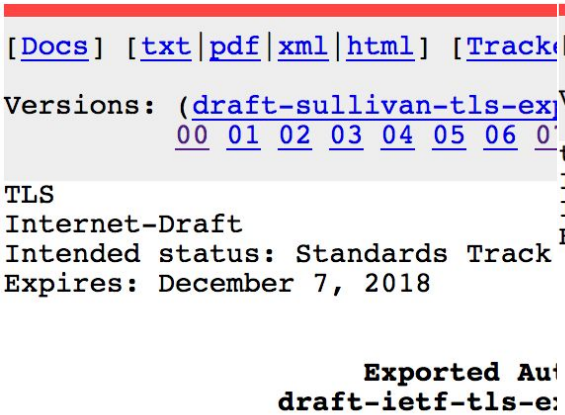
TLS
Internet-Draft
Intended status: Standards Track
Expires: December 7, 2018

N. Sullivan
Cloudflare Inc.
June 05, 2018

Exported Authenticators in TLS
draft-ietf-tls-exported-authenticator-07

What Lies Ahead?

- Feedback loop - modelling complex protocols is making tools better
  - Improved precision (granularity) of modelling
  - Improve automation
- TLS 1.3 extensions

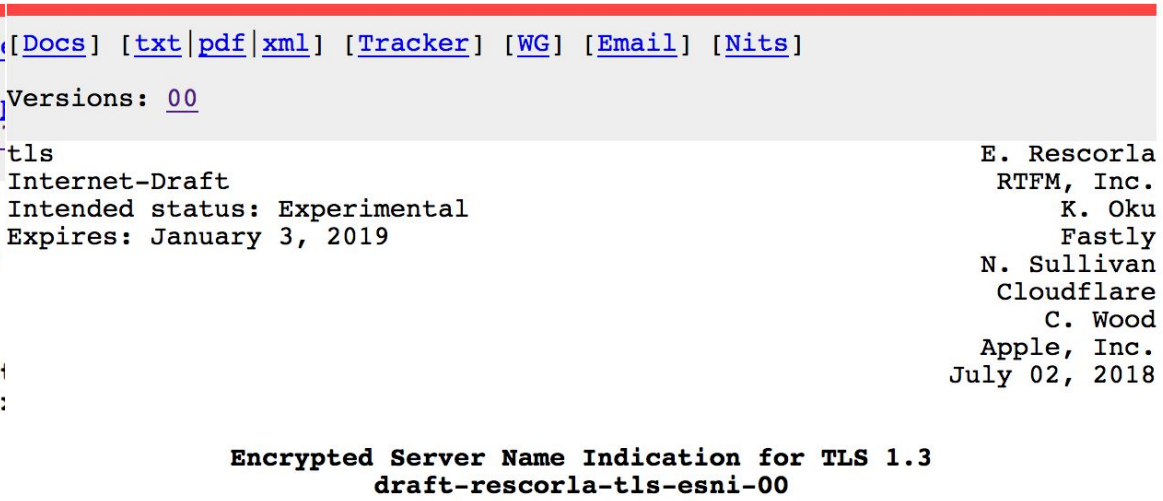


[Docs] [txt|pdf|xml|html] [Track

Versions: (draft-sullivan-tls-ex
00 01 02 03 04 05 06 0

TLS
Internet-Draft
Intended status: Standards Track
Expires: December 7, 2018

Exported Au
draft-ietf-tls-e



[Docs] [txt|pdf|xml] [Tracker] [WG] [Email] [Nits]

Versions: 00

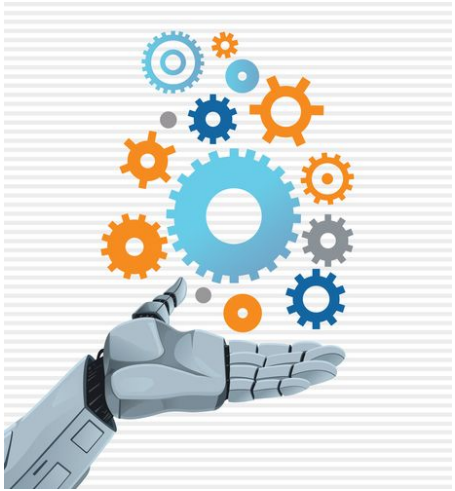
tls
Internet-Draft
Intended status: Experimental
Expires: January 3, 2019

E. Rescorla
RTFM, Inc.
K. Oku
Fastly
N. Sullivan
Cloudflare
C. Wood
Apple, Inc.
July 02, 2018

Encrypted Server Name Indication for TLS 1.3
draft-rescorla-tls-esni-00

Takeaways

- Logical core of TLS 1.3 seems sound!
- We have built a transparent model others can build on (Github)
- Many complementary approaches to analysing TLS
- Newer process allows for preemptive decision making and hopefully produces a stronger protocol, requiring less patching



Resources

- TLS 1.3 analysis github page: https://tls13tamarin.github.io/TLS13Tamarin/
- Papers:
  - [CHSM16] Automated Analysis and Verification of TLS 1.3: 0-RTT, Resumption and Delayed Authentication, https://ieeexplore.ieee.org/document/7546518/
  - [CHHSM17] A Comprehensive Symbolic Analysis of TLS 1.3, https://dl.acm.org/citation.cfm?id=3134063
- Symbolic analysis tools:
  - [Tamarin] Tamarin Prover, http://tamarin-prover.github.io/
  - [ProVerif] ProVerif, http://prosecco.gforge.inria.fr/personal/bblanche/proverif/

Bonus Slide

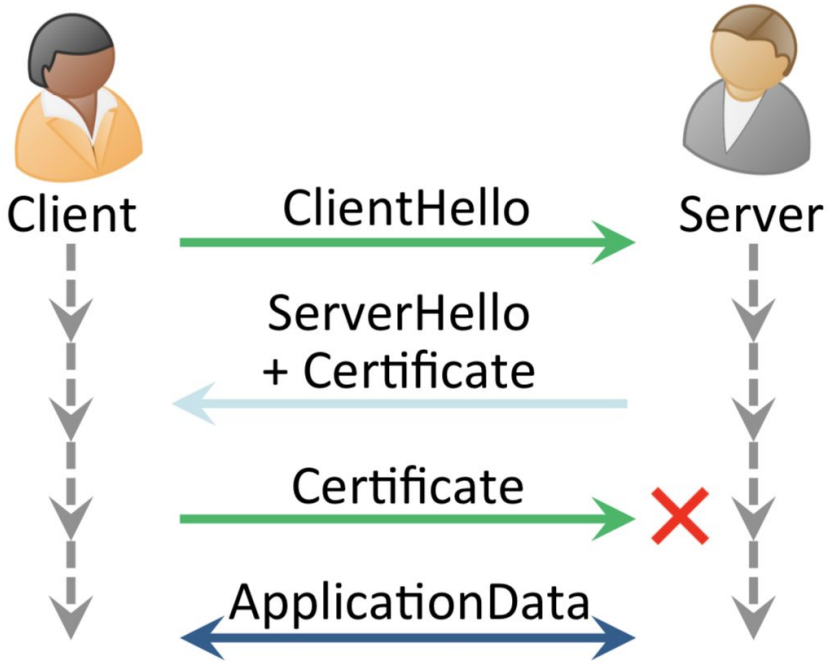


Figure: The awkward handshake.

See [CHHMS17] for details.