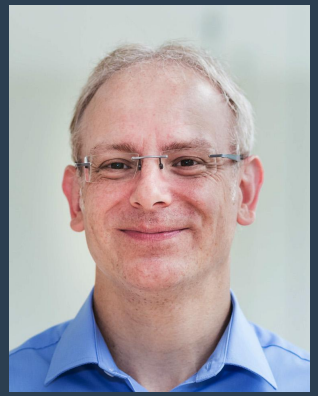
Automated Analysis of TLS 1.3

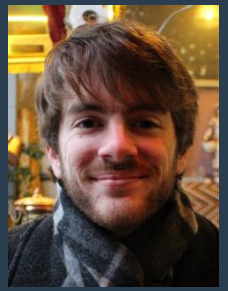
Cas Cremers



Marko Horvat

Jonathan Hoyland

Sam Scott



Thyla van der Merwe

Crypto Welcomes TLS 1.3
19 August 2018

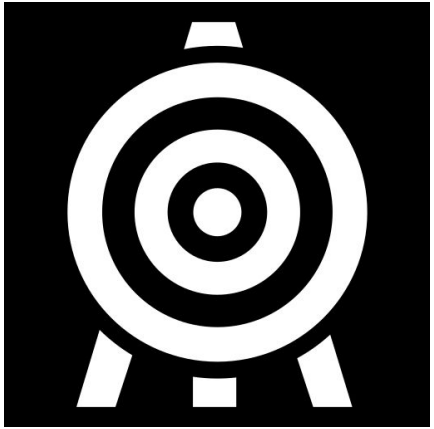
CISPA HELMHOLTZ-ZENTRUM i. G.

moz://a

Objectives

Analysis of the “logical core” of TLS 1.3 design

- Cover *all* modes and their interaction
- Detailed threat models
- Accurate authentication properties



Provide (relatively quick) feedback and guarantees

Methodology

Perform **symbolic analysis**
using the **Tamarin prover** [Tamarin]

Emperor Tamarin

Monkey species from South America

Tamarin is a good fit for TLS 1.3:

- Natural modeling of complex state machines
- Support for stateful protocols with loops
- Most accurate DH support in field

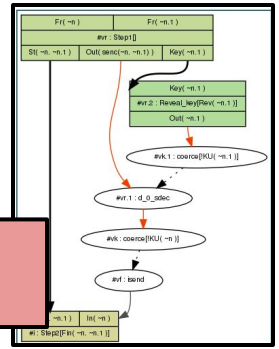
Tamarin prover

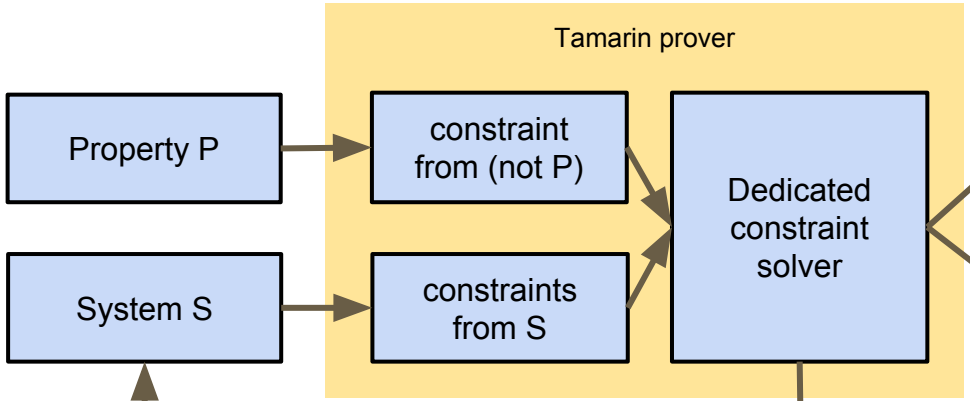
Theorem Prover

Constraint solver

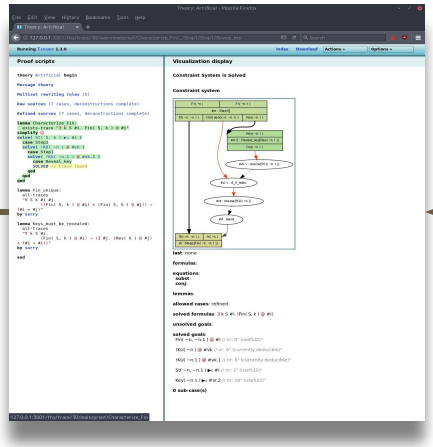
In one slide: Tamarin is a custom constraint solver impersonating as an (interactive) theorem prover

Tamarin prover





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



Run out of time or memory

Interactive mode
Inspect partial proof

(Simplified view - interface also allows direct interaction with solver)

Specifying protocols

Rewrite rules that specify transition system

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

(Very similar to Oracles that encode protocol behaviour)

Specifying protocols

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)

Specifying protocols

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)

Specifying protocols

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

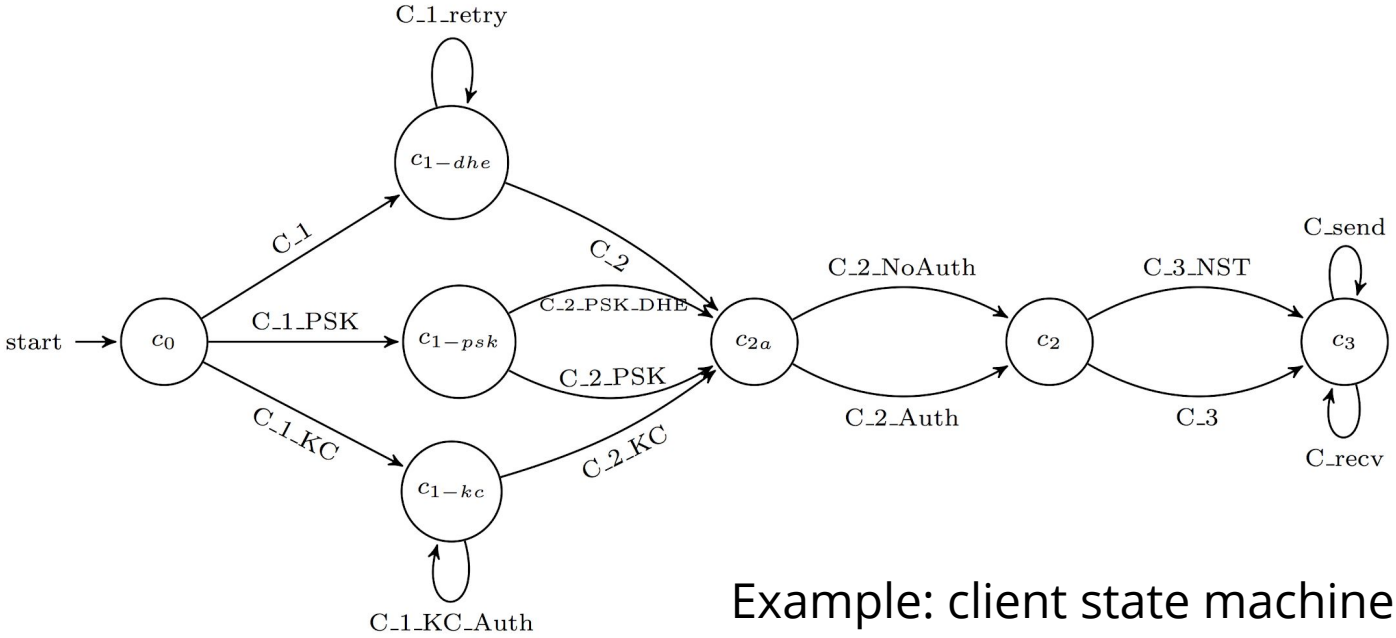
```
rule my_protocol_step2:
   [ In( m1 ),  State( ThreadID, `state1`, previousData ) ]
      --[ Accepted( ThreadID, k) ]->
   [ Out( m2 ), State( ThreadID, `state2`, newData ) ]
```

premises (LHS)

actions

conclusions (RHS)

Rules model state machine



Example: client state machine

Rules correspond to edges

Specifying adversary capabilities

- Also similar to Oracles

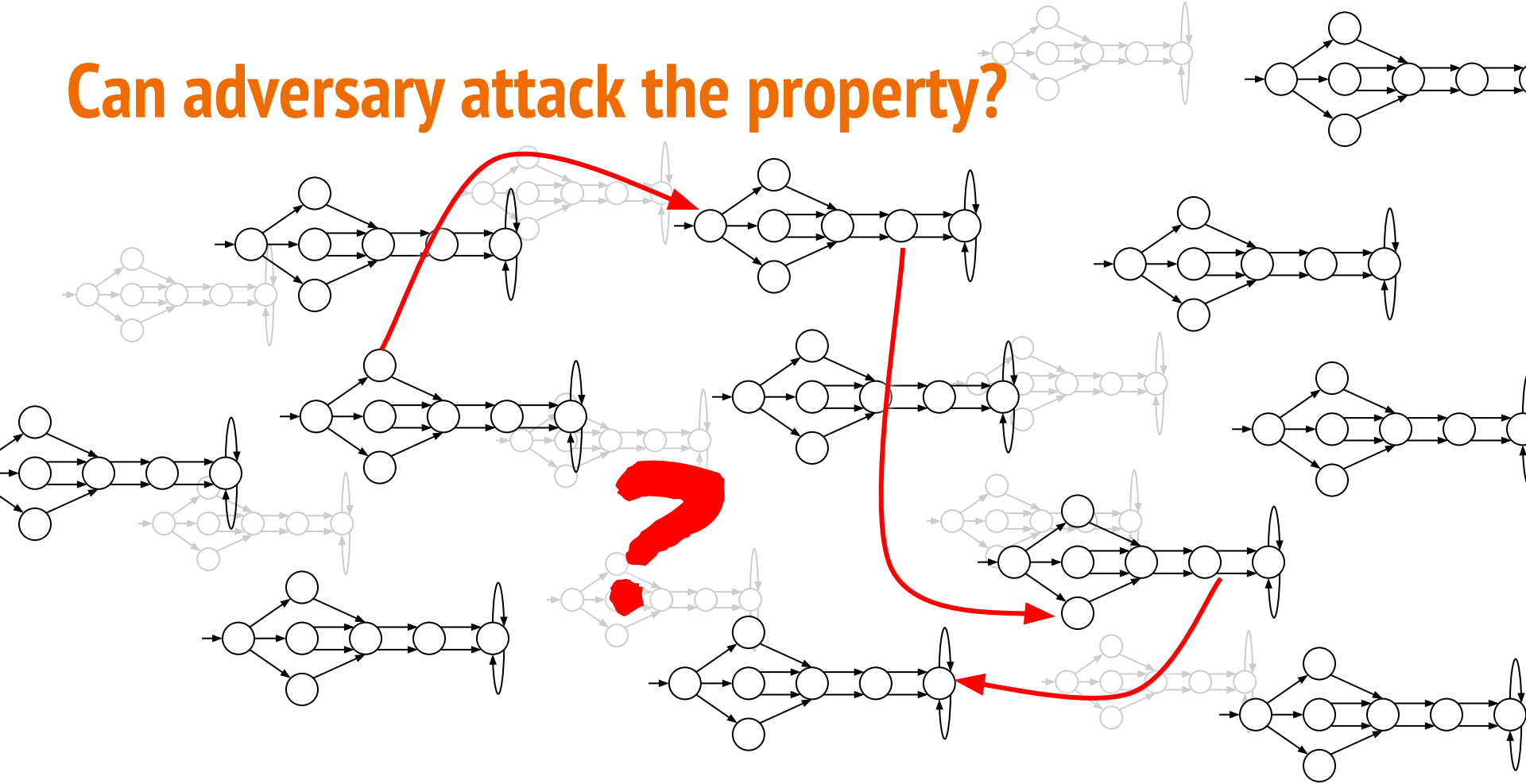
```
rule SessionKeyReveal:
   [ State( ThreadID, … , Key ) ]
      --[ SessionKeyReveal( ThreadID, Key ) ]->
   [ Out( Key ) ]
```

Specifying properties

- Guarded fragment of first order logic with timepoints

```
lemma my_secret_key:

   “Forall tid key #i.

      Accepted( tid, key )@i =>

         ( not Ex #j. K(key)@j ) ”
```

Can adversary attack the property?



The reality strikes back

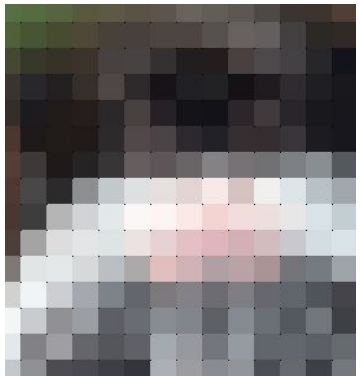
Reality

Computational

Symbolic

In theory, computational models are more accurate than symbolic models, but they also abstract away from many real-world aspects.

The reality strikes back *harder*



Reality **Computational** **Symbolic**

In practice however, computational analyses only consider very small parts of real-world systems to make analysis feasible. Symbolic methods may be able to cover larger parts. Hence incomparable guarantees.

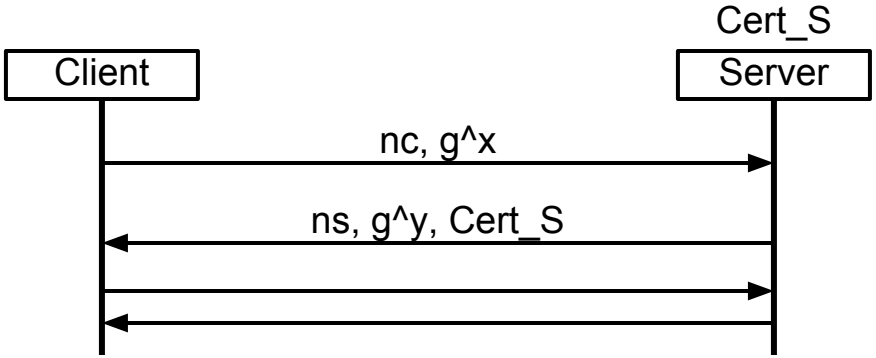
Results!

We analysed Draft 10, Draft 10+, Draft 21

Proofs for all main properties on **Draft 10** [CHSM16] and **Draft 21** [CHHSM17] in the symbolic model

During our analysis, around Draft 10:

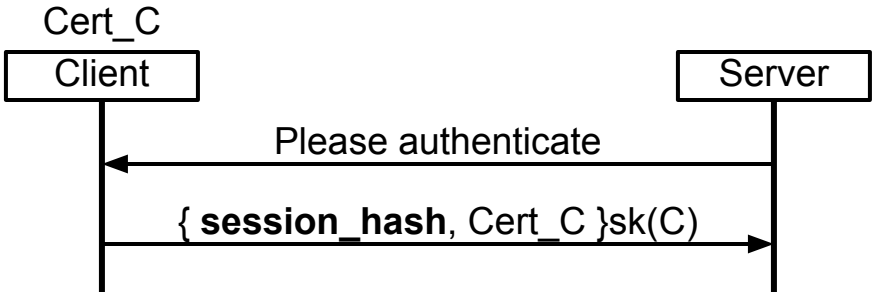
> "let's introduce *post-handshake client authentication*"
>
> **Tamarin finds an attack** on Draft 10+! [CHHMS16]
>
> - 18 messages
> - 3 modes



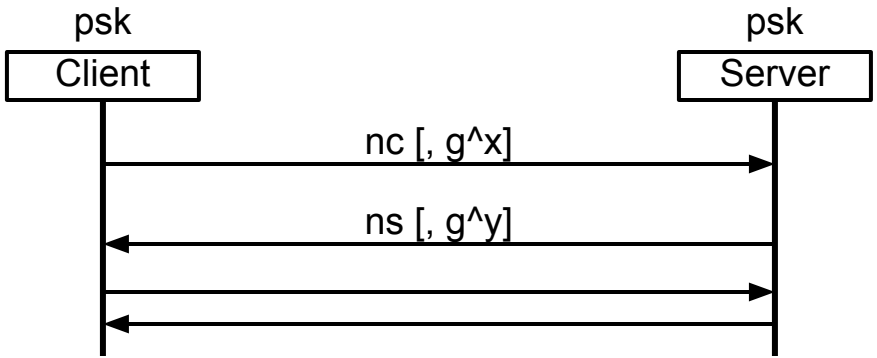
ECDH Handshake

(unilateral, only mentioning relevant items)

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

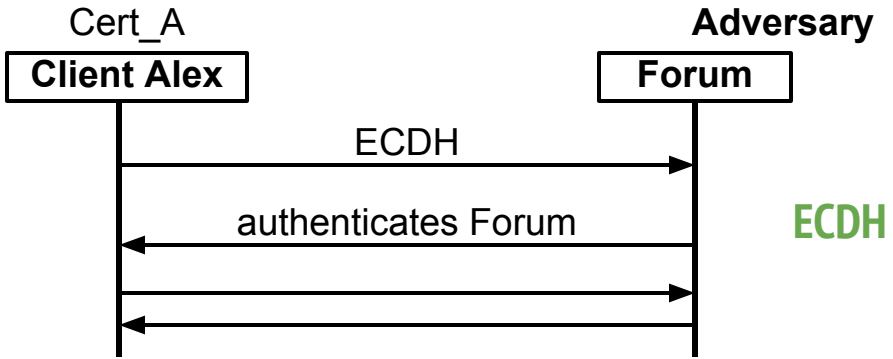


Post-handshake Client authentication



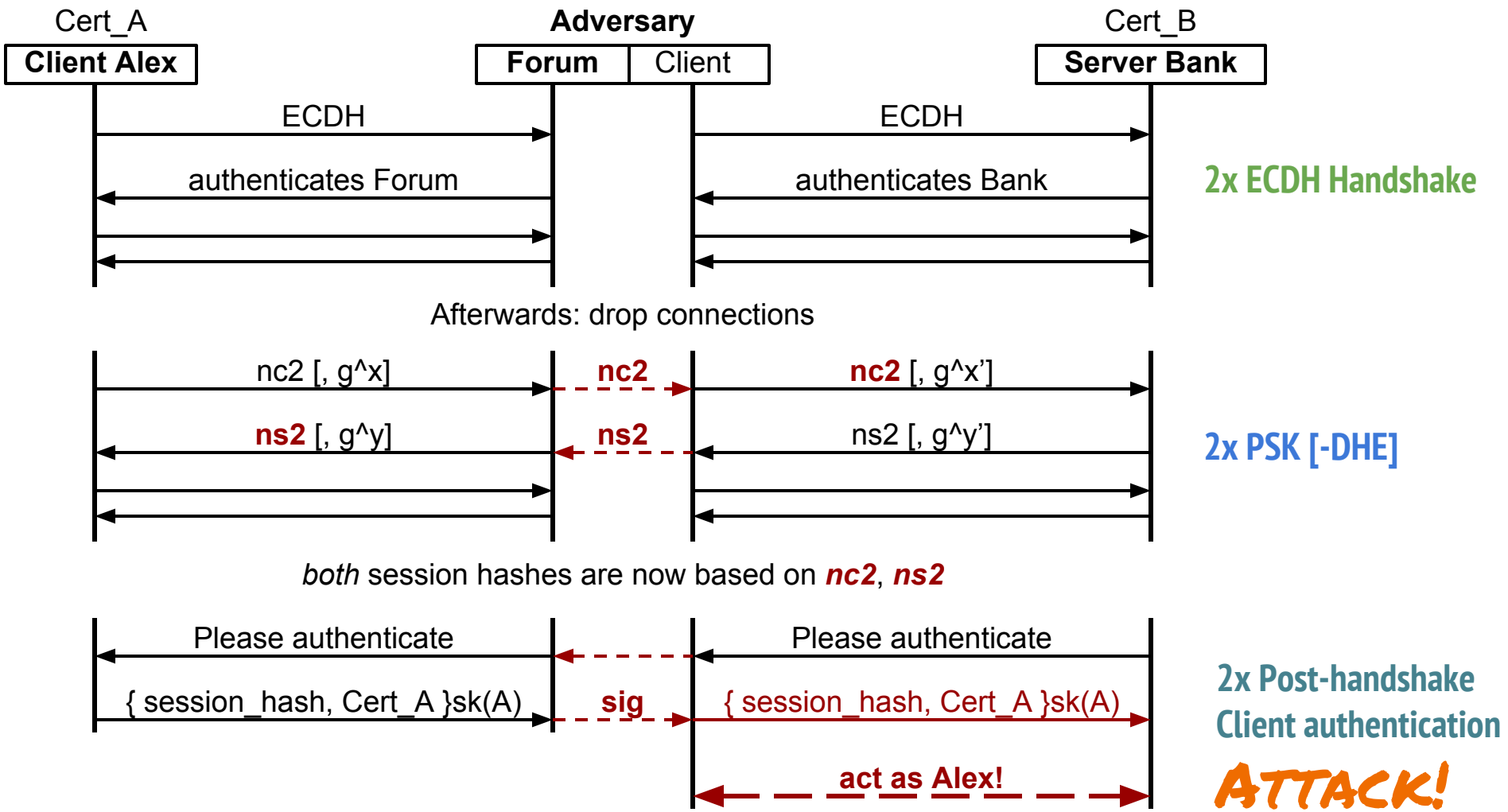
PSK [-DHE]

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



ECDH Handshake

ATTACK SETUP!


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!

Impact

Raised awareness:

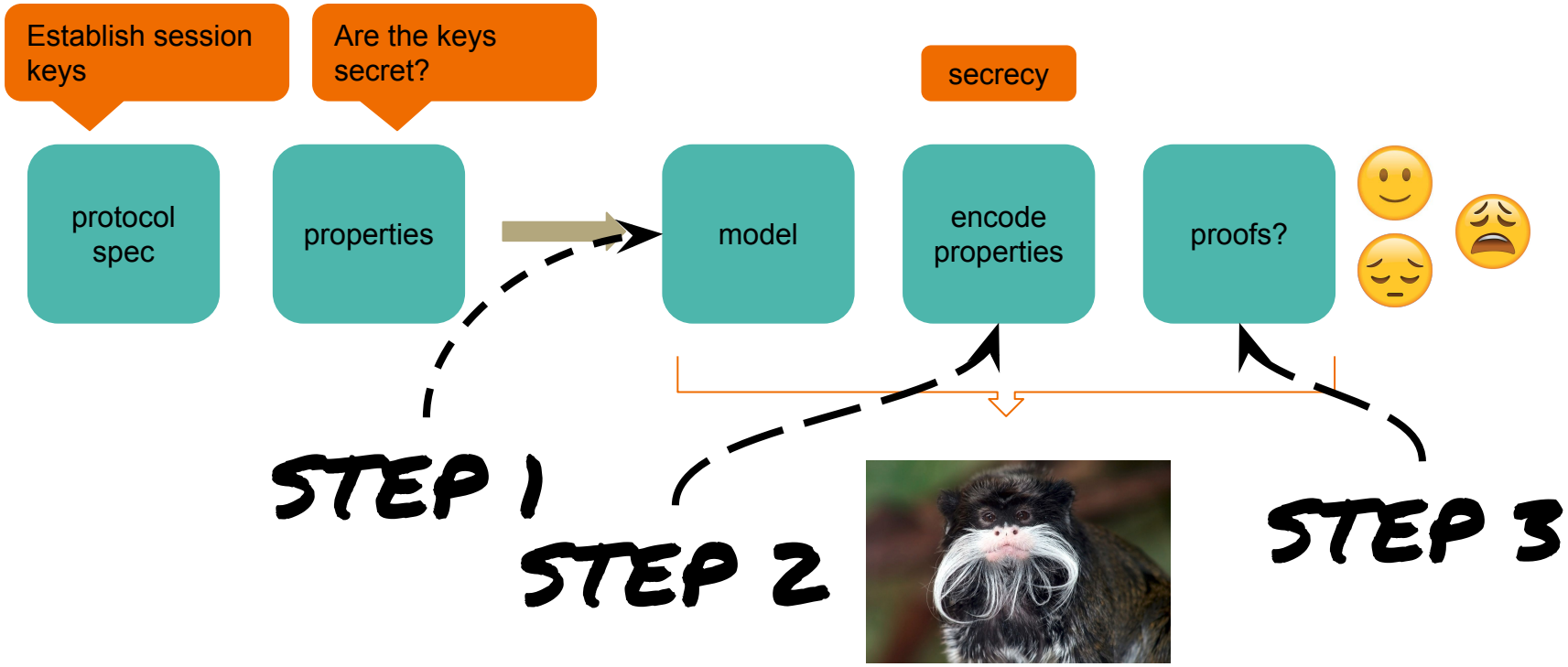
- subtle bugs
- complex to find for humans

Benefits of methodology:

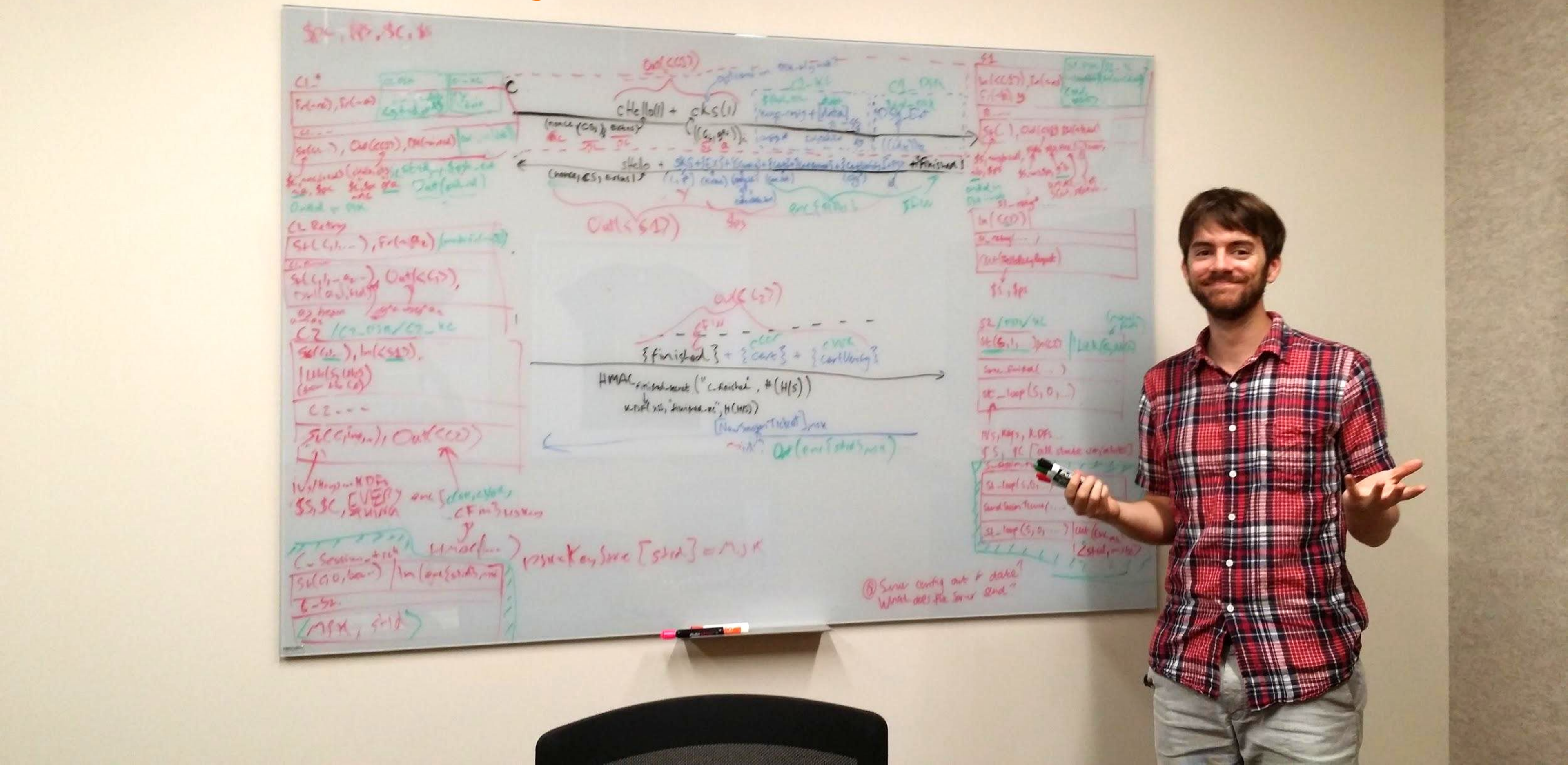
- Provide quicker analysis for proposed designs
- Complements other analysis approaches
    - There currently exists only a symbolic proof of the absence of this attack, and no computational one.


GIVE ME ALLLLL
THE DIRTY DETAILS
memegenerator.net

Analysis Process for TLS 1.3



STEP 1: Building the Model

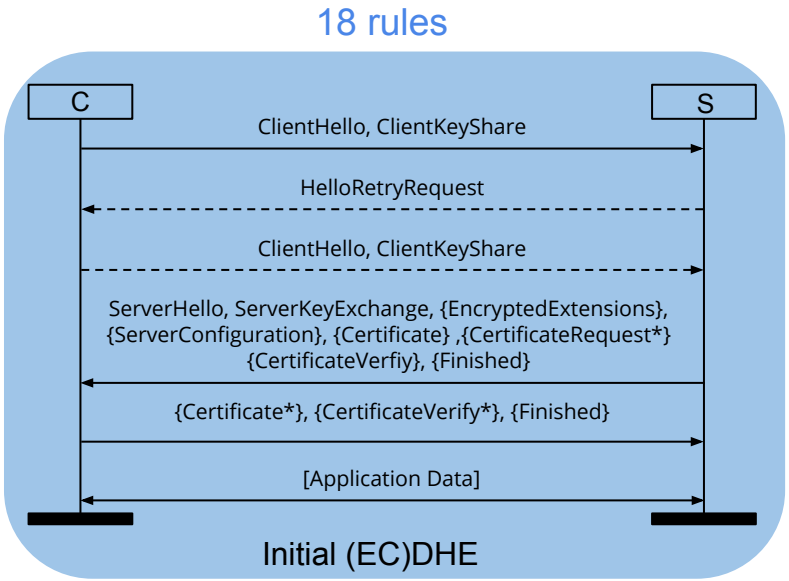


Step 1: Building the Model

- 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

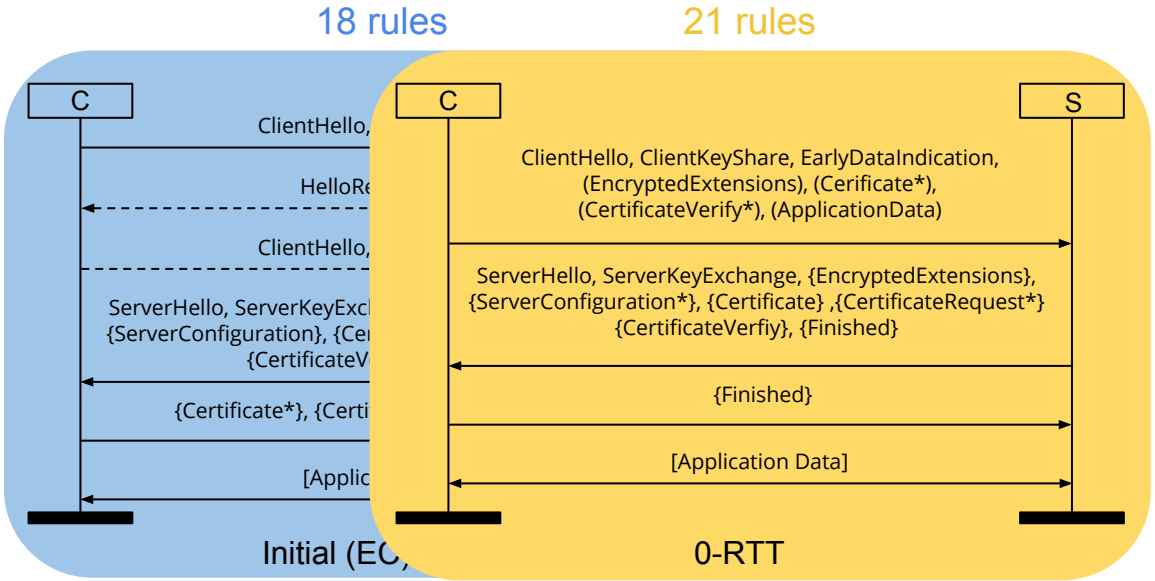
Step 1: Building the Model

- 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



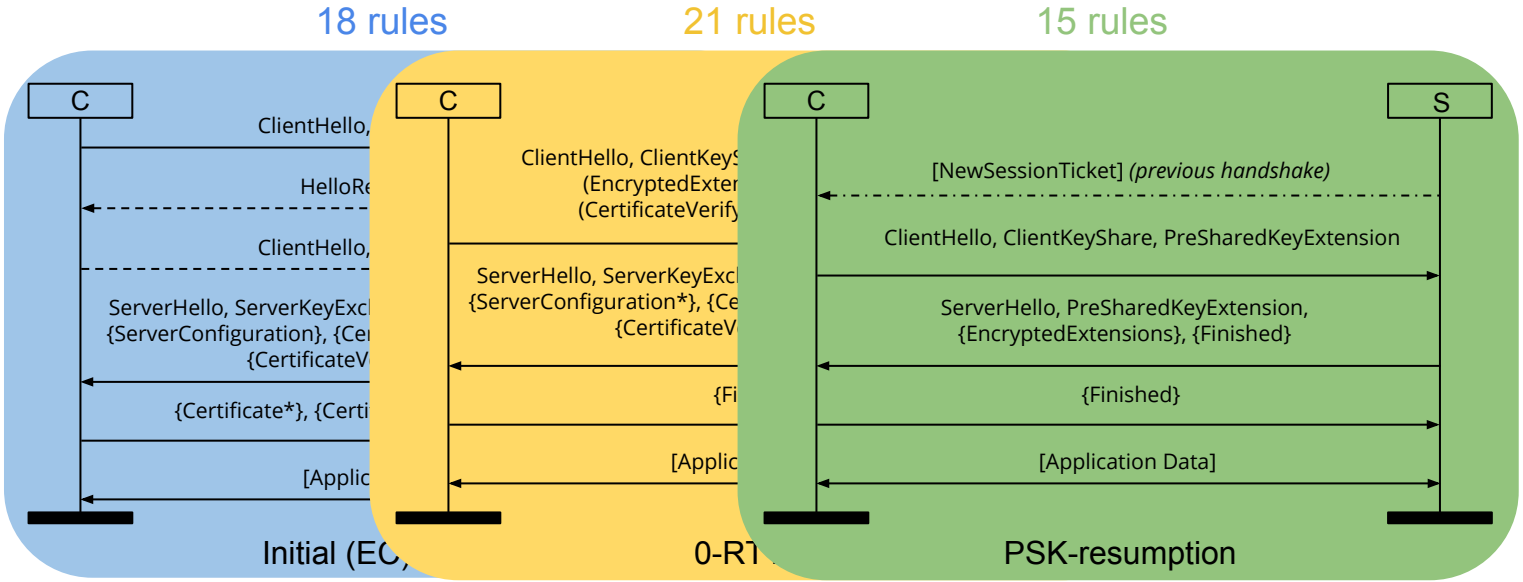
Step 1: Building the Model

- 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



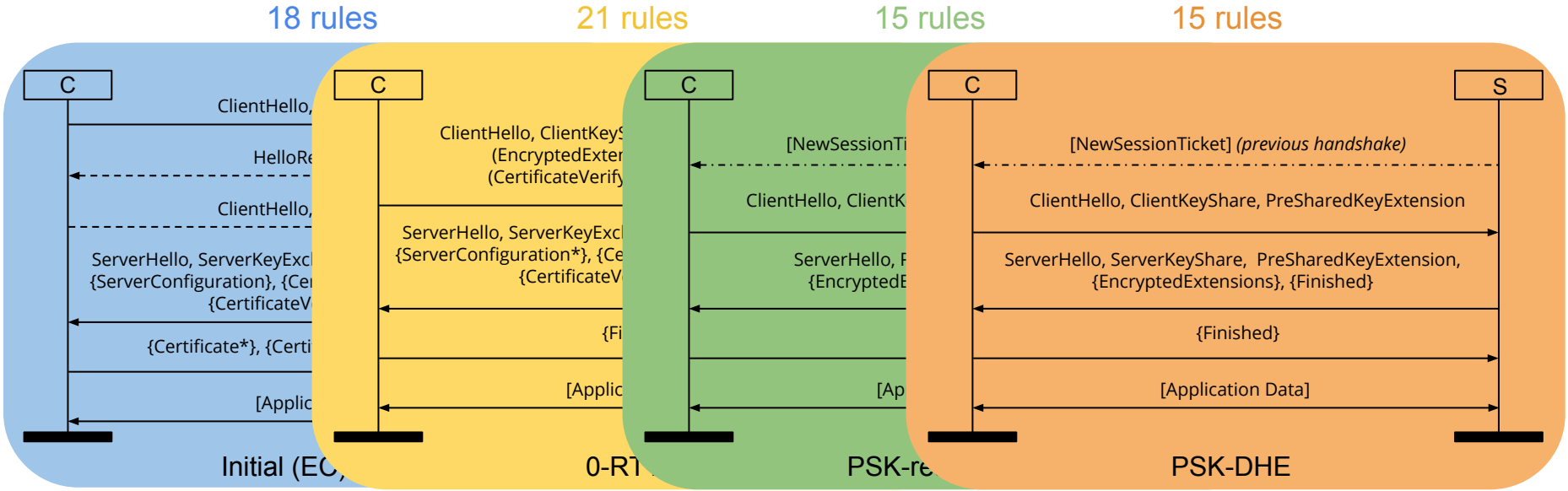
Step 1: Building the Model

- 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



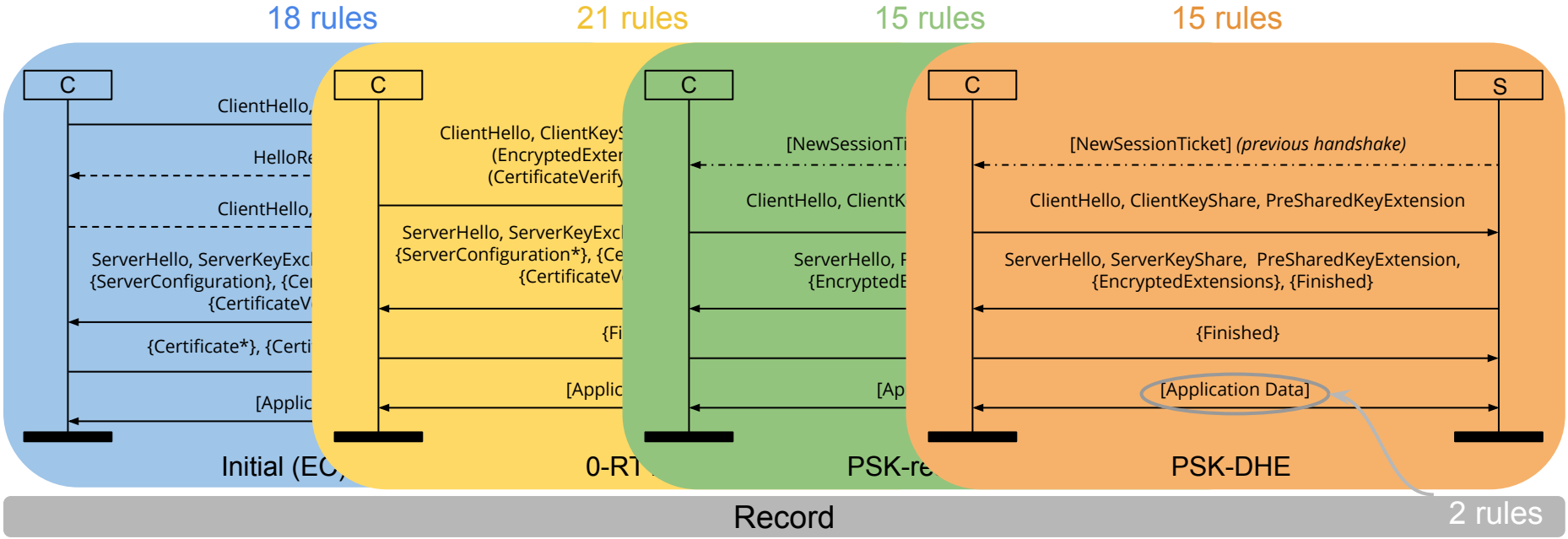
Step 1: Building the Model

- 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



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



Step 1: Client and Server Rules


```
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>)
   ]
```

premises (LHS)

actions

conclusions (RHS)

Step 1: Client and Server Rules


```
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>)
    ]
```

DH built-in

Client state created

Messages going out to the network

Step 1: Client and Server Rules


```
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, m
    , DHExp(C, ~a)
    , Out(<C,C1_MSGS>)
    ]
```

```
rule C_2:
  let
    // Default C2 values
    tid = ~nc
    C   = $C
    nc  = ~nc
    pc  = $pc

    // C2 using DHE (Client Key Share)
    ga   = 'g'^~a

                    ⋮

in
    [ St_init(C,1, tid, C, nc, pc, S, ~a, prev_messages, auth_status)
    , !Pk(S, pk(~ltkS)) // this somehow abstracts the CA verifying the ServerCertificate
    , In(<S,S1_MSGS_1,senc{<S1_MSGS_2,S1_MSGS_3>,ServerFinished}HKEYS>)
    ]
--[ C2(tid)
    , C_ACTIONS
    , UsePK(S, pk(~ltkS))

    , RunningNonces(C, S, 'client', <nc, ns>)
    , RunningSecrets(C, S, 'client', <ss, es>)
    , CommitNonces(C, S, 'client', <nc, ns>)
    , CommitSS(C, S, 'client', ss)
    , CommitES(C, S, 'client', es)
```

Client state accepted by next client rule

Messages coming in From the network

```
rule C_2_NoAuth:
let
    // Default C2 values
    tid = ~nc
    C   = $C
    nc  = ~nc
    pc  = $pc
    S   = $S
    ns  = ~ns
    ps  = $ps

    messages = prev_messages
    hs_hashc = HS_HASH
    client_fin_messages = messages

    // Client Finished
    fs_hash = HS_HASH
    client_fin = hmac(FS, 'client_finished', client_fin_messages)

    hs_messages = messages

    session_hash = HS_HASH

in
    [ St_init(C,2a, INIT_STATE, ss, es, prev_messages, config_hash, auth_status)
    ]
  --[ C2_NoAuth(tid)
    , C_ACTIONS
    , RunningSecrets(C, S, 'client', <ss, es>)
    , RunningTranscript(C, S, 'client', hs_messages)
    , CommitTranscript(C, S, 'client', client_fin_messages)
    , SessionKey(C, S, 'client', <KEYC, 'authenticated'>)
    , SessionKey(C, S, 'client', <KEYS, 'authenticated'>)
    ]->
    [ St_init(C,2, INIT_STATE, ss, es, messages, config_hash, auth_status)
    , Out(<C,senc{ClientFinished}HKEYC>)
    ]
```

SessionKey action logs the session key as computed

Step 1: Is a Complex Task!

- Modelling a complex protocol is not a simple exercise!
- Large number of rules and macros... necessitated by the specification.

Step 1: |

- Modelling
- Large num

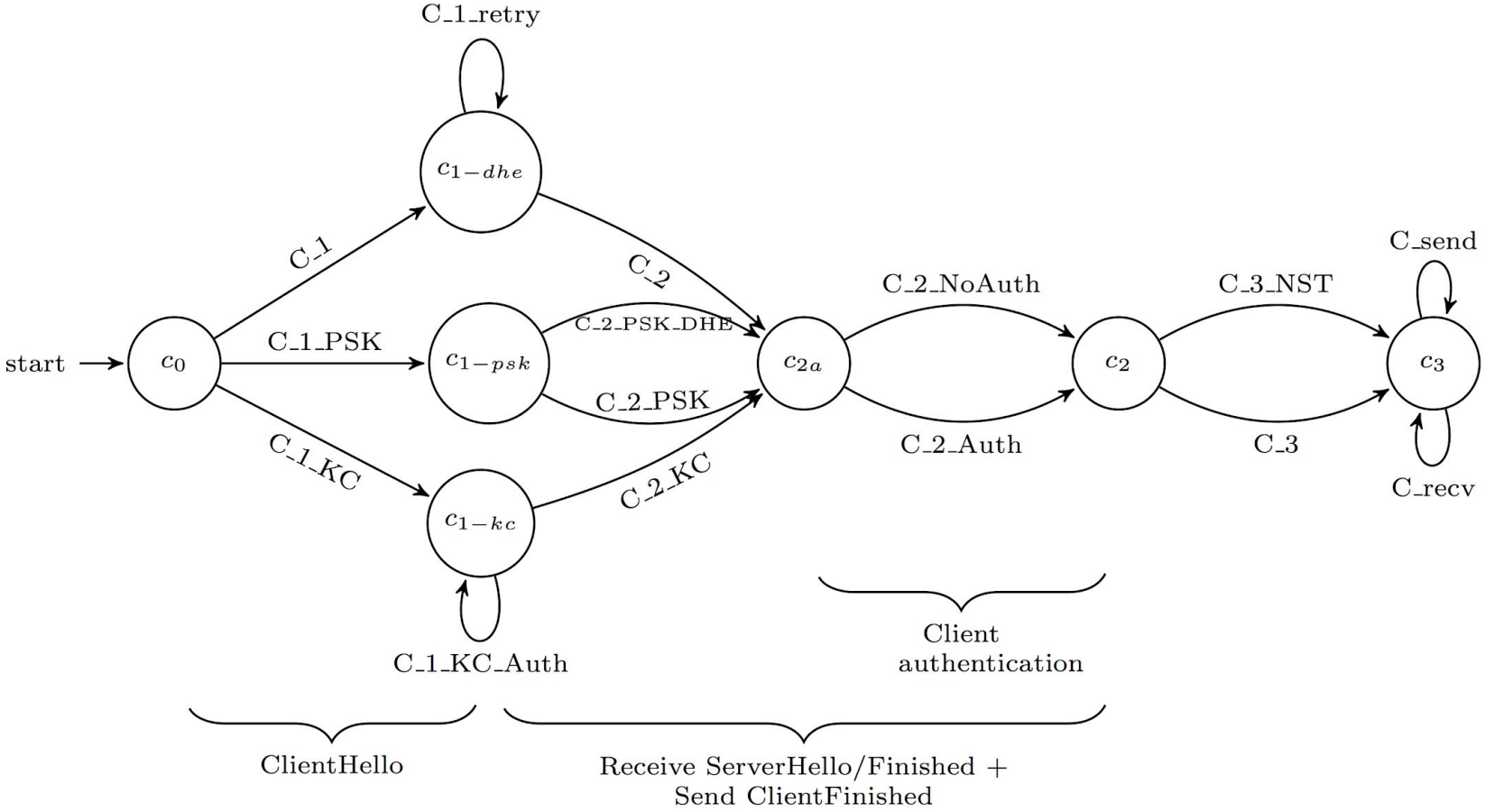
```
define(<!L!>,<!'256'!>)dnl
dnl  Definitions of key derivations
dnl  In each case, the local method should define hs_hash/session_hash
dnl  and also ss and/or es
define(<!xSS!>,<!HKDF('0',ss,'extractedSS',L)!>)dnl
define(<!xES!>,<!HKDF('0',es,'extractedES',L)!>)dnl
define(<!MS!>,<!HKDF(xSS,xES,'master_secret',L)!>)dnl
define(<!FS!>,<!HKDFExpand(xSS, 'finished_secret', fs_hash,L)!>)dnl
define(<!RS!>,<!HKDFExpand(MS, 'resumption_master_secret', session_hash,L)!>)dnl
dnl
define(<!EDKEYC!>,<!HKDFExpand1(xSS,'early_data_key_expansion',hs_hashc,L)!>)dnl
dnl
define(<!HKEYC!>,<!HKDFExpand1(xES, 'handshake_key_expansion', hs_hashc,L)!>)dnl
define(<!HKEYS!>,<!HKDFExpand2(xES, 'handshake_key_expansion', hs_hashs,L)!>)dnl
dnl
dnl Application keys should likely differ. 07 draft may "be revised" anyway.
define(<!KEYC!>,<!HKDFExpand1(MS, 'application_data_key_expansion', session_hash,L)!>)dnl
define(<!KEYS!>,<!HKDFExpand2(MS, 'application_data_key_expansion', session_hash,L)!>)dnl
dnl
```

Key computations

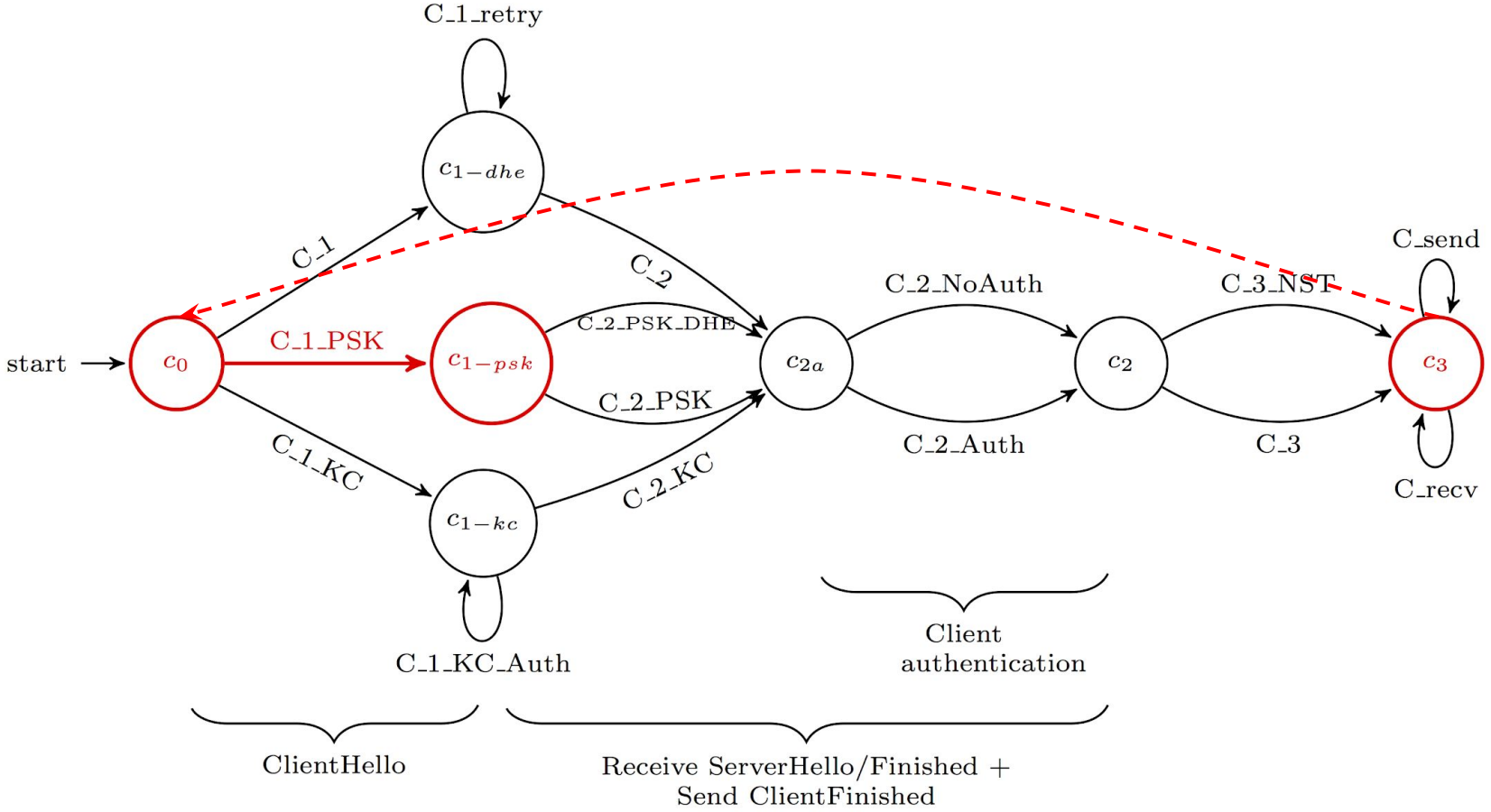
```
dnl Definition of C1 (client's handshake message)
define(<!ClientHello!>,<!nc, pc!>)dnl
define(<!ClientKeyShare!>,<!ga!>)dnl
dnl
dnl Definition of S1_Retry
define(<!HelloRetryRequest!>,<!ps!>)dnl
dnl
dnl Definition of S1
define(<!ServerHello!>,<!ns, ps!>)dnl
define(<!ServerKeyShare!>,<!gb!>)dnl
define(<!ServerFinished!>,<!server_fin!>) dnl hmac(FS, 'server_finished', hs_hash)dnl
dnl
dnl Additional messages sent in S1
define(<!ServerConfiguration!>,<!Y!>)dnl
define(<!ServerEncryptedExtensions!>,<!$exts!>) dnl i.e. abstracted awaydnl
define(<!ServerCertificate!>,<!pk(~ltkS)!>) dnl  Abstracted away as PKI infrastructure with !Pk facts given to clientdnl
define(<!ServerCertificateVerify!>,<!s_signature!>)dnl
define(<!CertificateRequest!>,<!$cert_req!>)dnl
```

Macros for just 3 of our rules!

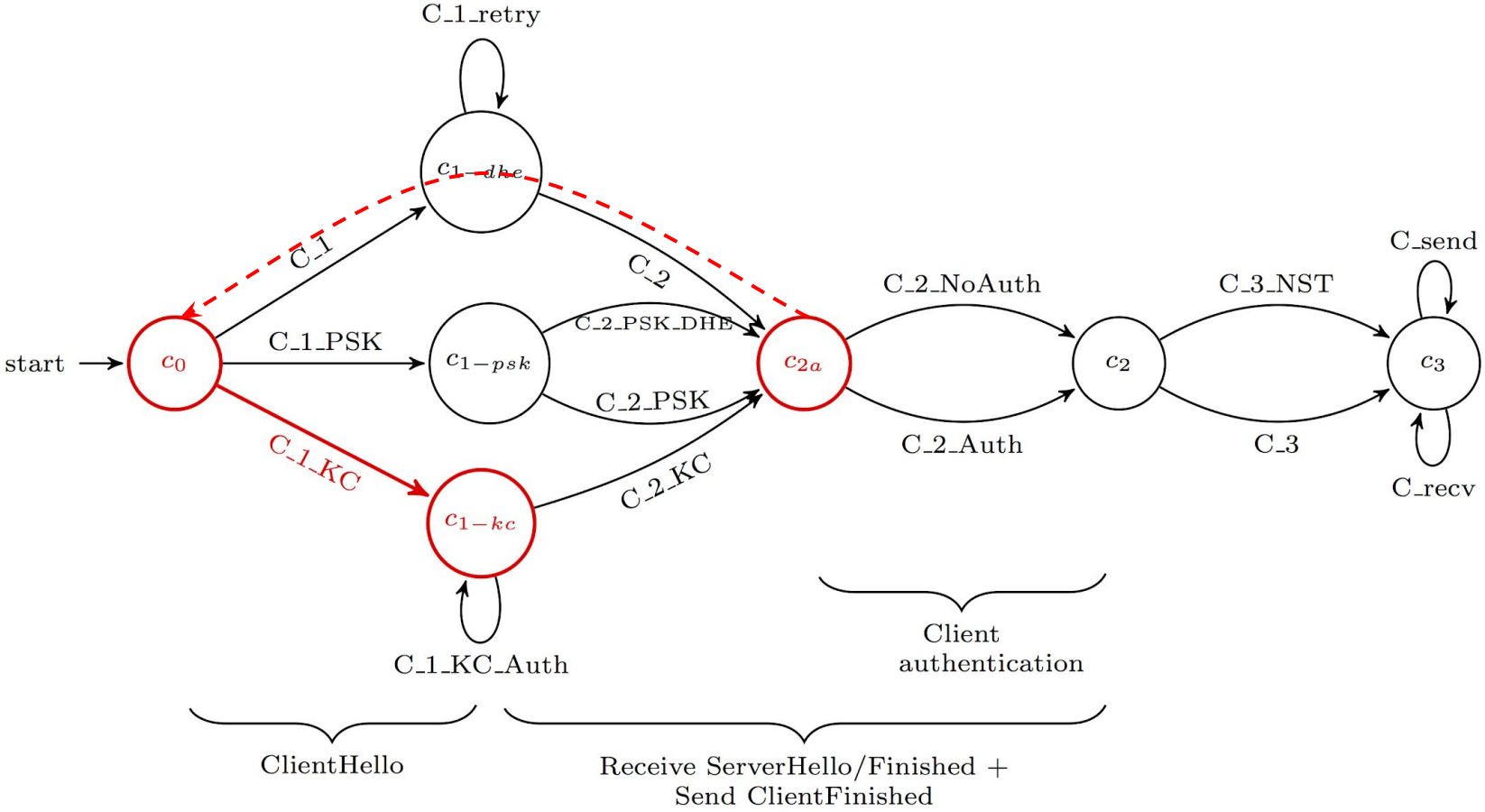
Step 1: Building the Model



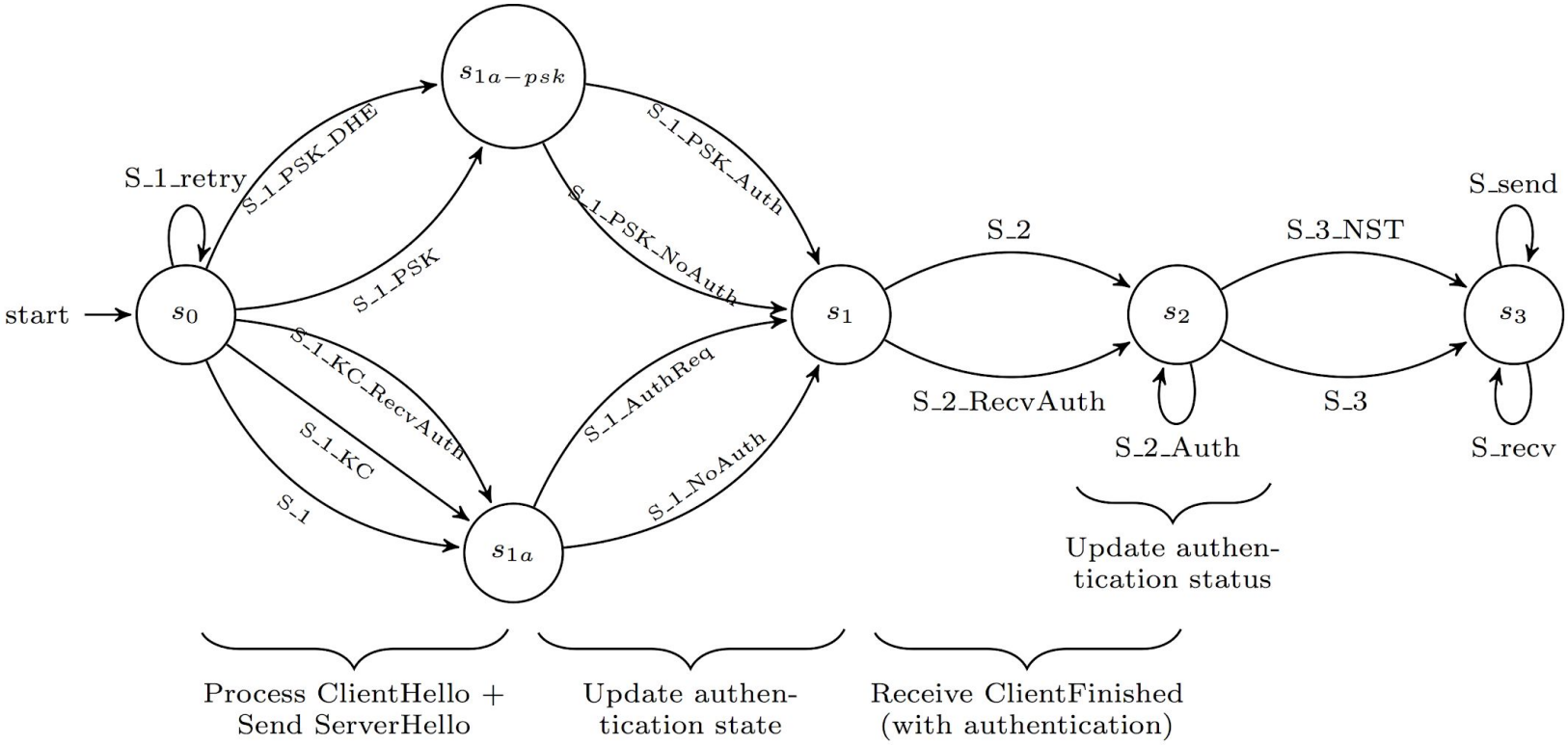
Step 1: Building the Model



Step 1: Building the Model



Step 1: Building the Model



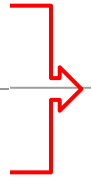
Step 1: Adversarial Capabilities

- 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

Step 2: Encoding 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 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).

Step 2: Encoding 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
```

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

Step 2: Encoding Properties

Constructed Tamarin encodings for all of the main properties:

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

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’
- Let’s get 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
by sorry

lemma secret_early_data_keys:
  all-traces
  "∀ actor peer k #i.
        ((EarlyDataKey( actor, peer, 'client', k ) @
#i) ∧
         (¬(∃ #r. RevLtk( peer ) @ #r))) ⇒
        (¬(∃ #j. !KU( k ) @ #j))"
simplify
by sorry

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))"
simplify
by sorry

lemma transcript_agreement [reuse]:
```

Visualization display

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

1. **solve(** SessionKey(actor, peer, role,
   <k, 'authenticated'>
) @ #i **)** // nr. 0

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

Constraint system

last: none

formulas:
∀ #r. (RevLtk(actor) @ #r) ⇒ ¬(#r < #i)

∀ #r. (RevLtk(peer) @ #r) ⇒ ¬(#r < #i)

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) ∧

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
by sorry

lemma secret_early_data_keys:
  all-traces
  "∀ actor peer k #i.
        ((EarlyDataKey( actor, peer, 'client', k ) @
#i) ∧
         (¬(∃ #r. RevLtk( peer ) @ #r))) ⇒
        (¬(∃ #j. !KU( k ) @ #j))"
simplify
by sorry

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))"
simplify
by sorry

lemma transcript_agreement [reuse]:
```

encoding

Visualization display

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

1. **solve(** SessionKey(actor, peer, role, <k, 'authenticated'>) @ #i **)** // nr. 0

proof approaches

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

Constraint system

last: none

formulas:
∀ #r. (RevLtk(actor) @ #r) ⇒ ¬(#r < #i)

∀ #r. (RevLtk(peer) @ #r) ⇒ ¬(#r < #i)

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) ∧

constraint system

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
  by sorry
next
  case C_2_Auth_case_2
  by sorry
next
  case C_2_NoAuth_case_1
  by sorry
next
  case C_2_NoAuth_case_2
  by sorry
next
  case S_2_Auth_case_1
  by sorry
next
  case S_2_Auth_case_2
  by sorry
next
  case S_2_case_1
  by sorry
next
  case S_2_case_2
  by sorry
qed
```

Visualization display

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

1. **solve(** F_St_C_2a_init(~nc, $C, ~nc, $pc, $S, ~ns, $ps, ss, es,
   prev_messages, config_hash, 'no_auth'
) ▶$_0$ #i **)** // nr. 3 (from rule C_2_Auth)

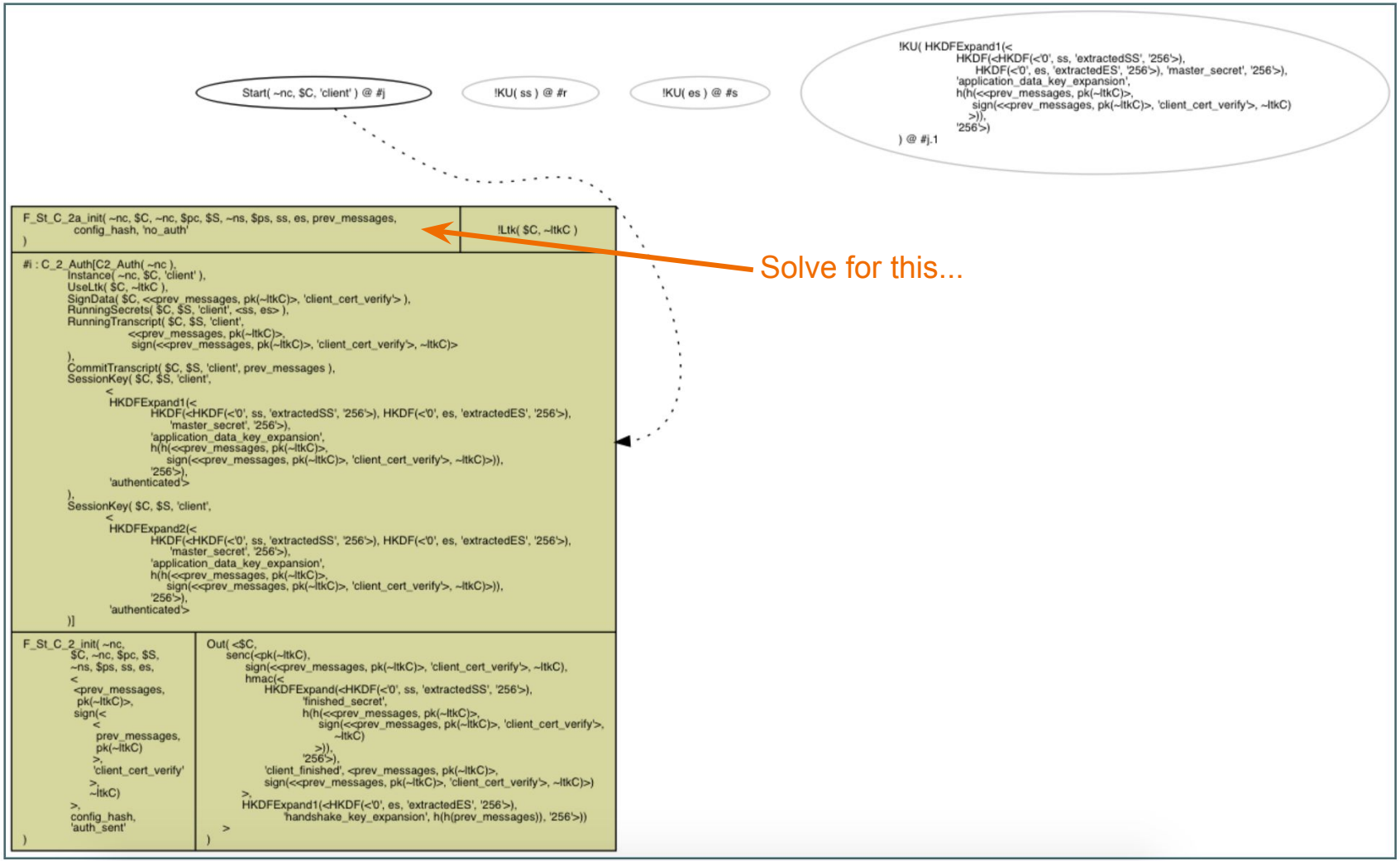
2. **solve(** !Ltk($C, ~ltkC) ▶$_1$ #i **)** // nr. 4 (from rule C_2_Auth)

3. **solve(** Start(~nc, $C, 'client') @ #j **)** // nr. 8

4. **solve(** !KU(HKDFExpand1(<
   HKDF(<
   HKDF(<'0', ss, 'extractedSS', '256'
   >),
   HKDF(<'0', es, 'extractedES', '256'
   >),
   'master_secret', '256'>),
   'application_data_key_expansion',
   h(h(<<prev_messages, pk(~ltkC)>,
   sign(<<prev_messages, pk(~ltkC)>,
   'client_cert_verify'>,
   ~ltkC)
   >)),
   '256'>)
) @ #j.1 **)** // nr. 5

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

Constraint system


Start(~nc, $C, 'client') @ #j
!KU(ss) @ #r
!KU(es) @ #s
!KU(HKDFExpand1(<
HKDF(<HKDF(<'0', ss, 'extractedSS', '256'>),
HKDF(<'0', es, 'extractedES', '256'>), 'master_secret', '256'>),
'application_data_key_expansion',
h(h(<<prev_messages, pk(~ltkC)>,
sign(<<prev_messages, pk(~ltkC)>, 'client_cert_verify'>, ~ltkC)
>)),
'256'>)
) @ #j.1
F_St_C_2a_init(~nc, $C, ~nc, $pc, $S, ~ns, $ps, ss, es, prev_messages,
config_hash, 'no_auth'
)
!Ltk($C, ~ltkC)
Solve for this...
#i : C_2_Auth[C2_Auth(~nc),
Instance(~nc, $C, 'client'),
UseLtk($C, ~ltkC),
SignData($C, <<prev_messages, pk(~ltkC)>, 'client_cert_verify'>),
RunningSecrets($C, $S, 'client', <ss, es>),
RunningTranscript($C, $S, 'client',
<<prev_messages, pk(~ltkC)>,
sign(<<prev_messages, pk(~ltkC)>, 'client_cert_verify'>, ~ltkC)>
),
CommitTranscript($C, $S, 'client', prev_messages),
SessionKey($C, $S, 'client',
<
HKDFExpand1(<
HKDF(<HKDF(<'0', ss, 'extractedSS', '256'>), HKDF(<'0', es, 'extractedES', '256'>),
'master_secret', '256'>),
'application_data_key_expansion',
h(h(<<prev_messages, pk(~ltkC)>,
sign(<<prev_messages, pk(~ltkC)>, 'client_cert_verify'>, ~ltkC)>)),
'256'>),
'authenticated'>
),
SessionKey($C, $S, 'client',
<
HKDFExpand2(<
HKDF(<HKDF(<'0', ss, 'extractedSS', '256'>), HKDF(<'0', es, 'extractedES', '256'>),
'master_secret', '256'>),
'application_data_key_expansion',
h(h(<<prev_messages, pk(~ltkC)>,
sign(<<prev_messages, pk(~ltkC)>, 'client_cert_verify'>, ~ltkC)>)),
'256'>),
'authenticated'>
)]
F_St_C_2_init(~nc,
$C, ~nc, $pc, $S,
~ns, $ps, ss, es,
<
<prev_messages,
pk(~ltkC)>,
sign(<
<
prev_messages,
pk(~ltkC)
>,
'client_cert_verify'
>,
~ltkC)
>,
config_hash,
'auth_sent'
)
Out(<$C,
senc(<pk(~ltkC),
sign(<<prev_messages, pk(~ltkC)>, 'client_cert_verify'>, ~ltkC),
hmac(<
HKDFExpand(<HKDF(<'0', ss, 'extractedSS', '256'>),
'finished_secret',
h(h(<<prev_messages, pk(~ltkC)>,
sign(<<prev_messages, pk(~ltkC)>, 'client_cert_verify'>,
~ltkC)
>)),
'256'>),
'client_finished', <prev_messages, pk(~ltkC)>,
sign(<<prev_messages, pk(~ltkC)>, 'client_cert_verify'>, ~ltkC)>)
>,
HKDFExpand1(<HKDF(<'0', es, 'extractedES', '256'>),
'handshake_key_expansion', h(h(prev_messages)), '256'>))
>
)

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
  by sorry
next
  case C_2_NoAuth_case_1
  by sorry
next
  case C_2_NoAuth_case_2
  by sorry
```

Visualization display

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

1. **solve(** F_St_C_2a_init(~nc, $C, ~nc, $pc, $S, ~ns, $ps, ss, es,
   prev_messages, config_hash, 'no_auth'
) ▶₀ #i **)** // nr. 3 (from rule C_2_Auth)

2. **solve(** !Ltk($C, ~ltkC) ▶₁ #i **)** // nr. 4 (from rule C_2_Auth)

3. **solve(** Start(~nc, $C, 'client') @ #j **)** // nr. 8

4. **solve(** !KU(HKDFExpand2(<
   HKDF(<
   HKDF(<'0', ss, 'extractedSS', '256'
   >),
   HKDF(<'0', es, 'extractedES', '256'
   >),
   'master_secret', '256'>),
   'application_data_key_expansion',
   h(h(<<prev_messages, pk(~ltkC)>,
   sign(<<prev_messages, pk(~ltkC)>,
   'client_cert_verify'>,
   ~ltkC)
   >)),
   '256'>)
) @ #j.1 **)** // nr. 5

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

Constraint system

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:

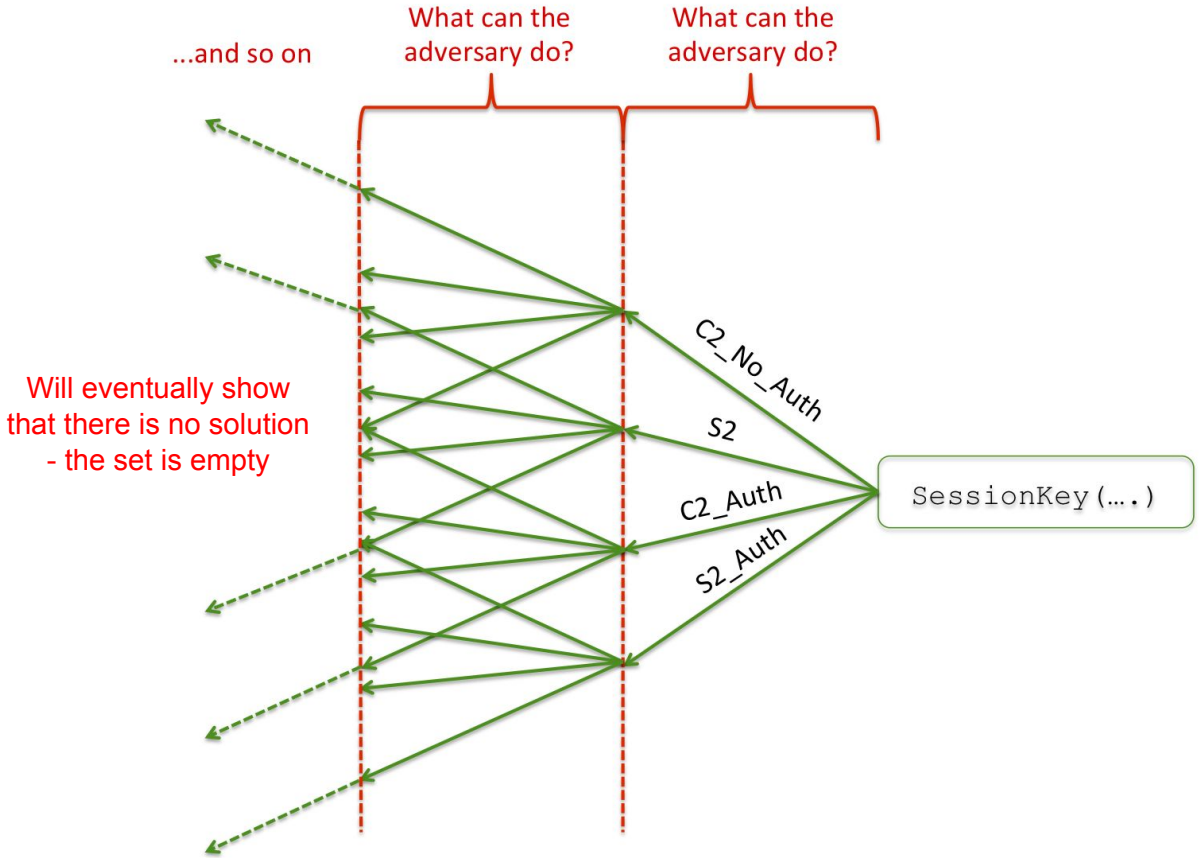
conj:

lemmas:

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

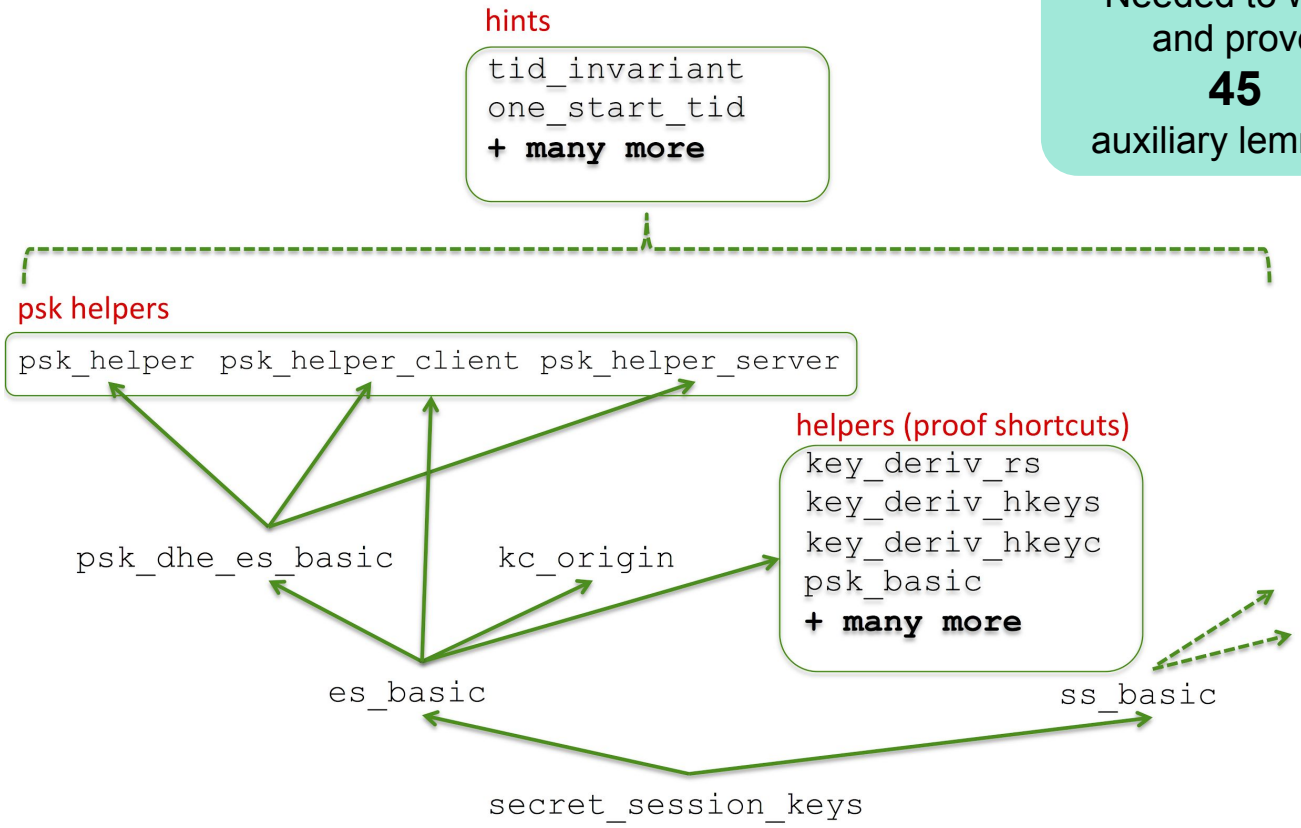
∀ actor actor2 psk_id psk_id2 peer peer2 rs auth_status
auth_status2 #i #j.
(UsePSK(actor, psk_id, peer, rs, 'client', auth_status
) @ #i) ∧
(UsePSK(actor2, psk_id2, peer2, rs, 'server', auth_status2
) @ #j)

Step 3: Producing Proofs



Step 3: Producing Proofs

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



Step 3: Producing Proofs


```
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))"
```

Step 3: Producing Proofs


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

+ mutual

Step 3: Producing Proofs


```
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

Step 3: Producing Proofs


```
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
          transc
```

```
lemma secret_early_data_keys:
  all-traces
  "∀ actor peer k #i.
        ((EarlyDataKey( actor, peer, 'client', k ) @
#i) ∧
         (¬(∃ #r. RevLtk( peer ) @ #r))) ⇒
        (¬(∃ #j. !KU( k ) @ #j))"
```

Step 3: Producing Proofs


```
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

- You've seen the message flows of the attack

- BUT how did we find it?!

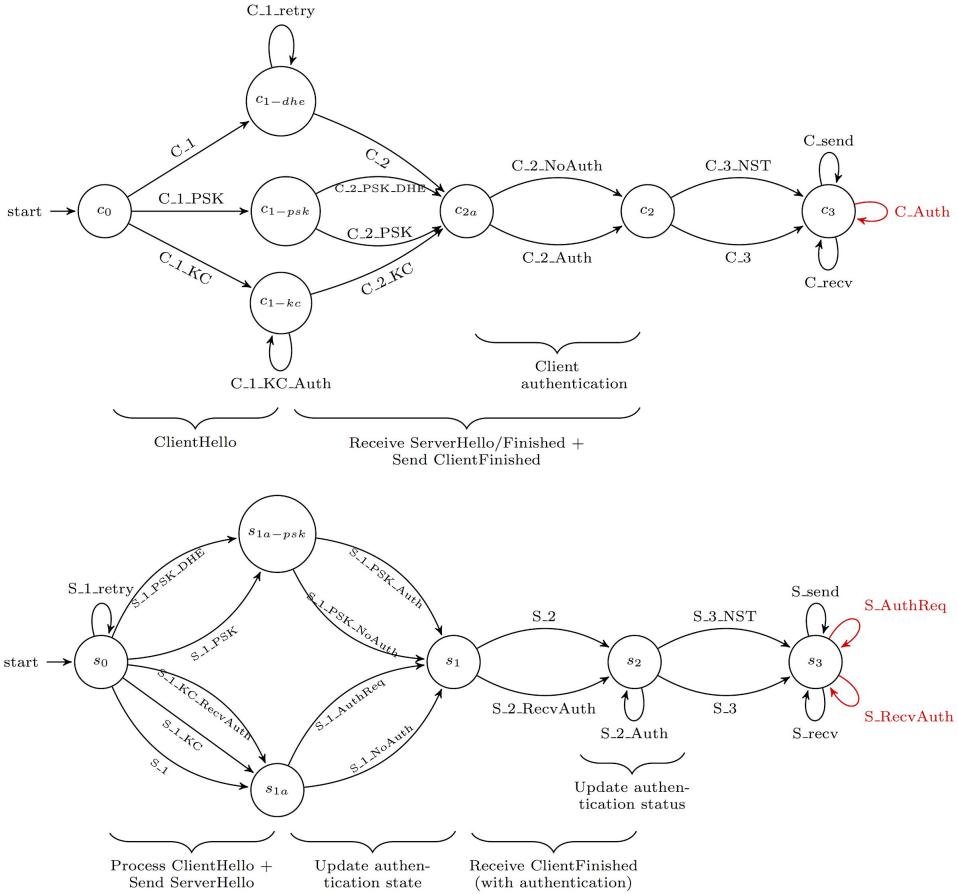
2x ECDH Handshake

2x PSK [-DHE]

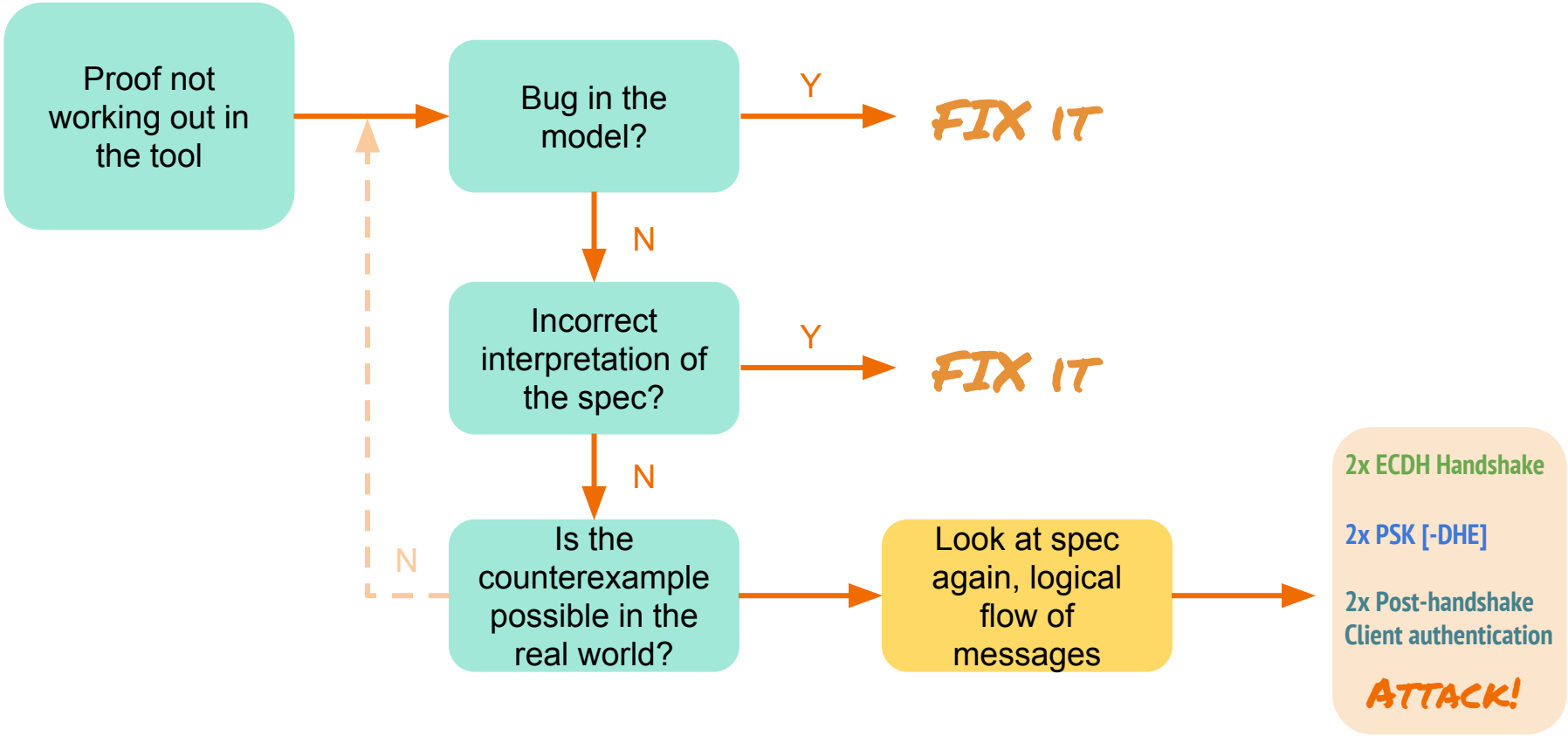
2x Post-handshake
Client authentication

ATTACK!

Finding An Attack



Finding An Attack



Step 1: Building the Model

- TLS 1.3 has been 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:

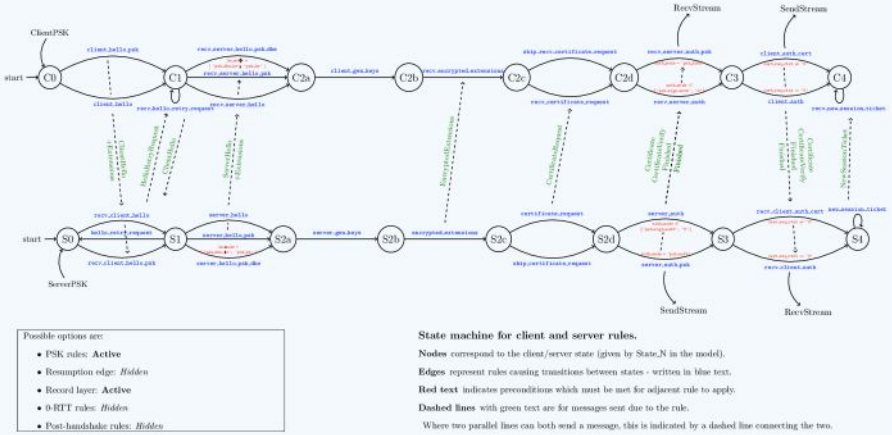
- (EC)DHE (Diffie-Hellman both the finite field and elliptic curve varieties),
- PSK-only, and
- PSK with (EC)DHE

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.

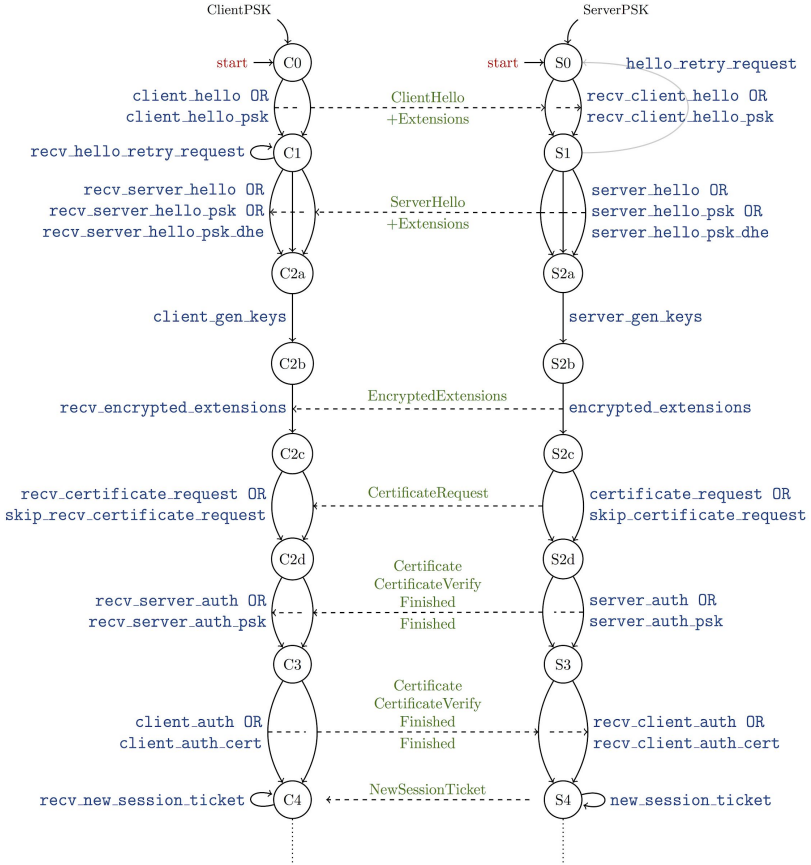
Dashed lines with green text are for messages sent due to the rule.

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

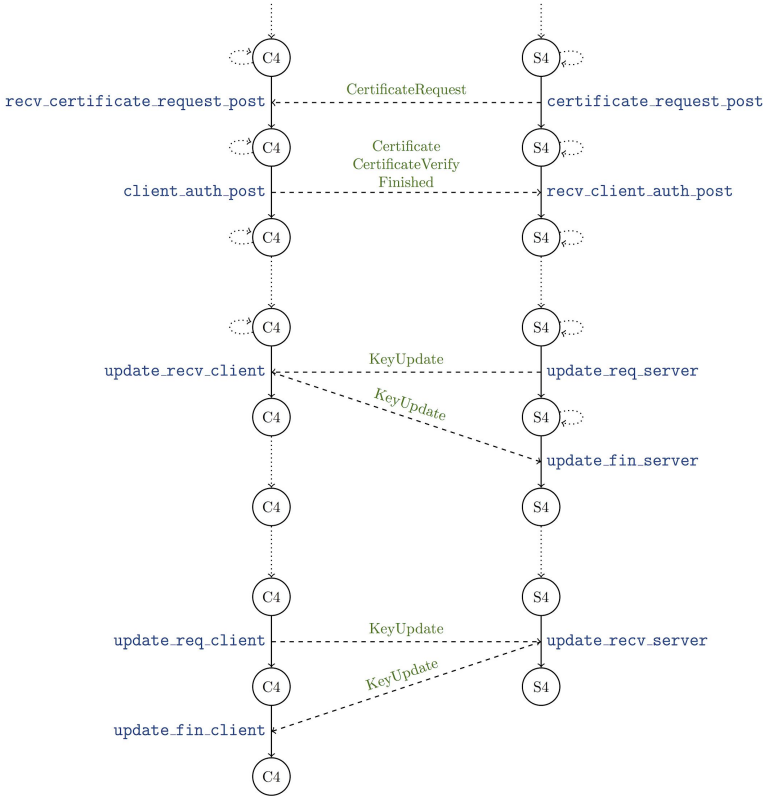
For example, we see that a PSK-only handshake is captured through the invocation of the rules `client_hello_psk -> recv_client_hello_psk -> server_hello_psk -> ...` for the PSK-DHE handshake, the rule `server_hello_psk_dhe` would be used instead.

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

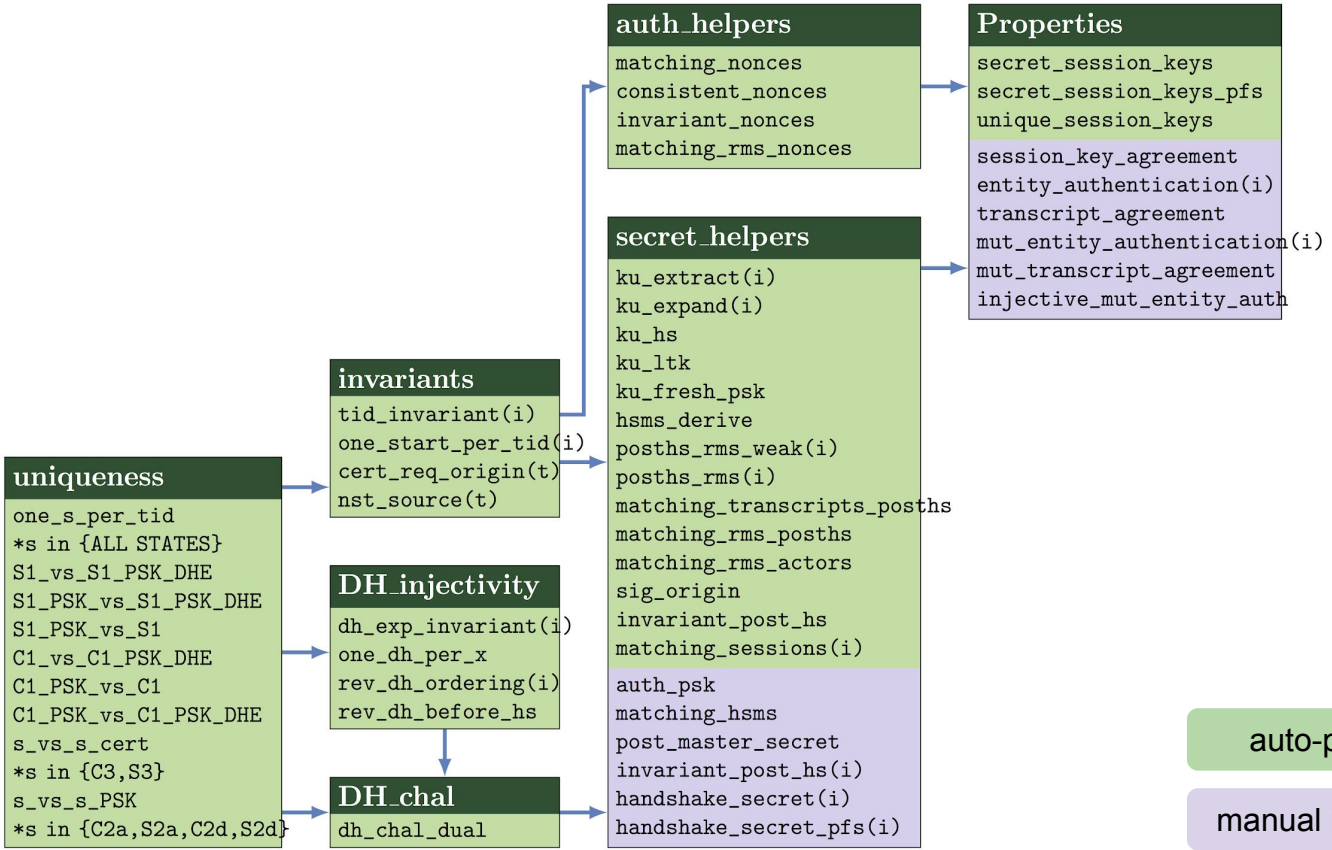
Step 1: Building the Model



Step 1: Building the Model



Step 2: Encoding Security Properties



Step 3: Producing Proofs

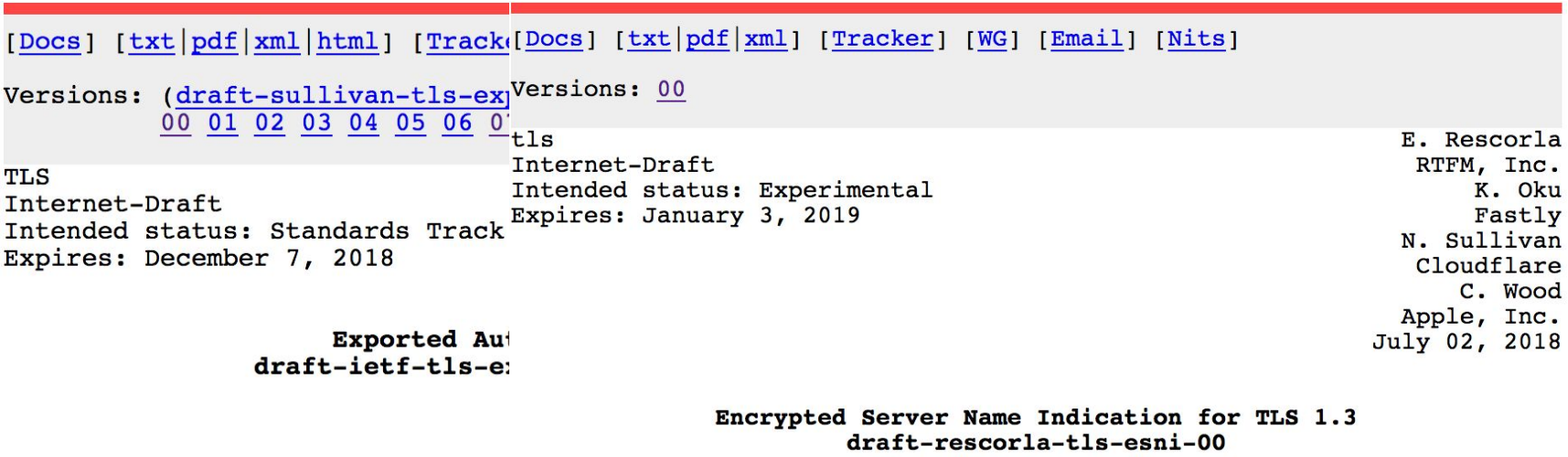
Security Property	
Establishing the same session keys	✓
Secret session keys	✓
Peer authentication	✓ See [CHHMS17]
Uniqueness of session keys	✓
Downgrade protection (within 1.3)	✓
Perfect forward secrecy	✓
Key Compromise Impersonation (KCI) resistance	✓

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

Future Work

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



- TLS 1.1 and TLS 1.2 for protocol version downgrades

Takeaways

- Logical core of TLS 1.3 seems sound!
- We have built a transparent model others can build on (Github)
- Symbolic analysis
  - Complementary approach to other analysis methods
- Relatively fast turnaround and can directly produce attacks

The future is bright!

Tamarin Tutorial at Eurocrypt in Darmstadt. See you there!

May 19 - 23, 2019

cas.cremers@cispa.saarland, tvandermerwe@mozilla.com

https://tls13tamarin.github.io/TLS13Tamarin/

Bonus Slide

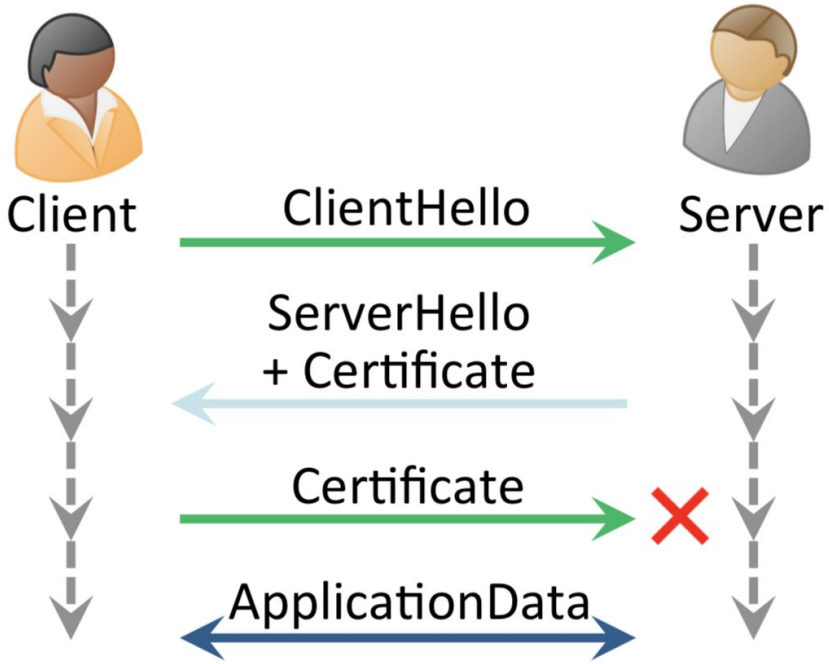


Figure: The awkward handshake.

See [CHHMS17] for details.

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/