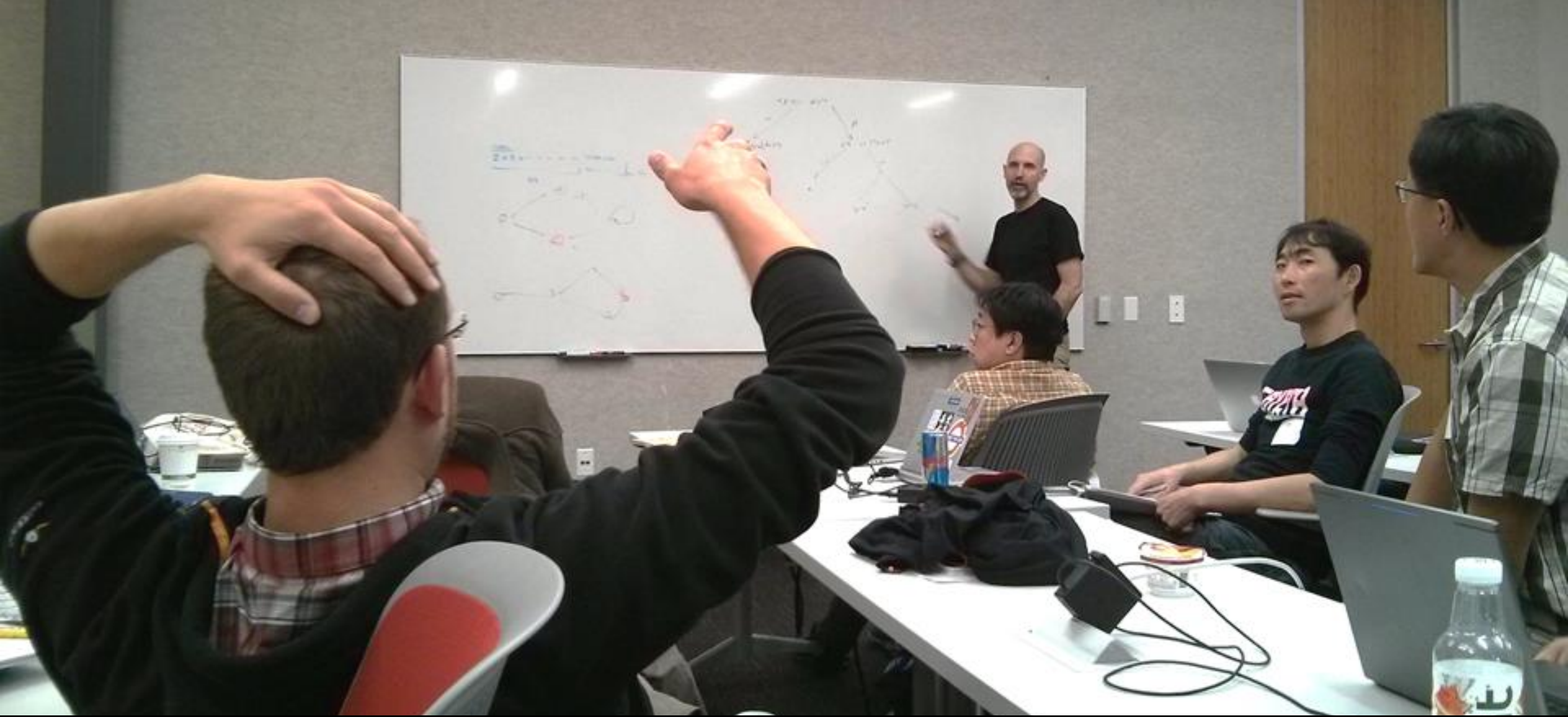
CISPA
HELMHOLTZ CENTER FOR
INFORMATION SECURITY

Formal Analysis of Real-World Security Protocols

Lecture 10: Security Protocol Standards

Cas Cremers | 27.01.2025


Hello
Barbie



May 2016

Mozilla HQ, Mountain View, CA, USA

This lecture

- Security protocol standards
- IETF TLS 1.3 (Book chapter 19.1)

Security Protocol Standards

Who creates security protocols?

- **Standardization bodies**
  - **ISO** "International Organization for Standardization"
    NIST "National Institute of Standards and Technology"
    IETF "Internet Engineering Task Force"
    ...
- **Industry groups**
  - 3GPP "3rd Generation Partnership Project (3GPP)"
    (Global, standardizes mobile telecommunications technology)
- **Companies/individuals**
  - Secure messengers, door locks, OS vendors, car vendors, ...

ISO: International Organization for Standardization

"ISO was founded on 23 February 1947, and (as of July 2024) it has published over 25,000 international standards covering almost all aspects of technology and manufacturing. It has over 800 technical committees (TCs) and subcommittees (SCs) to take care of standards development." (Source: Wikipedia)

Structure: Member states can propose new standards or modifications to TCSubcommittees responsible for standards

Examples:

- "ISO/IEC 9798: Information technology — Security techniques — Entity authentication"
- ISO/IEC 11770 for key establishment mechanisms.

NIST: National Institute of Standards and Technology

"Promote U.S. innovation and industrial competitiveness by advancing measurement science, standards, and technology in ways that enhance economic security and improve our quality of life."
(Source: Wikipedia)

Examples:

- AES (Advanced Encryption Standard)
- SHA-2 (SHA256, etc)
- "Post-Quantum Cryptography Standardization"

IETF: Internet Engineering Task Force

"[IETF] has no formal membership roster or requirements and all its participants are volunteers. Their work is usually funded by employers or other sponsors. [...] Anyone can participate by signing up to a working group mailing list, or registering for an IETF meeting."
(Source: Wikipedia)

3 yearly meetings across the world, 1000-1500 participants.
Next meeting: IETF 122 Bangkok 15 Mar 2025 - 21 Mar 2025

Examples:

- TLS, DNS[SEC], UDP, TCP, IKE, MLS

IETF standards development

Documents called **RFC**s

"Request For Comments" (historical name)

Discussion on mailing lists

Main decisions taken in physical meetings

- "On Consensus and Humming in the IETF" (https://datatracker.ietf.org/doc/html/rfc7282)

IETF TLS 1.3

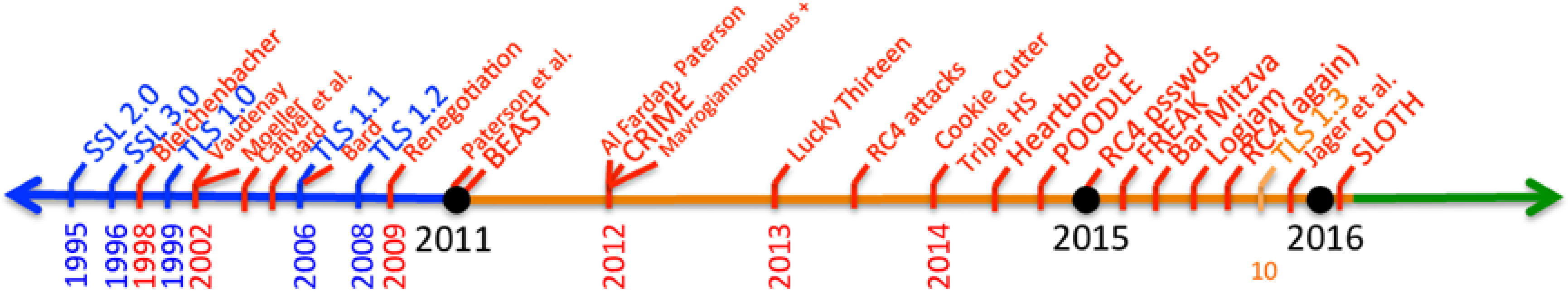
TLS: "Transport Layer Security"

- *"Transport Layer Security (TLS) is a* ***cryptographic protocol designed to provide communications security over a computer network****, such as the Internet. The protocol is widely used in applications such as email, instant messaging, and voice over IP, but its use in securing* ***HTTPS*** *remains the most publicly visible."* (Source: Wikipedia)

- The most used security protocol globally
  - Approximately 1 billion TLS handshakes per second globally
  - Amazon's AWS alone does about 50 million per second

Security of TLS over time

Security of TLS over time



TLS 1.3 Development

Fall 2018: TLS 1.3

- Led by the IETF
- Editor: Eric Rescorla
- Development: 2014–2018

https://datatracker.ietf.org/doc/html/rfc8446

160 pages (but dependent on many related standards)

TLS 1.3 analysis using Tamarin

Cas Cremers

Marko Horvat

Sam Scott

Thyla van der Merwe

UNIVERSITY OF OXFORD

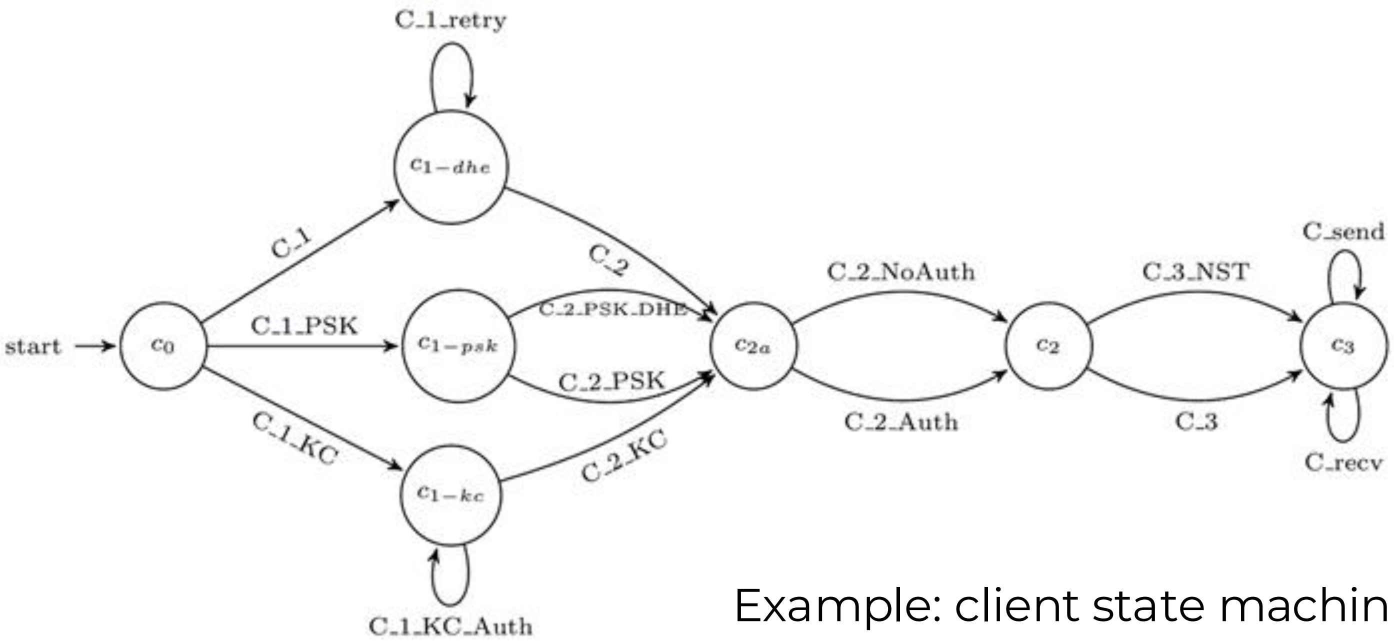
ROYAL HOLLOWAY UNIVERSITY OF LONDON

- 2015 – 2016:
  Built a symbolic model of the TLS 1.3 specification under development at that time (draft 10)

- Goal: verify the core properties of TLS 1.3
  as an authenticated key exchange protocol
  - secrecy of session keys
  - unilateral (mutual) authentication

Reading the Specification (Draft of 160 pages)

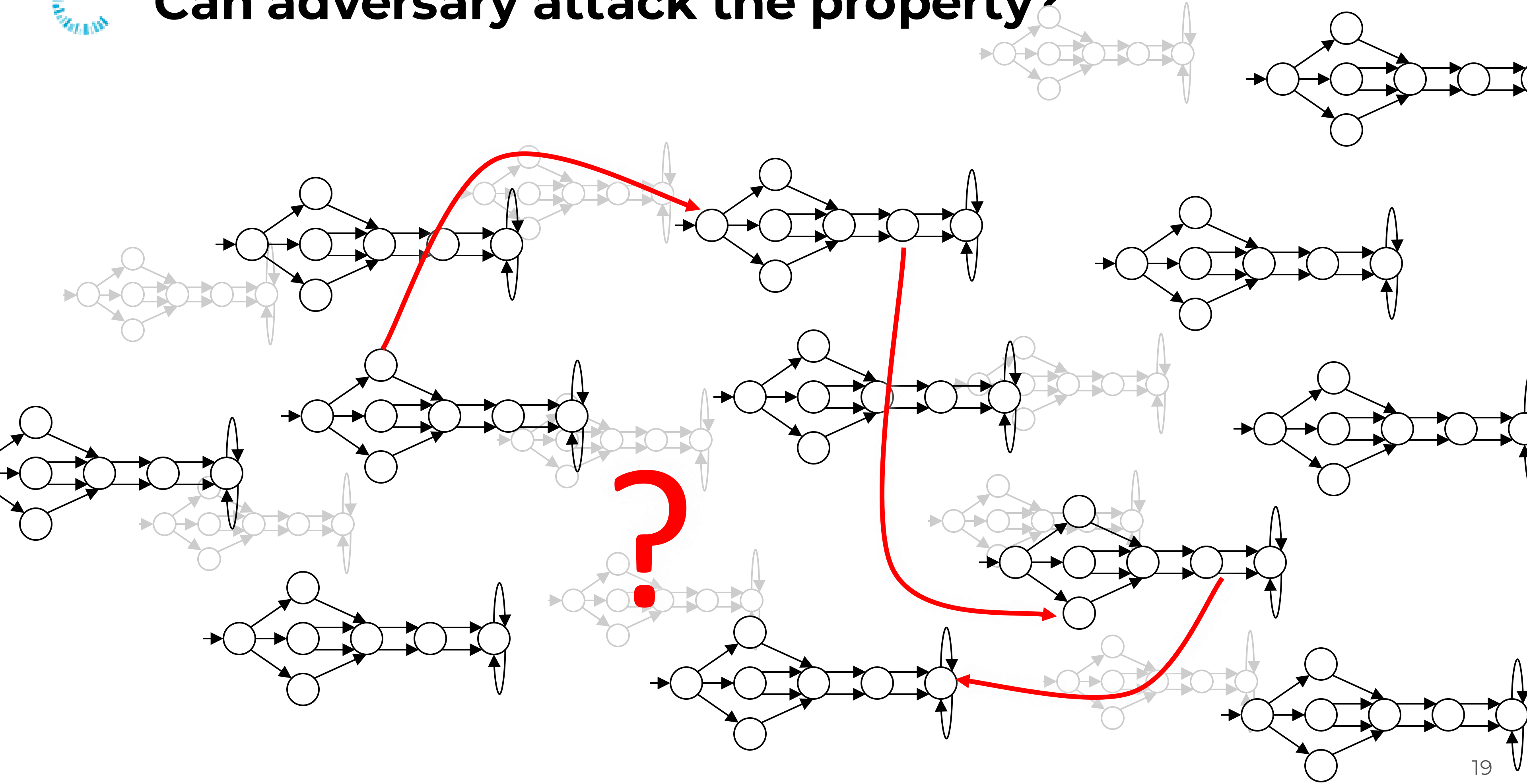
Rules model state machine



Example: client state machine

Rules correspond to edges

Can adversary attack the property?



Results!

We analysed Draft 10, Draft 10+, Draft 21

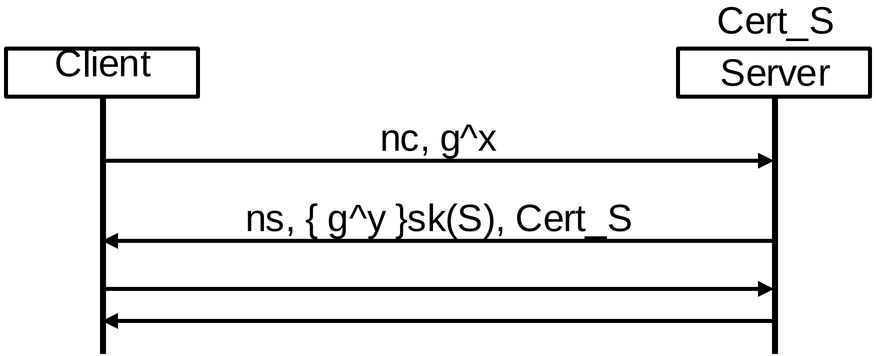
Proofs for all main properties on **Draft 10** [CHSM16] and **Draft 21** [CHHSM17] in the symbolic model

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

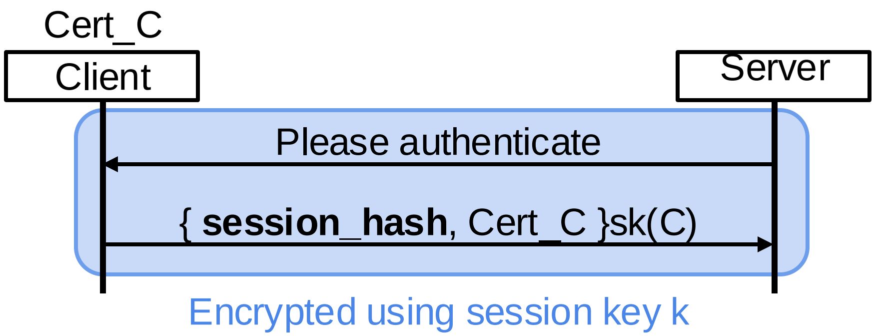
(unilateral, only mentioning relevant items)

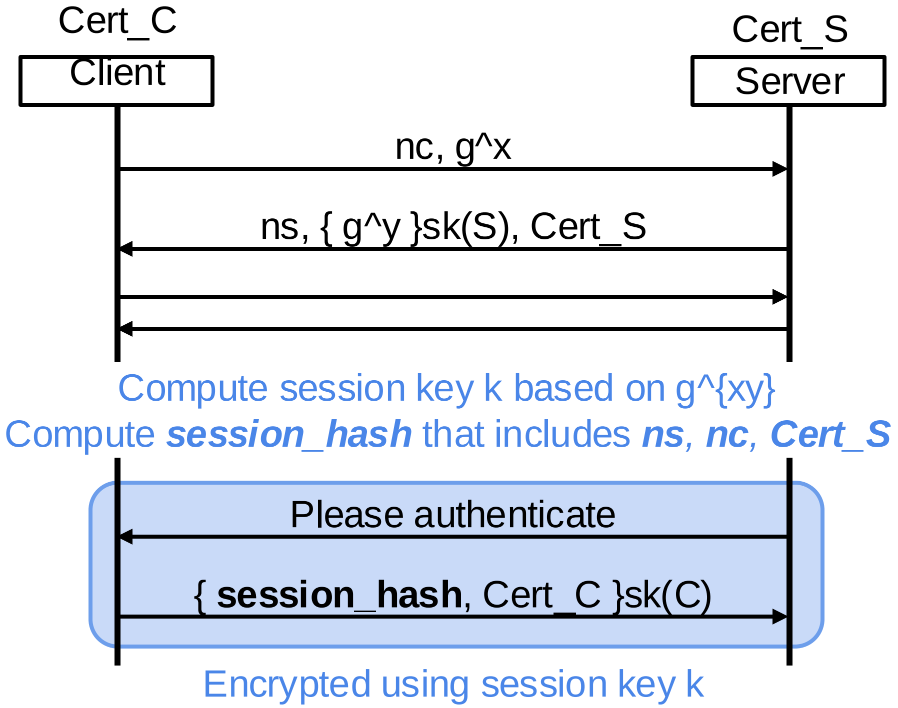


Compute session key k based on g^{xy}

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

Post-handshake Client authentication





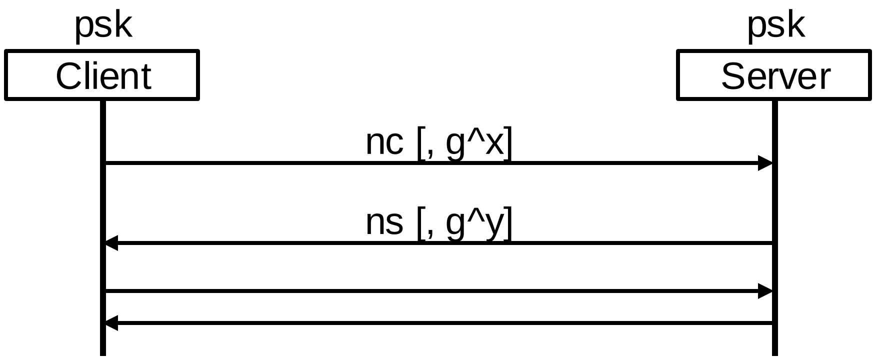
ECDH Handshake

(unilateral, only mentioning relevant items)

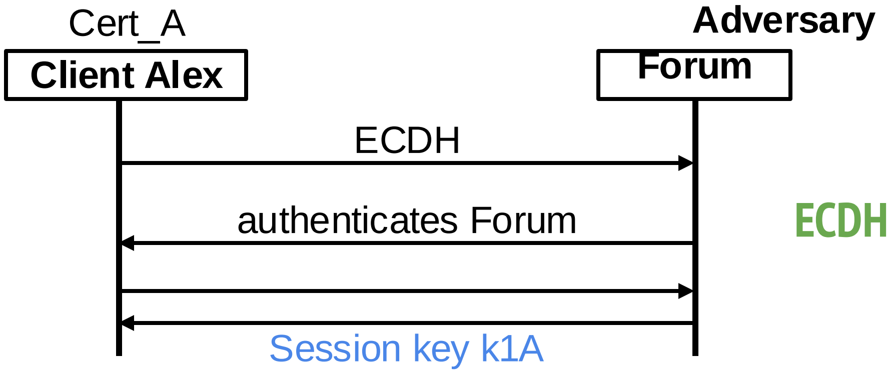
Post-handshake Client authentication

If the intended peers are honest, we now have a mutually authenticated channel: both parties are sure they are communicating with the right partner.

PSK [-DHE]

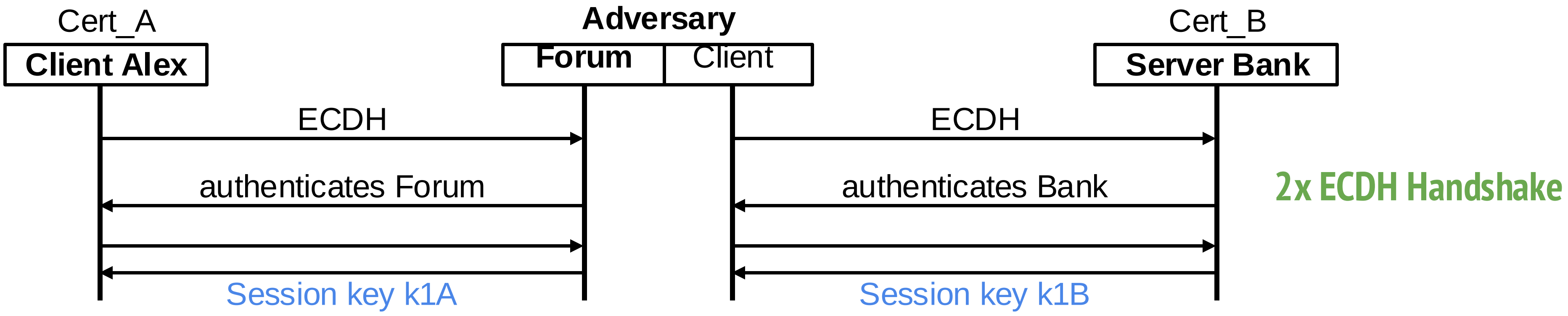


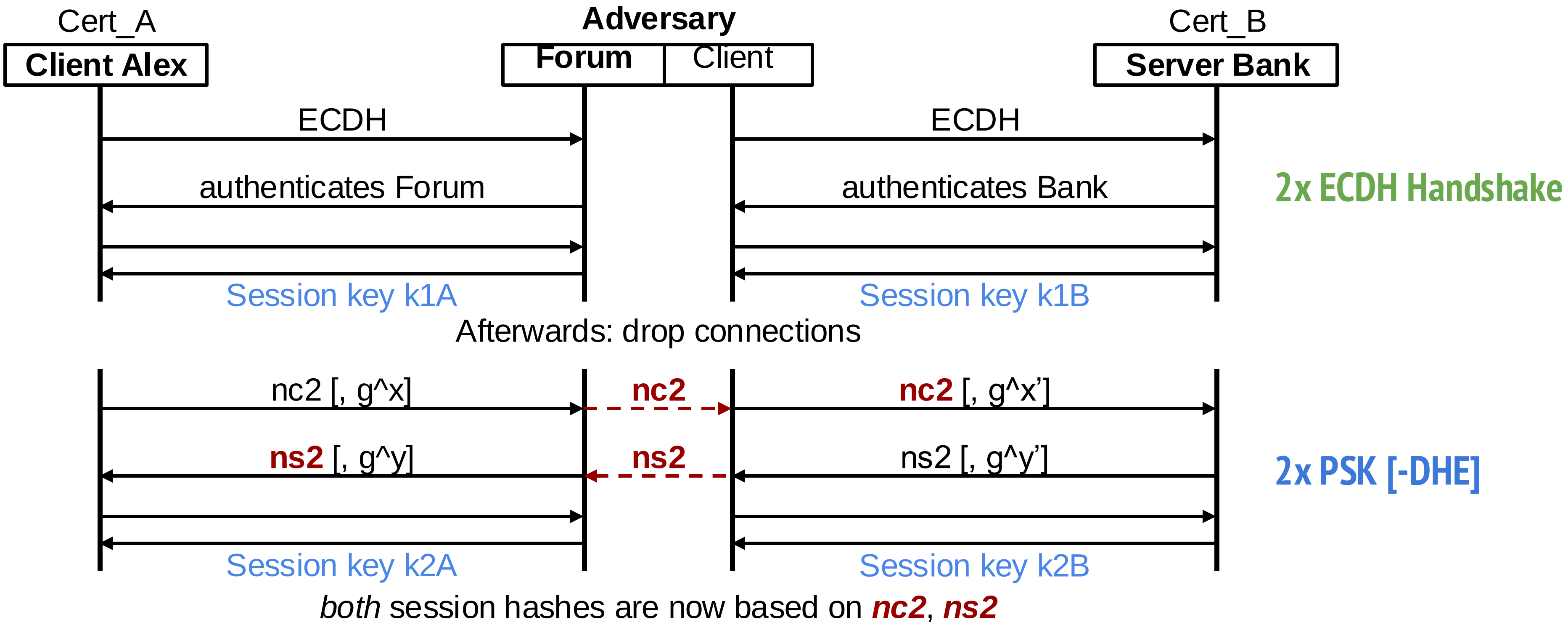
Compute new session key k based on psk [, g^{xy}]
Compute ***session_hash*** that includes ***ns***, ***nc***

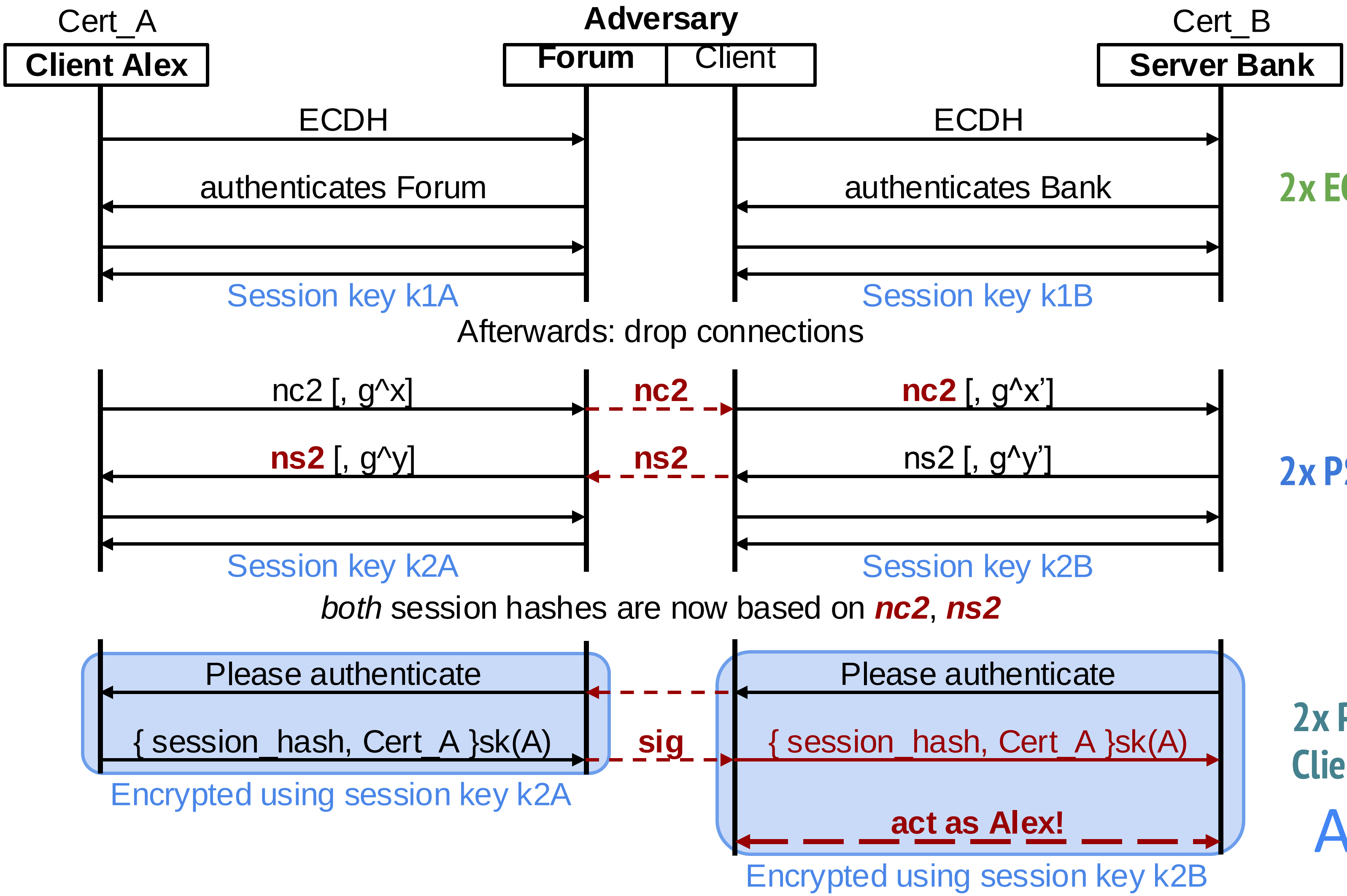


ECDH Handshake

Attack setup!


Cert_A
Client Alex
Adversary
Forum
Client
Cert_B
Server Bank
ECDH
authenticates Forum
Session key k1A
ECDH
authenticates Bank
Session key k1B
2x ECDH Handshake


Cert_A
Adversary
Cert_B
Client Alex
Forum
Client
Server Bank
ECDH
ECDH
authenticates Forum
authenticates Bank
2x ECDH Handshake
Session key k1A
Session key k1B
Afterwards: drop connections
nc2 [, g^x]
nc2
nc2 [, g^x']
ns2 [, g^y]
ns2
ns2 [, g^y']
2x PSK [-DHE]
Session key k2A
Session key k2B
both session hashes are now based on ***nc2***, ***ns2***



2x ECDH Handshake

2x PSK [-DHE]

2x Post-handshake Client authentication

Attack!

Cause and mitigation

- Prime example of an attack that can arise because of the interaction of modes
- No binding between the client signature and session for which it is intended
- Complicated to find
    - requires 18 messages to set up
    - involves 2 handshakes, 2 resumptions, 1 client auth...
- Communicated this to the IETF TLS Working Group...

Cause and mitigation

https://www.ietf.org/mail-archive/web/tls/current/msg18215.html

Dear all,

We [1] are in the process of performing an automated symbolic analysis of the TLS 1.3 specification draft (revision 10) using the Tamarin prover [2], which is a tool for automated security protocol analysis.

While revision 10 does not yet appear to permit certificate-based client authentication in PSK (and in particular resumption using PSK), we modelled what we believe is the intended functionality. By enabling client authentication either in the initial handshake, or with a post- handshake signature over the handshake hash, our Tamarin analysis finds an attack. The result is a complete breakage of client authentication, as the attacker can impersonate a client when communicating with a server:

Suppose a client Alice performs an initial handshake with Charlie. Charlie, masquerading as Alice, subsequently performs a handshake with Bob. Following a PSK resumption, Bob requests authentication from Charlie (impersonating Alice). Charlie then requests authentication from Alice, and the returned signature

IETF WG mailing list reactions

“Nice analysis! I think that the composition of different mechanisms in the protocol is likely to be where many subtle issues lie, and analyses like this one support that concern.”

IETF WG mailing list reactions

“Nice analysis! I think that the composition of different mechanisms in the protocol is likely to be where many subtle issues lie, and analyses like this one support that concern.”

“Thanks for posting this. It's great to see people doing real formal analysis of the TLS 1.3 draft; this is really helpful in guiding the design.”

IETF WG mailing list reactions

“Nice analysis! I think that the composition of different mechanisms in the protocol is likely to be where many subtle issues lie, and analyses like this one support that concern.”

“Thanks for posting this. It's great to see people doing real formal analysis of the TLS 1.3 draft; this is really helpful in guiding the design.”

“The result motivates and confirms the need to modify the handshake hashes to contain the server Finished when we add post-handshake authentication as is done in PR#316, which of course we'll be discussing in Yokohama.”

Impact

Raised awareness:

- subtle bugs
- complex to find for humans

Benefits of methodology:

- Provide quicker analysis for proposed designs
- Complements other analysis approaches

The finer details

Analysis Process for TLS 1.3

Establish session keys

Are the keys secret?

secrecy

protocol spec

properties

model

encode properties

proofs?

Analysis Process for TLS 1.3

Establish session keys

Are the keys secret?

secrecy

protocol spec

properties

model

encode properties

proofs?

STEP 1

Analysis Process for TLS 1.3

Establish session keys

Are the keys secret?

secrecy

protocol spec

properties

model

encode properties

proofs?

STEP 1

STEP 2

Analysis Process for TLS 1.3

Establish session keys

Are the keys secret?

secrecy

protocol spec

properties

model

encode properties

proofs?

STEP 1

STEP 2

STEP 3

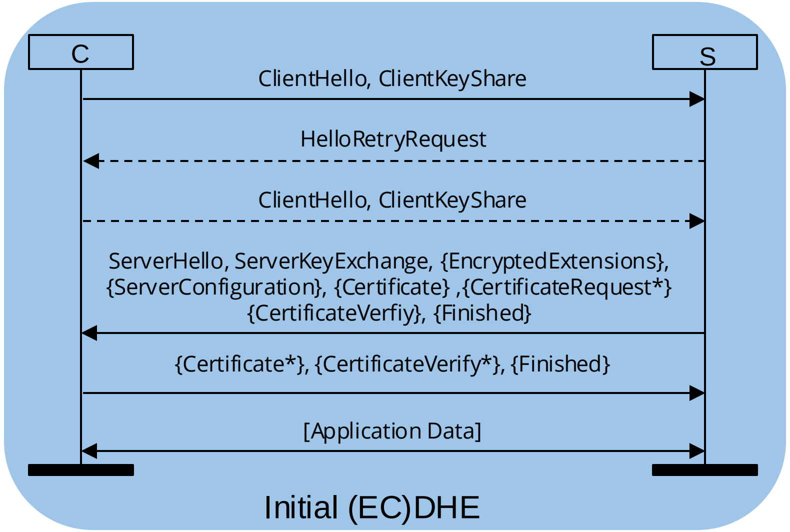
Step 1: Building the Model

- 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

Step 1: Building the Model

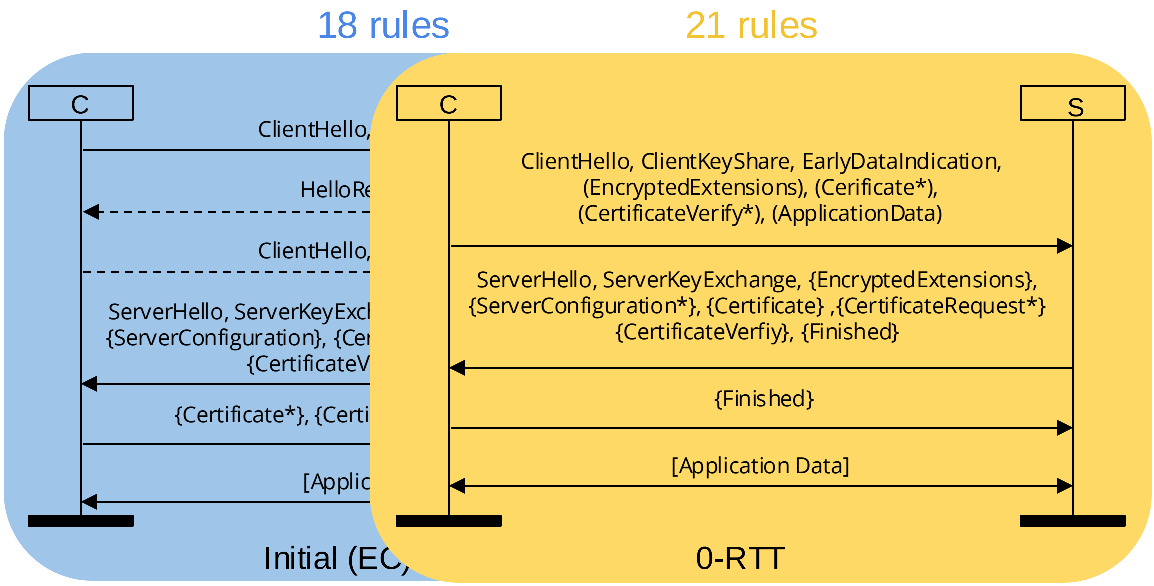
- Encode honest party and adversary actions as Tamarin rules
- Honest client and server rules correspond to flights of messages
- Rules transition protocol from one state to the next

18 rules



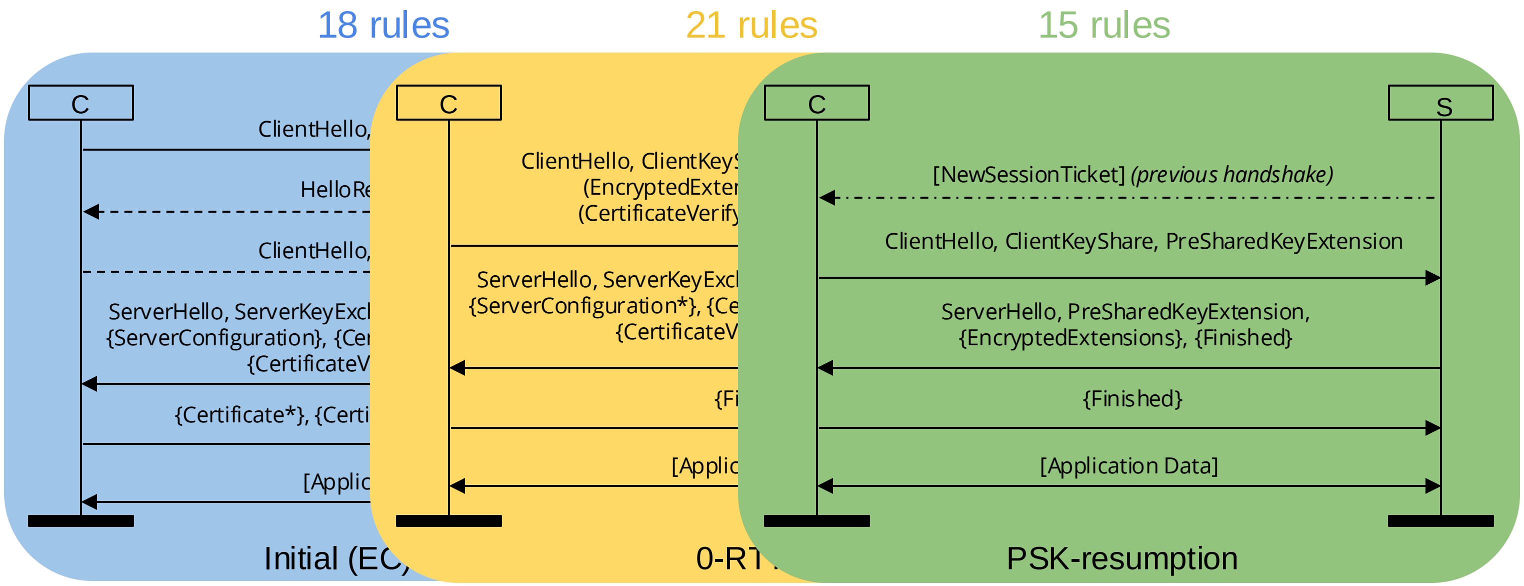
Step 1: Building the Model

- 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



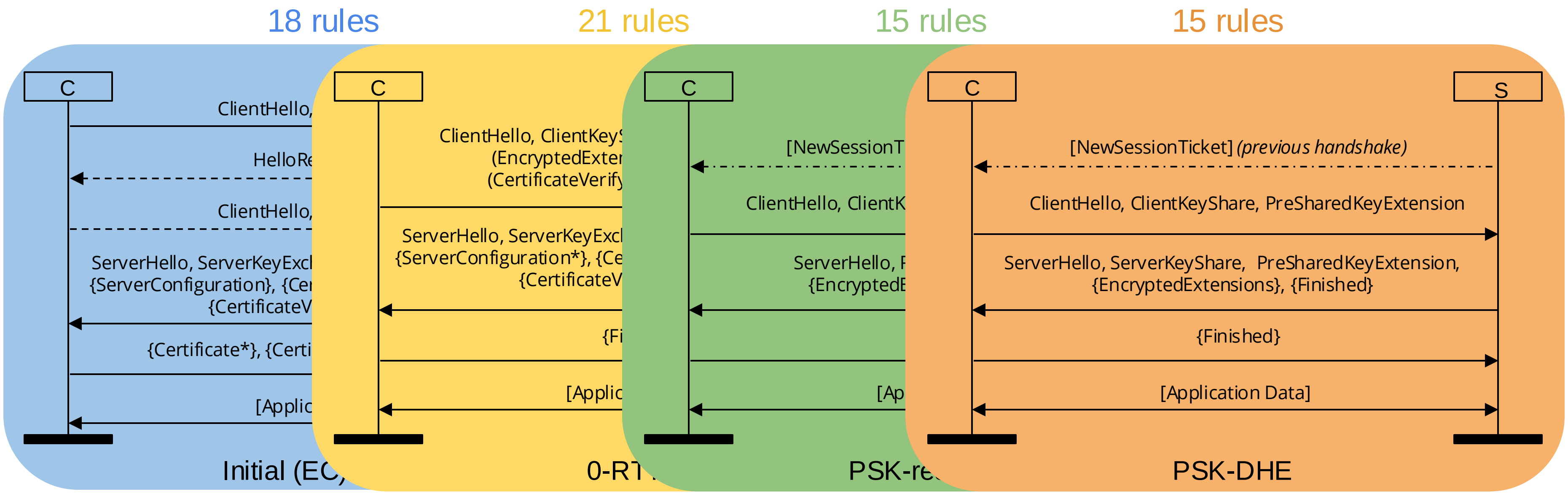
Step 1: Building the Model

- 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



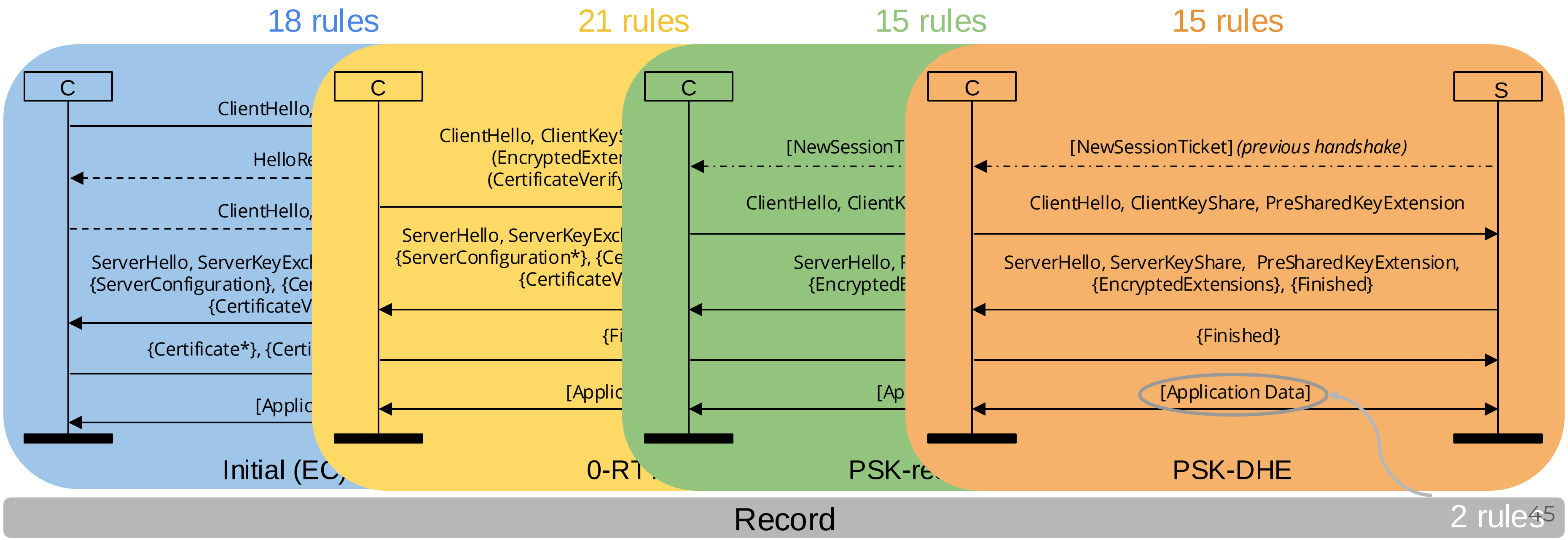
Step 1: Building the Model

- 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>)
    ]
```

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: Building the Model

- Modelling a complex protocol in Tamarin is not a simple exercise!
- Large number of rules and macros...

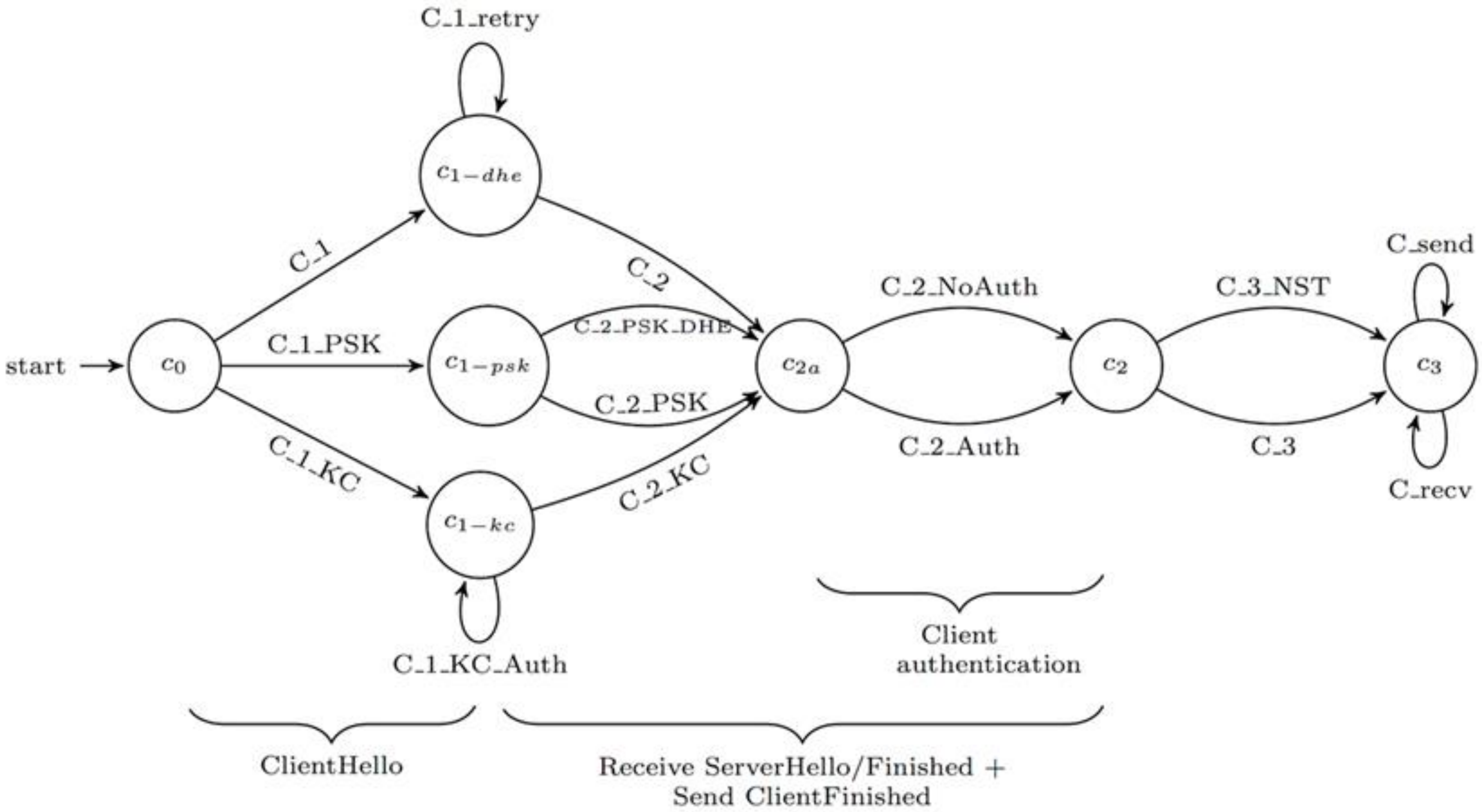
```
define(<!L!>,<!'256'!>)dnl
dnl  Definitions of key derivations
dnl  In each case, the local method should define hs_hash/session_hash
dnl  and also ss and/or es
define(<!xSS!>,<!HKDF('0',ss,'extractedSS',L)!>)dnl
define(<!xES!>,<!HKDF('0',es,'extractedES',L)!>)dnl
define(<!MS!>,<!HKDF(xSS,xES,'master_secret',L)!>)dnl
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
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 h
dnl
dnl Additional messages sent in S1
define(<!ServerConfiguration!>,<!Y!>)dnl
define(<!ServerEncryptedExtensions!>,<!$exts!>)
define(<!ServerCertificate!>,<!pk(~ltkS)!>) dnl
define(<!ServerCertificateVerify!>,<!s_signatur
define(<!CertificateRequest!>,<!$cert_req!>)dnl
```

Key computations

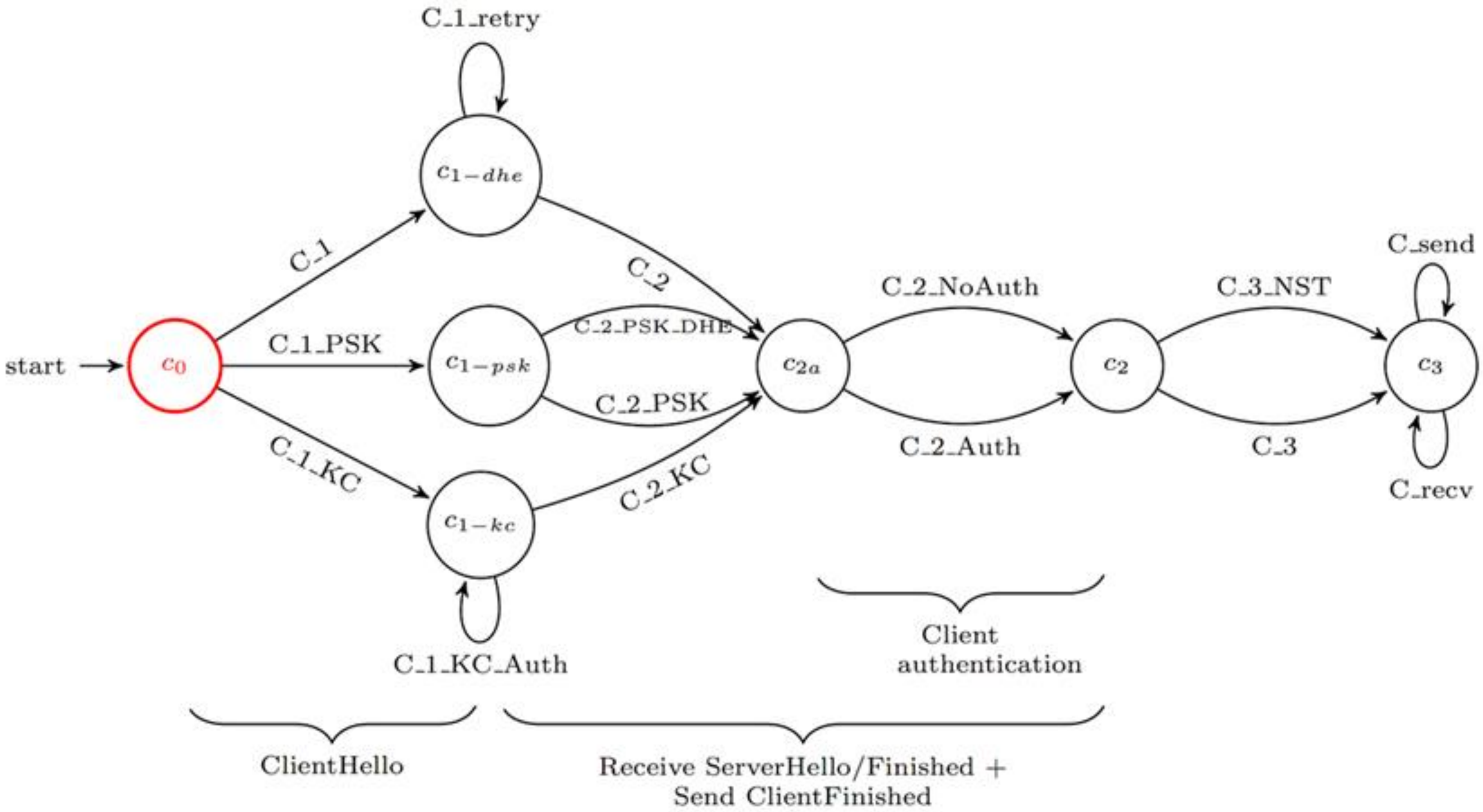
Macros for just 3 of our rules!

Note:
This notation is obsolete.
Tamarin a much more readable macro system now.

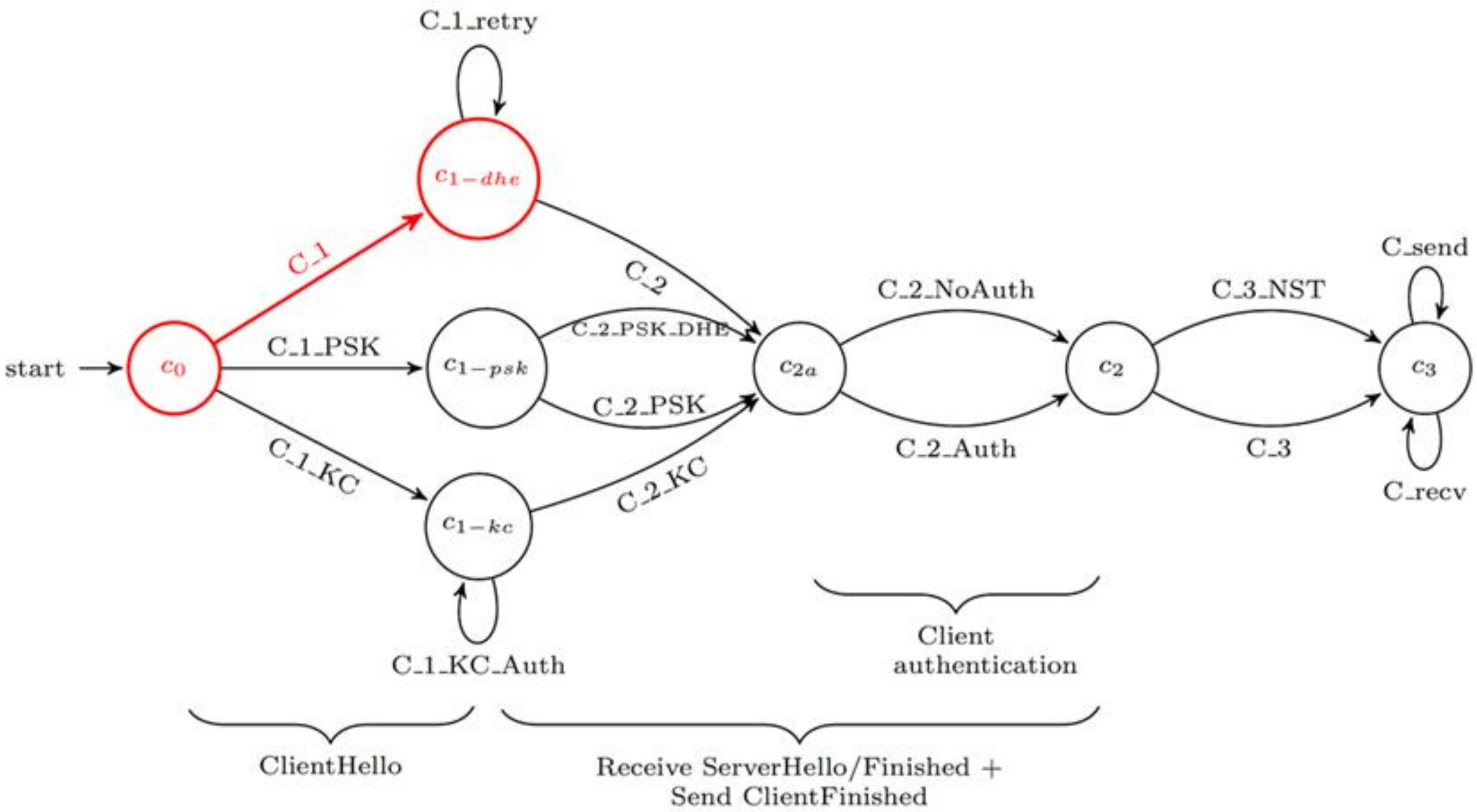
Step 1: Building the Model



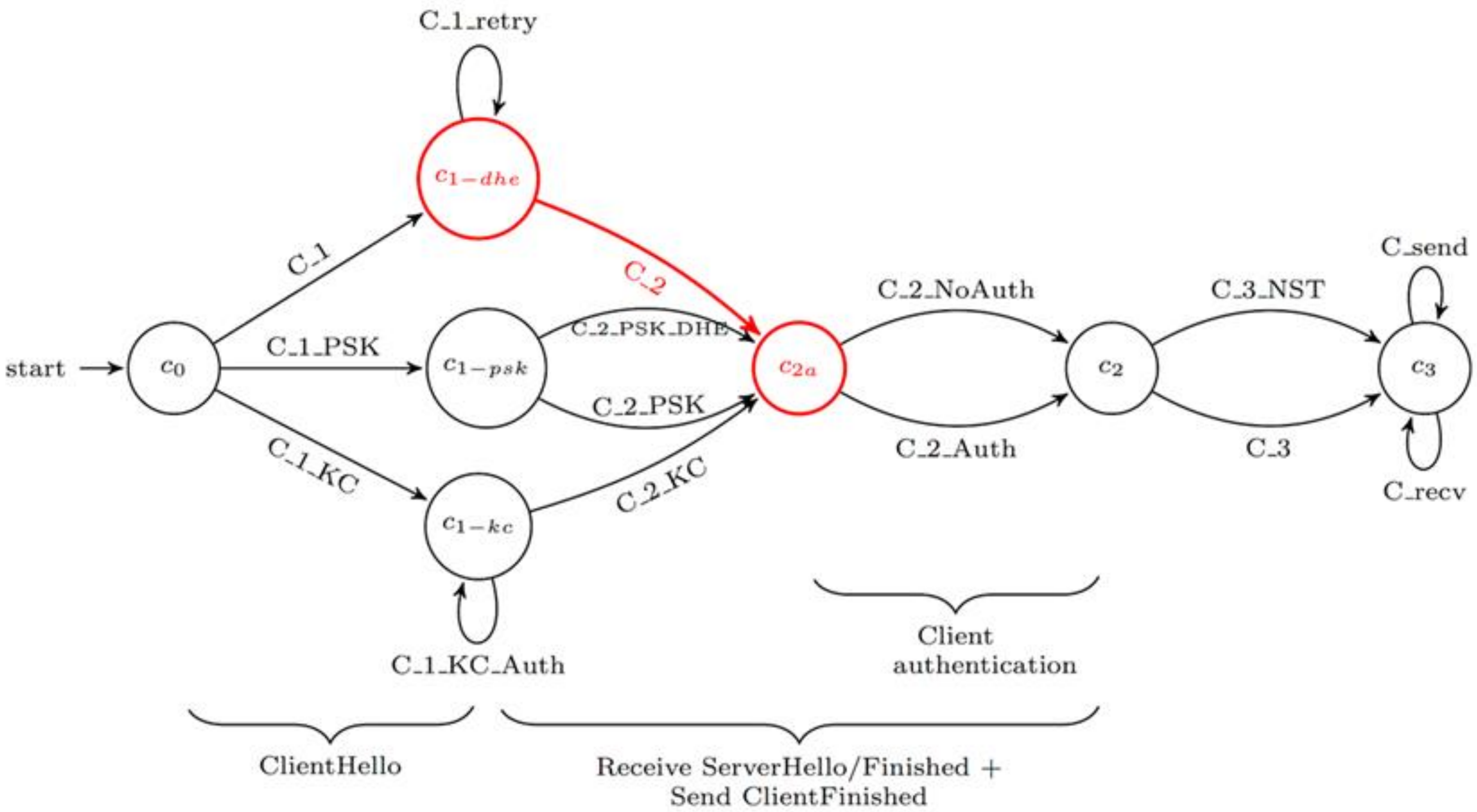
Step 1: Building the Model



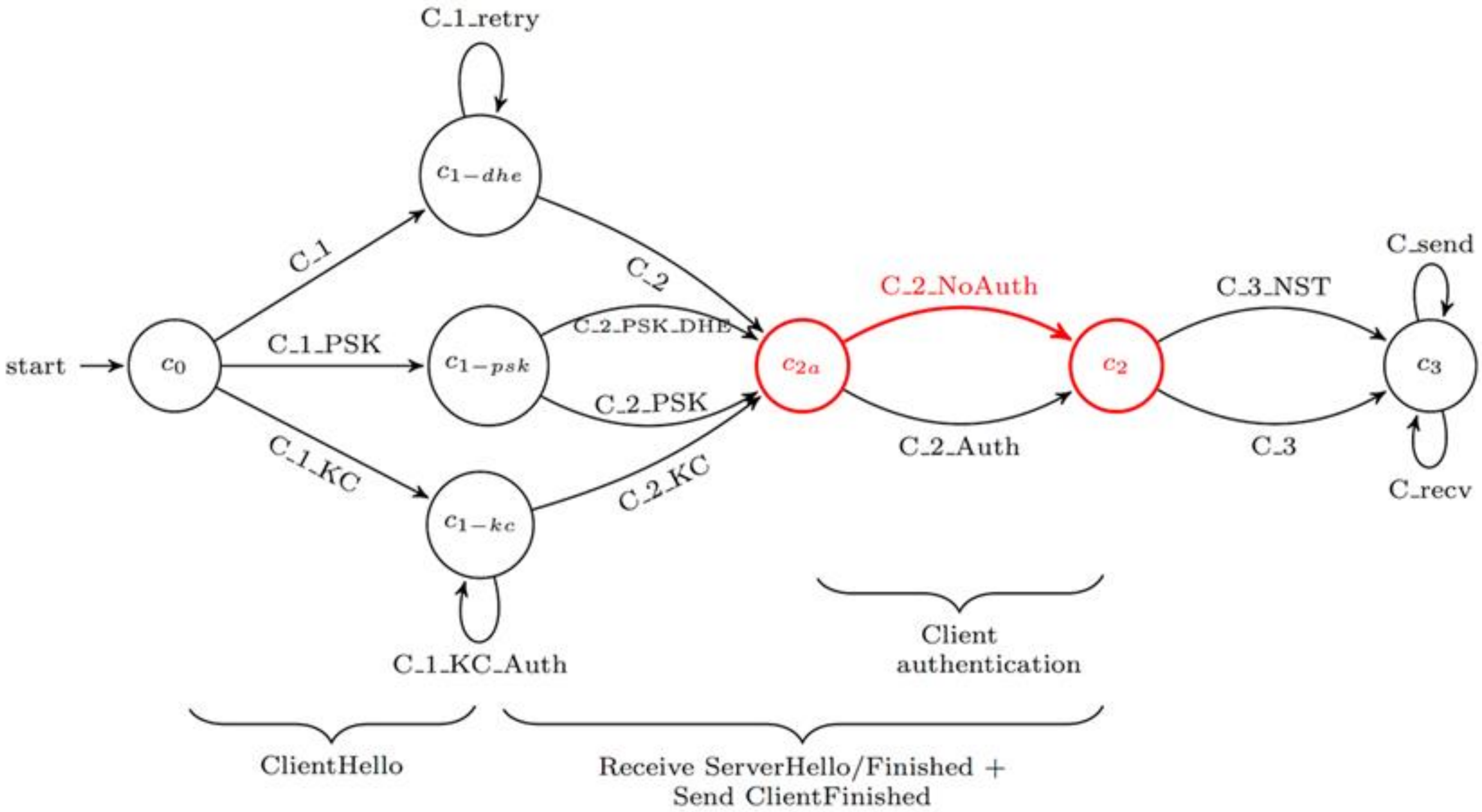
Step 1: Building the Model



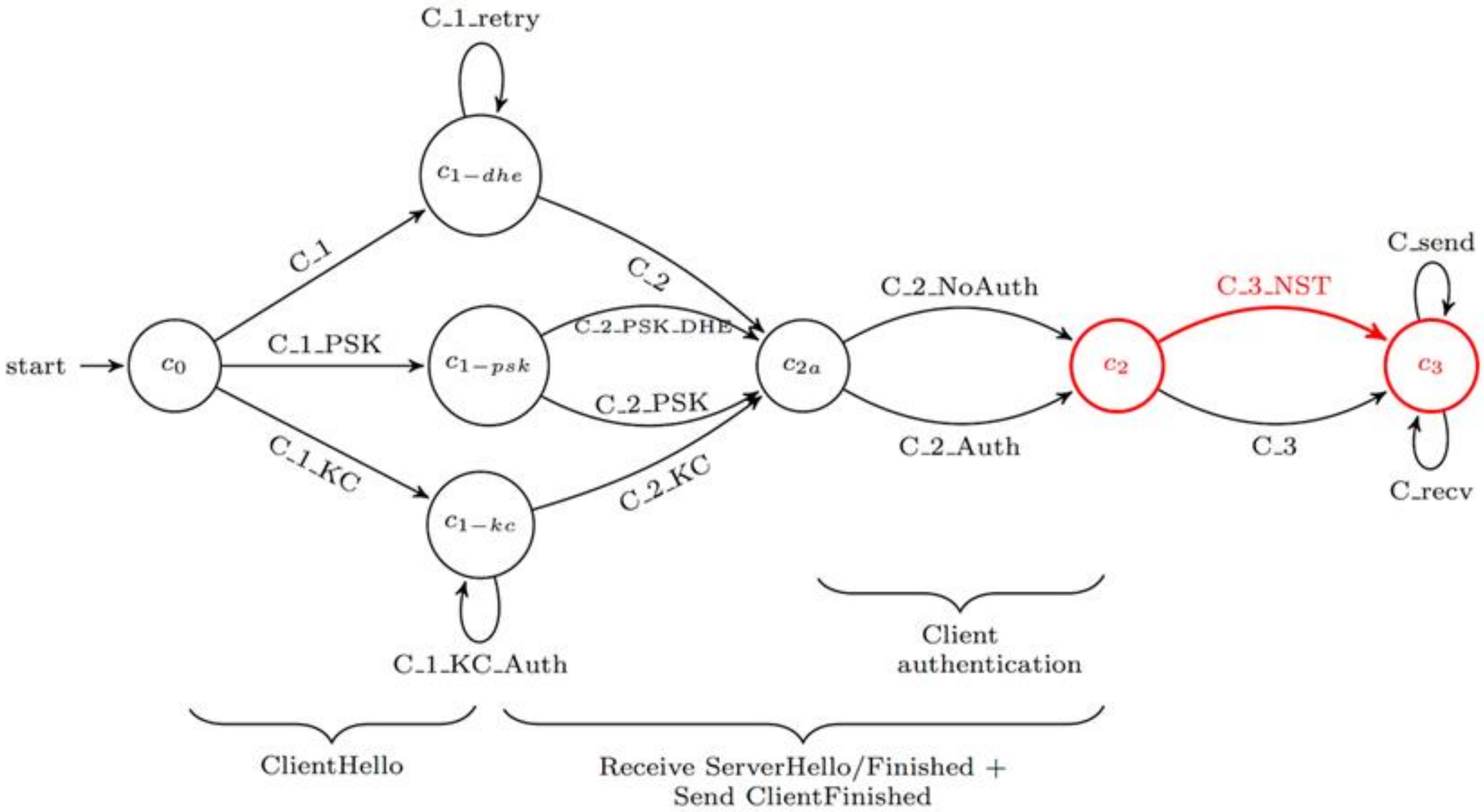
Step 1: Building the Model



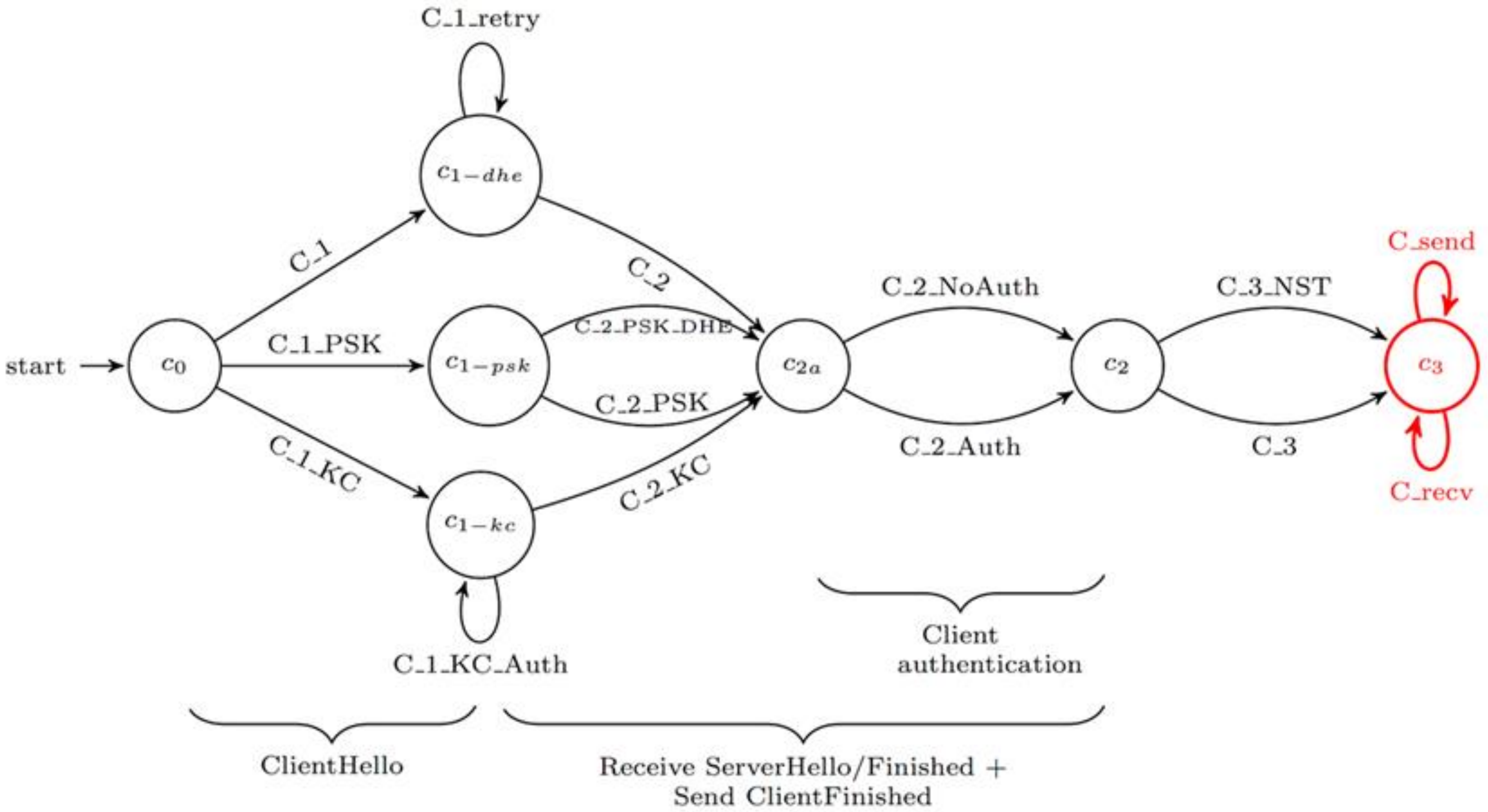
Step 1: Building the Model



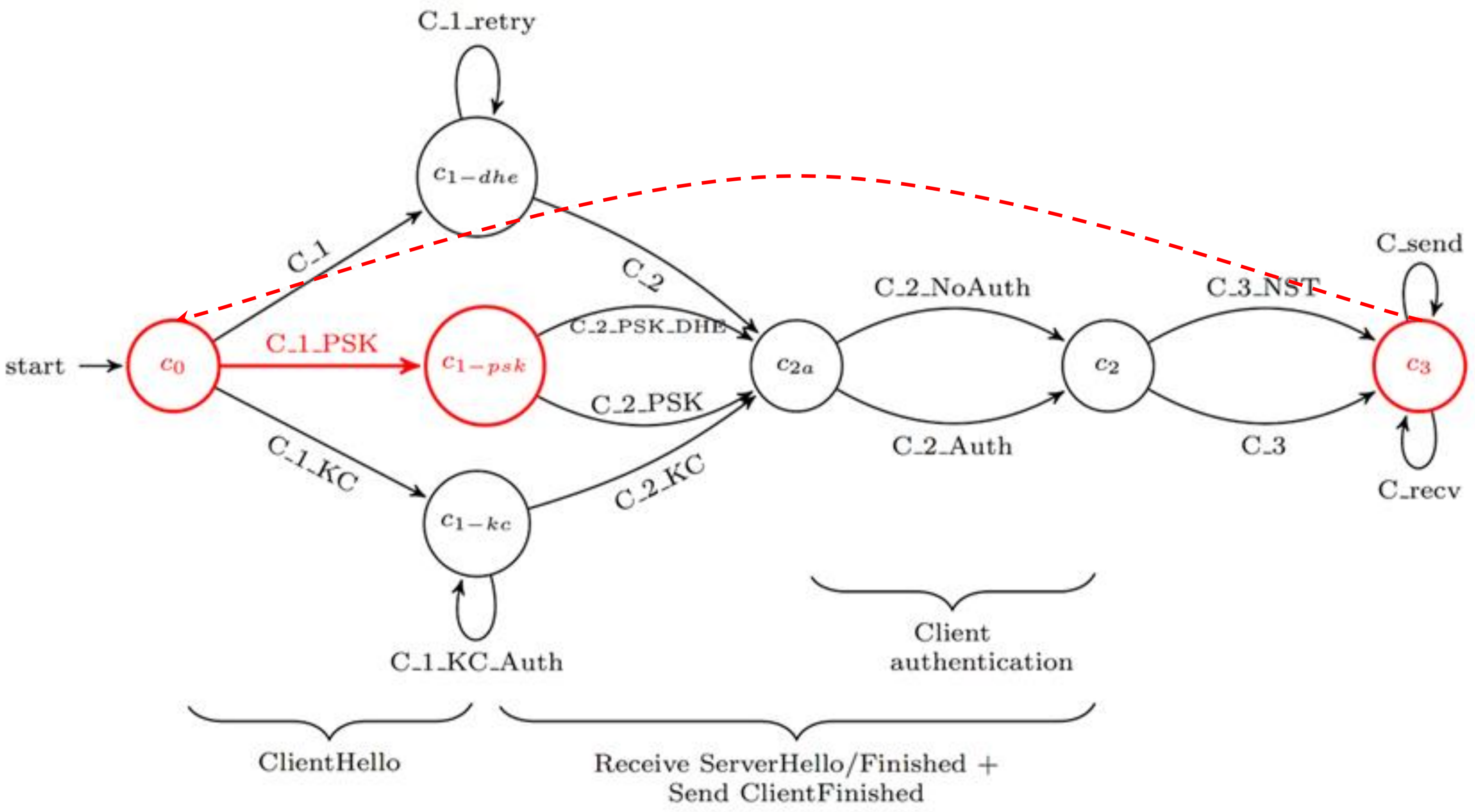
Step 1: Building the Model



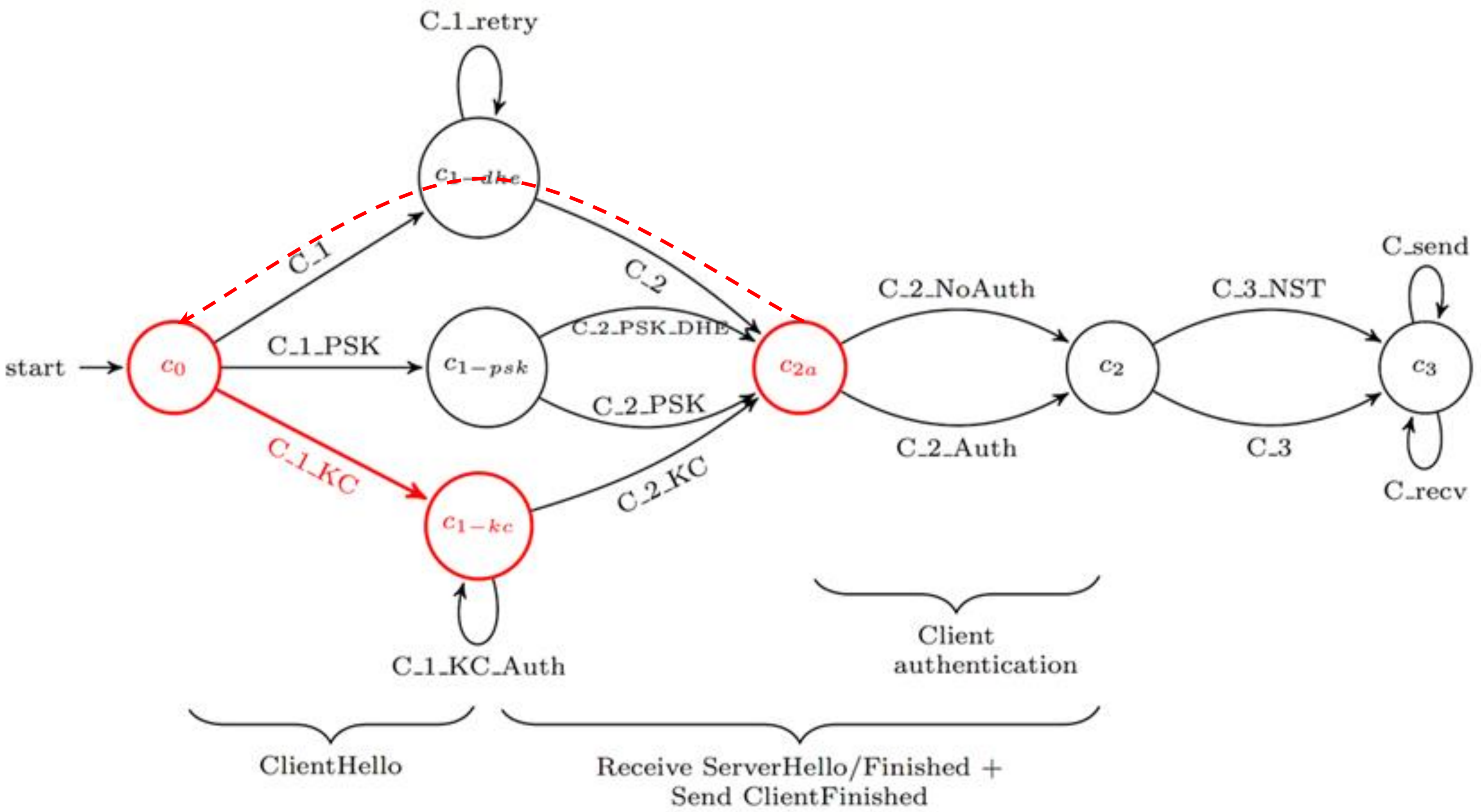
Step 1: Building the Model



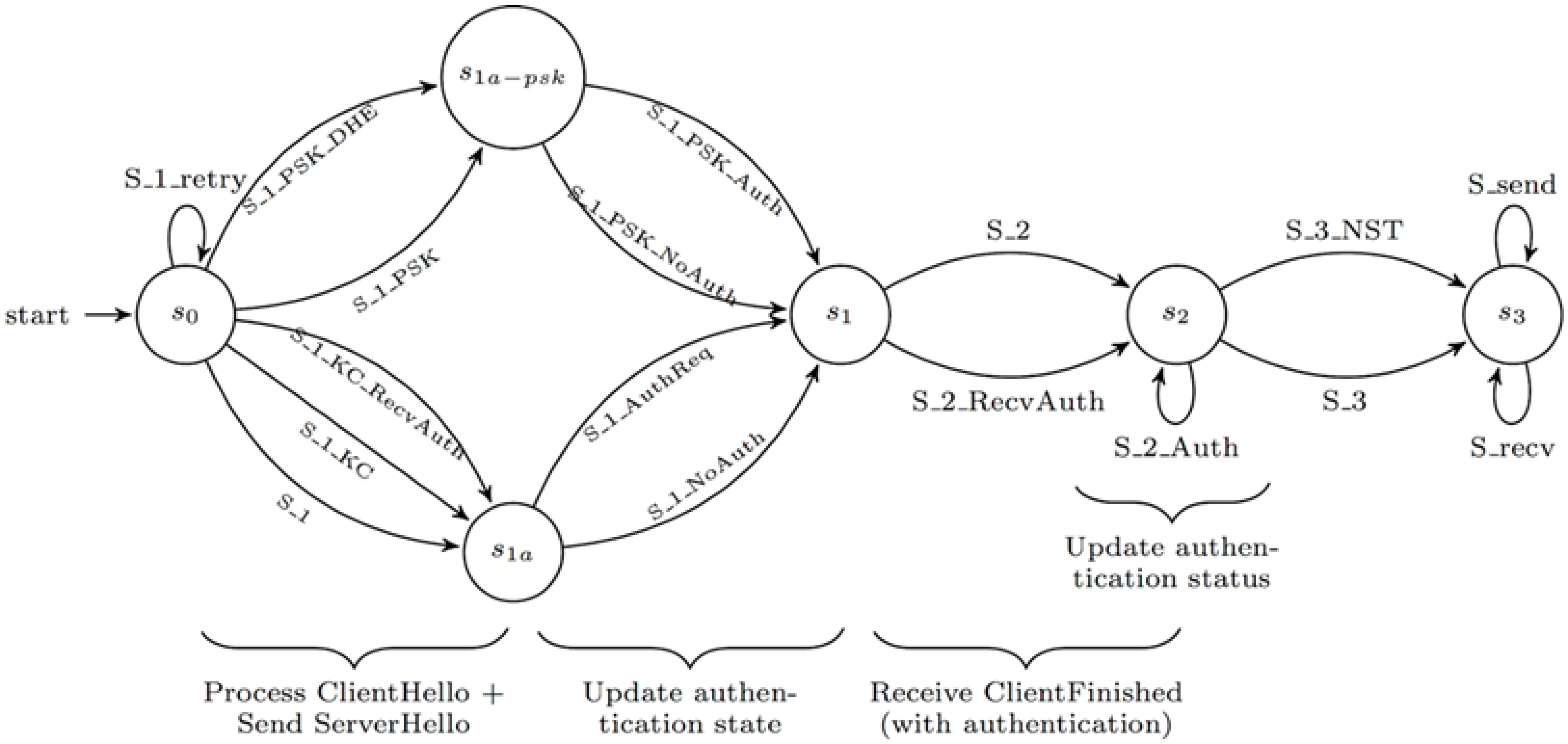
Step 1: Building the Model



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))
  (1)==> not Ex #j. K(k)@j
```

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))
  (1)==> 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))
  (1)==> 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)
Mutual authentication
Confidentiality of ephemeral secret
Confidentiality of static secret
Perfect forward secrecy
Integrity of handshake messages

```
entity_authentication
mutual_entity_authentication
```

```
secret_early_data_keys
secret_session_keys(with PFS)
```

```
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…

Running Tamarin 1.4.0 Index Download Actions » Options »

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) ∧

Running TAMARIN 1.4.0 Index Download Actions » Options »

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

SessionKey(actor, peer, role, <k, 'authenticated'>) @ #i   !KU(k) @ #j

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) ∧

Running Tamarin 1.4.0 | Index | Download | Actions » | Options »

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
```

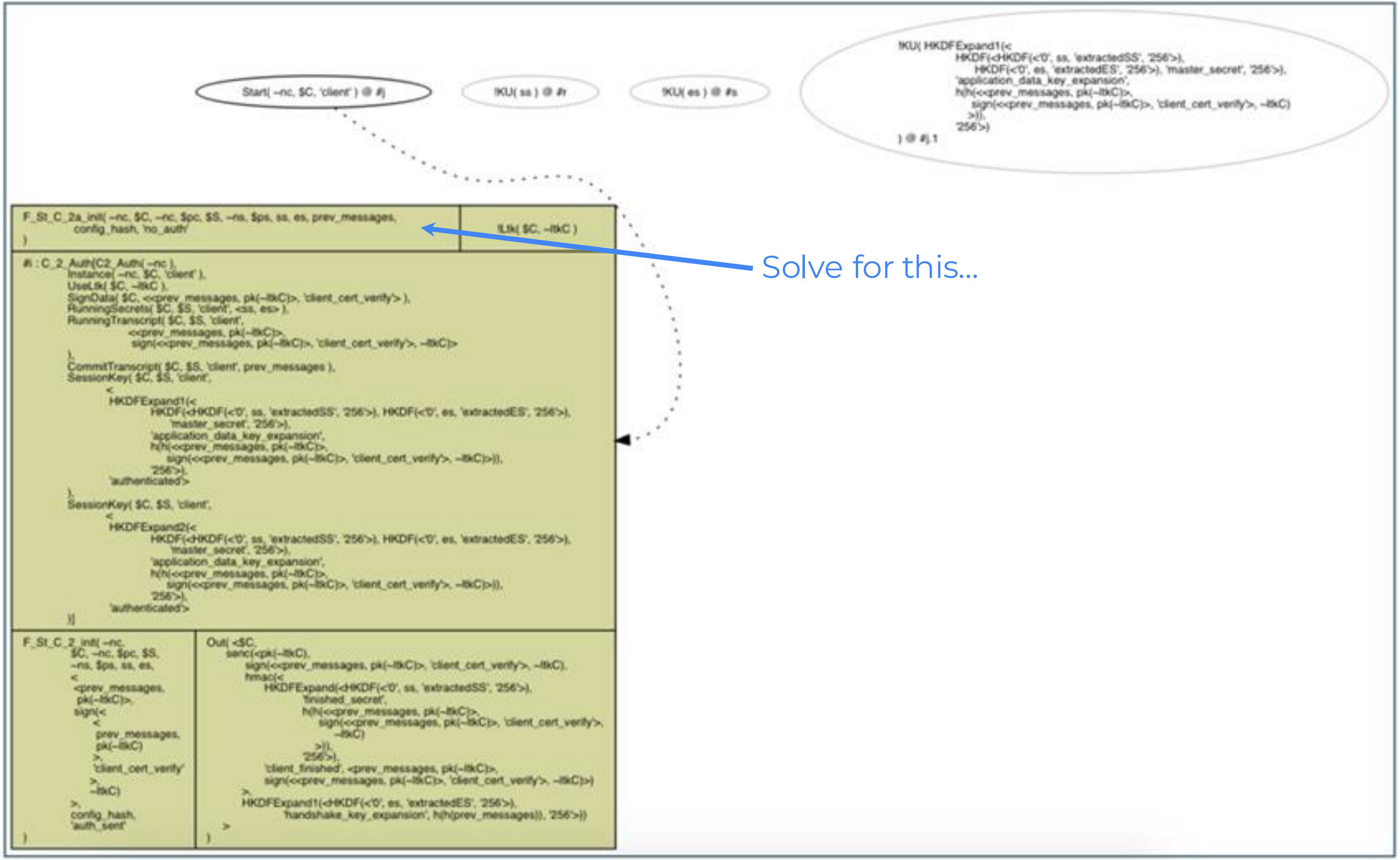
Visualization display

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

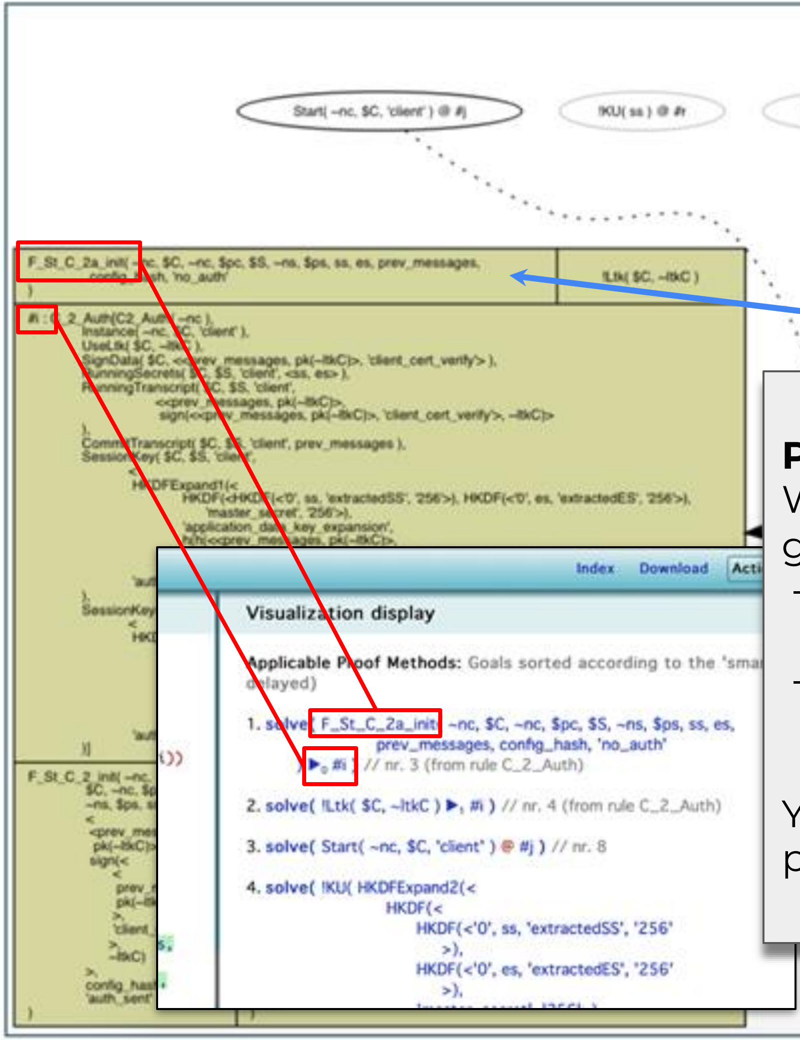
```
1. solve( F_St_C_2a_init( ~nc, $C, ~nc, $pc, $S, ~ns, $ps, ss, es,
                          prev_messages, config_hash, 'no_auth'
        ) ▶₀ #i ) // nr. 3 (from rule C_2_Auth)

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)
#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'>)) >)
Solve for this...



Solve for this...

Pro tip:
When trying to solve for a specific open goal in your graph, remember:
- The timepoint variable
  Here: **#i**
- The fact name (if multiple unsolved)
  Here: **F_St_C_2a_init**

You can use these to easily find the right proof method to apply.

Running Tamarin 1.4.0 Index Download Actions » Options »

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

Running Tamarin 1.4.0 — Index — Download — Actions » — Options »

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

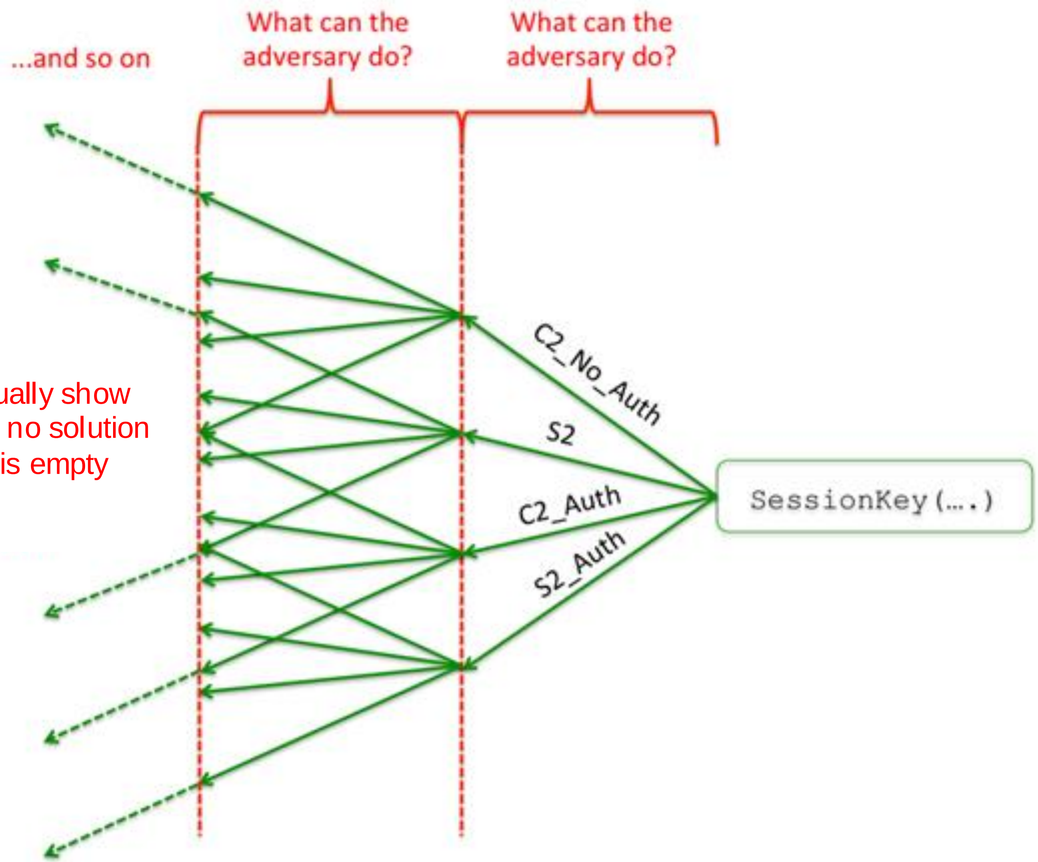
formulas:
 ∃ actor peer k #i.
  (EarlyDataKey(actor, peer, 'client', k) @ #i)
 ∧
  (∀ #r. (RevLtk(peer) @ #r) ⇒ ⊥) ∧ (∃ #j. (!KU(k) @ #j))

equations:
 subst:
 conj:

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

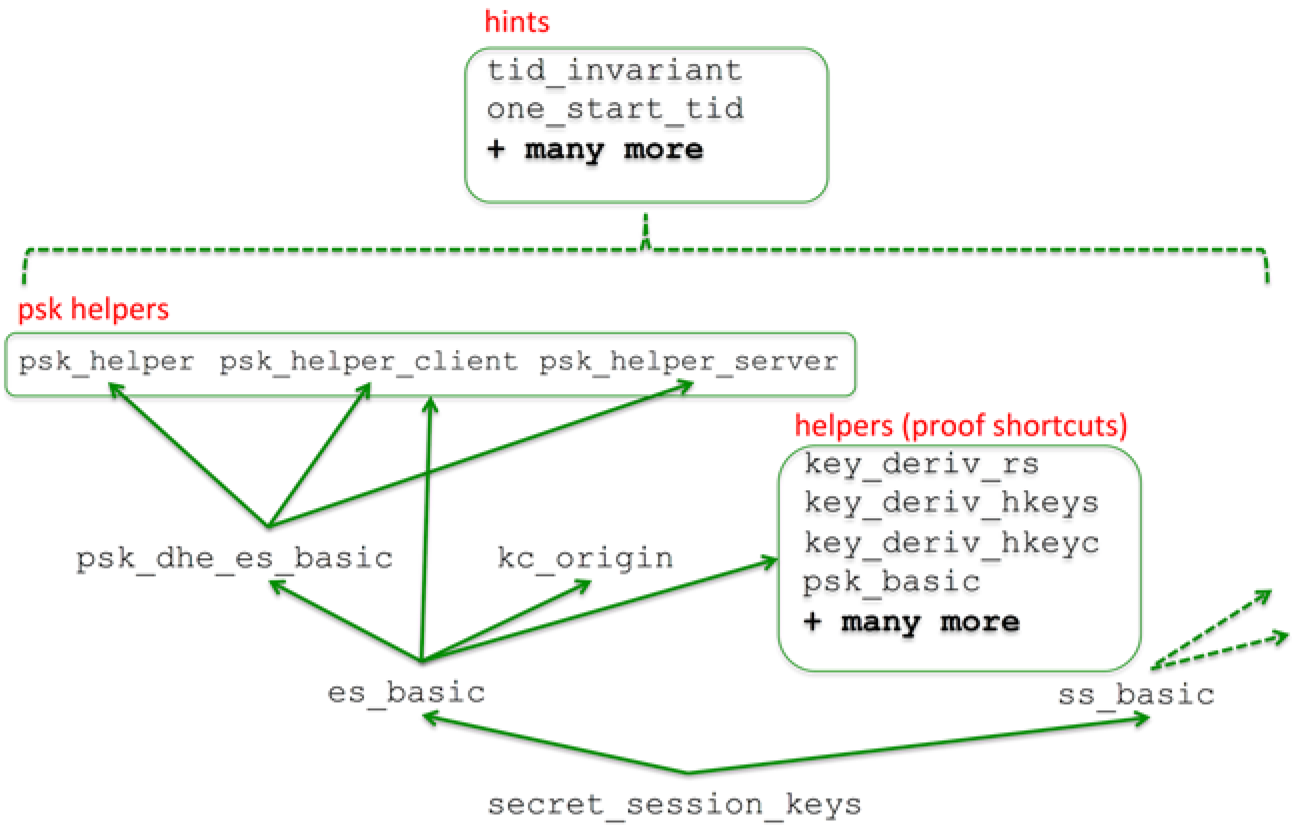
 ∀ actor actor2 psk_id psk_id2 peer peer2 rs auth_status
  auth_status2 #i #j.
  (UsePSK(actor, psk_id, peer, rs, 'client', auth_status
) @ #i) ∧
  (UsePSK(actor2, psk_id2, peer2, rs, 'server', auth_status2
) @ #i)

Step 3: Producing Proofs

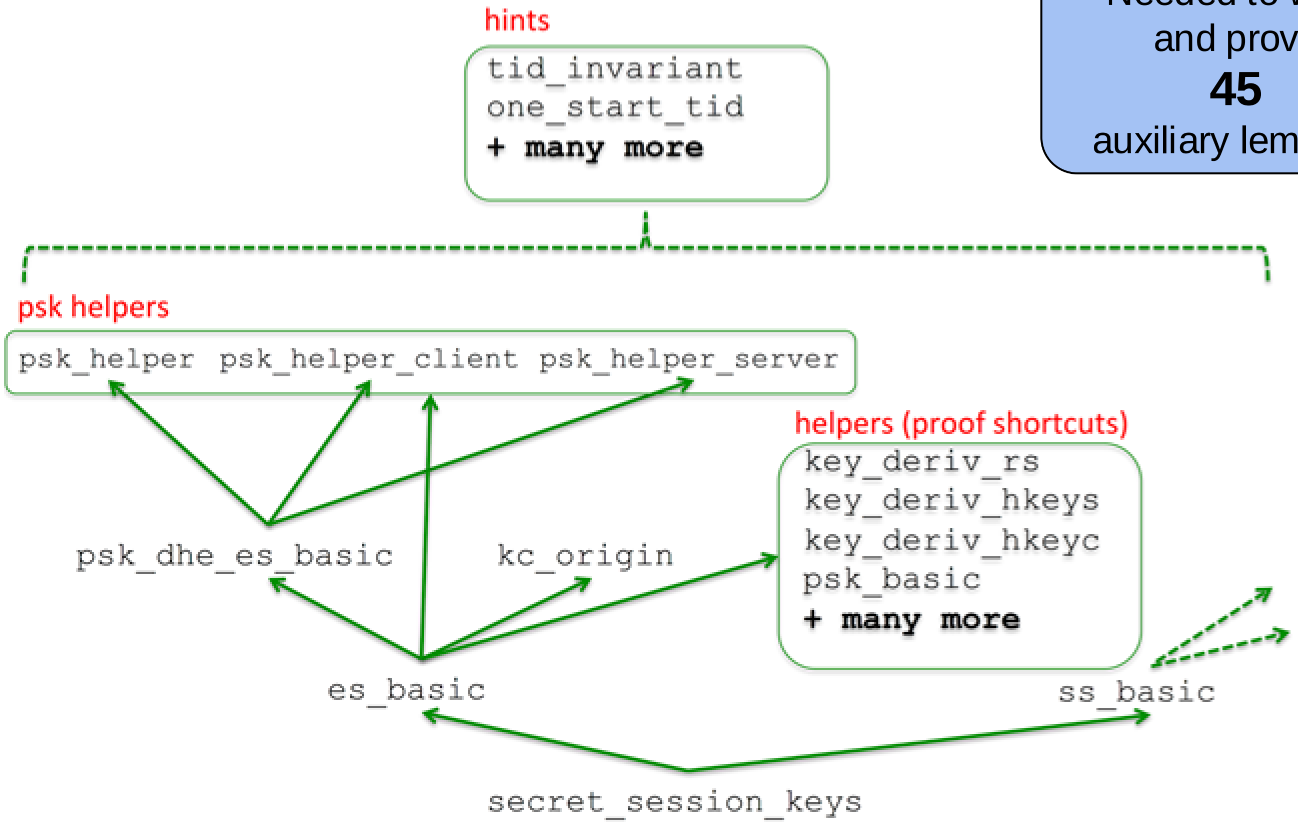


Will eventually show that there is no solution - the set is empty

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
  "∀
'auth
∨
#i)))
```

```
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

Step 3: Producing Proofs

```
lemma secret_session_keys:
  all-traces
  "∀
'auth
∨
#i)))
```

```
lemma entity_authentication [reuse]:
  all-traces
  "∀
  ) @ #
  nonce
```

```
lemma transcript_agreement [reuse]:
  all-traces
  "∀ actor peer transcript #i.
         ((CommitTranscript( actor, peer, 'client',
transcript
          ) @ #i) ∧
          (¬(∃ #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
  "∀
'auth
      ) @ #
∨
#i)))
      nonce
        transc
```

```
lemma entity_authentication [reuse]:
  all-traces
  "∀
```

```
lemma transcript_agreement [reuse]:
  all-traces
  "∀ actor peer transcript #i.
        ((CommitTranscript( actor, peer, 'client',
transcript
        ) @ #i) ∧
```

```
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))"
```

Draft 10 + next mechanism

Finding An Attack

- You've seen the message flows of the attack
- BUT how did we find it?!

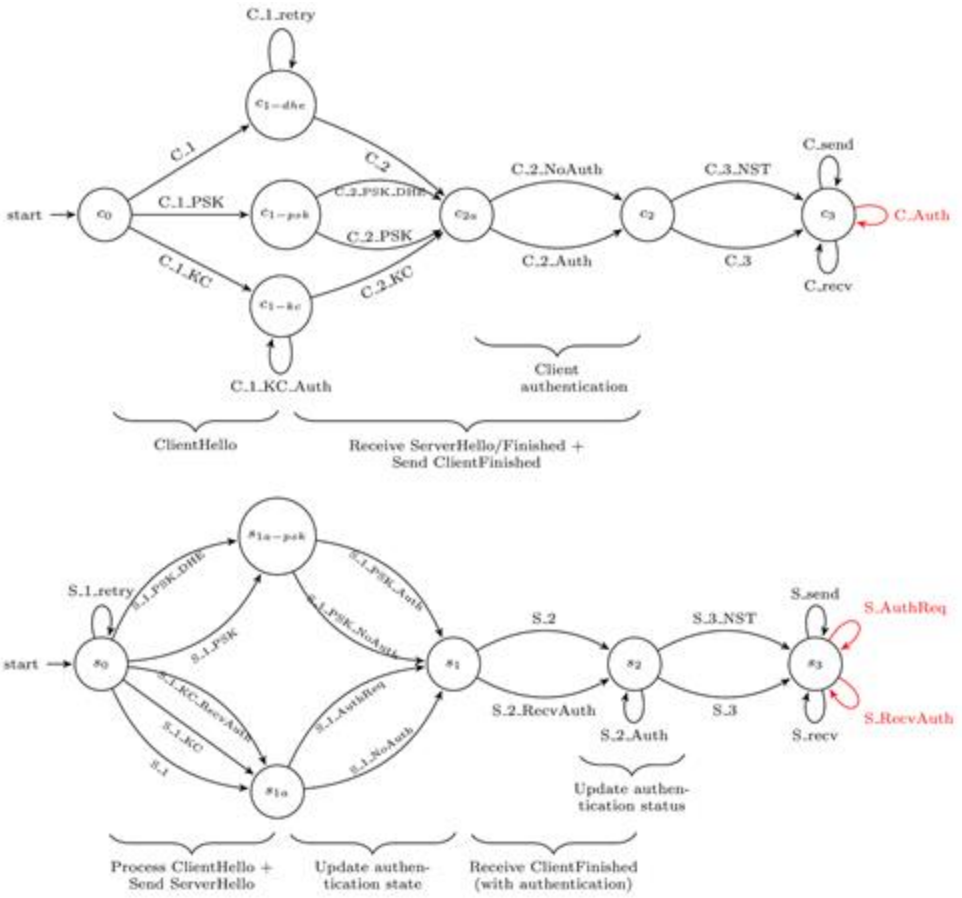
2x ECDH Handshake

2x PSK [-DHE]

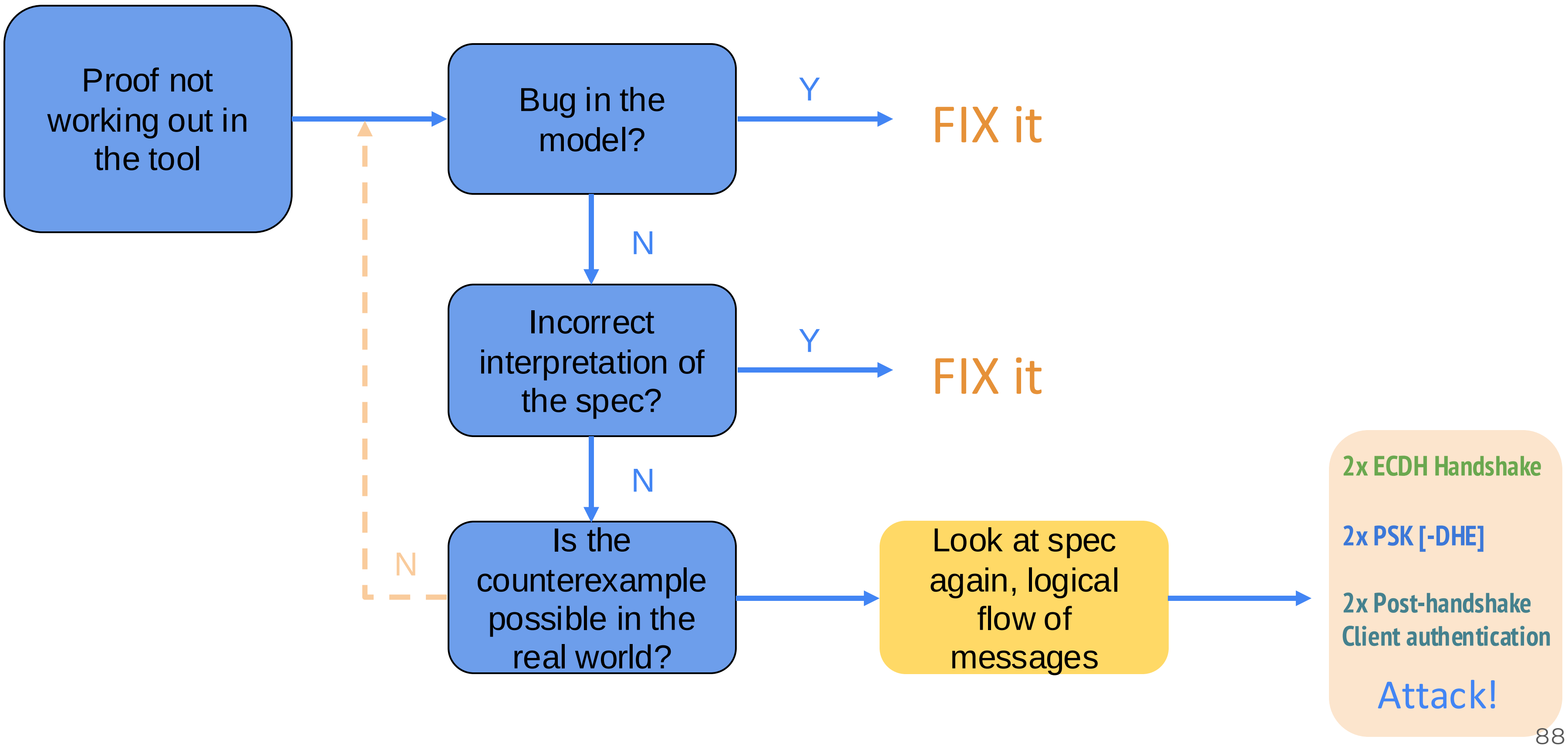
2x Post-handshake Client authentication

Attack!

Finding An Attack



Finding An Attack



Draft 21

Step 1: Building the Model

- TLS 1.3 has been a rapidly moving target
- Draft 21 - a completely new protocol!

- We modelled draft 21 in a far more granular fashion than draft 18
    - higher transparency - good for us, also good for everyone else!

https://samscott89.github.io/TLS13_Tamarin/

TLS 1.3 Protocol Overview

---snip---

TLS supports three basic key exchange modes:

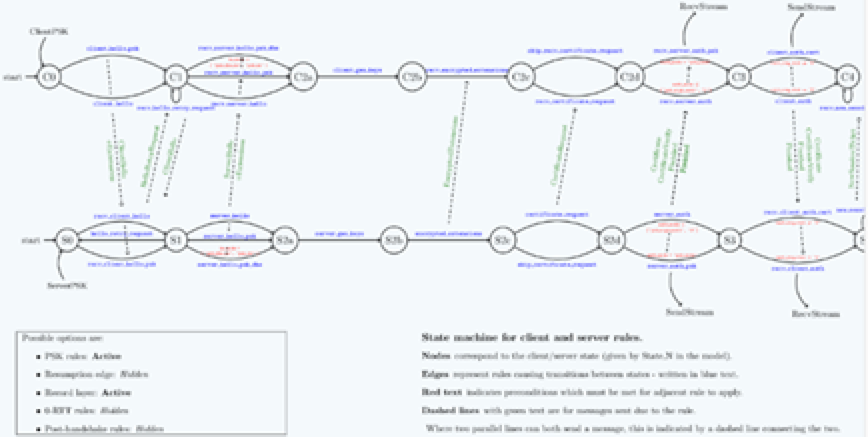
- (EC)DHE (Diffie-Hellman both the finite field and elliptic curve varieties),
- PSK-only, and
- PSK with (EC)DHE

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}                -------->
       [Application Data]        <------->     [Application Data]

               +  Indicates noteworthy extensions sent in the
                  previously noted message.
```

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:

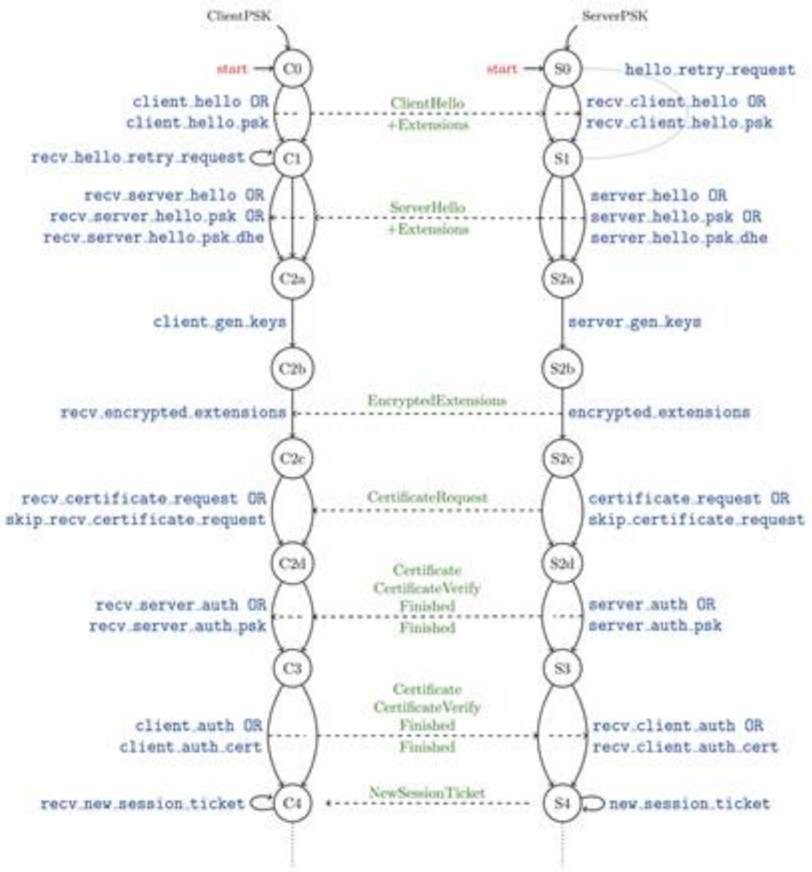


For example, we see that a PSK-only handshake is captured through the invocation of the rules `client_hello_p` `-> recv_client_hello_psk -> server_hello_psk -> ...` for the PSK-DHE handshake, the rule `server_hello_psk_dhe` would be used instead.

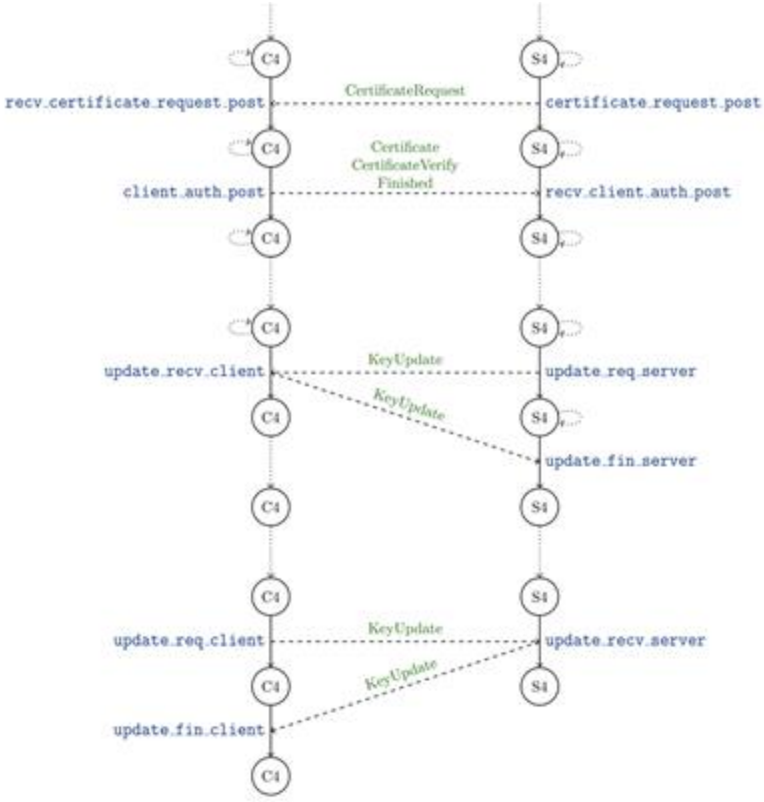
We associate with each handshake *message* (i.e. not necessarily each flight) a distinct rule, to help separate concerns.

https://samscott89.github.io/TLS13_Tamarin/

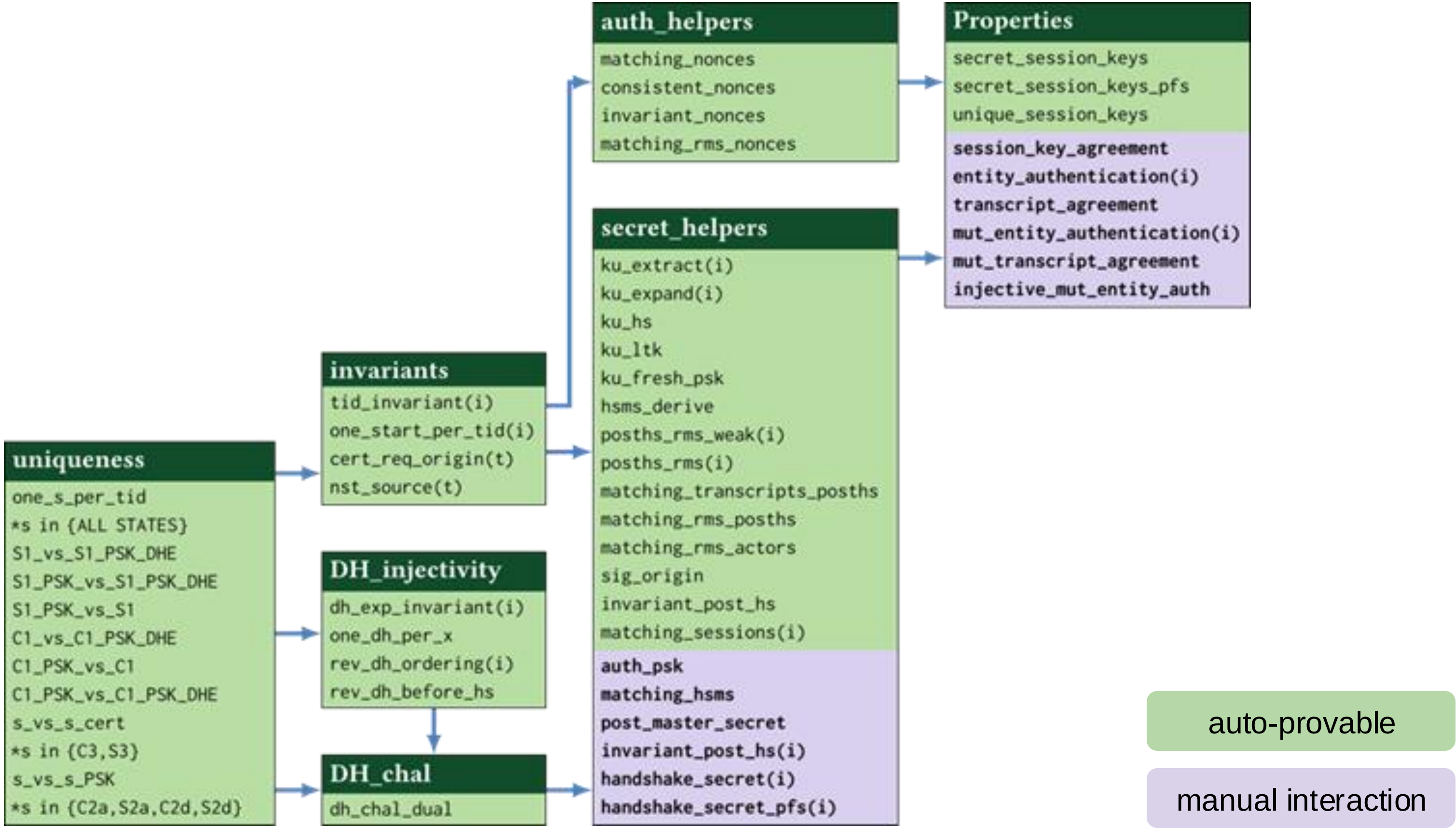
Step 1: Building the Model



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	✓
Uniqueness of session keys	✓
Downgrade protection (within 1.3)	✓
Perfect forward secrecy	✓
Key Compromise Impersonation (KCI) resistance	✓

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	✓

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

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

Future Work

- Feedback loop - modelling complex protocols is making Tamarin 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	N. Sullivan
Internet-Draft	Cloudflare Inc.
Intended status: Standards Track	June 05, 2018
Expires: December 7, 2018	

Future Work

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

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

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

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

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

Versions: 00

tls — E. Rescorla
Internet-Draft — RTFM, Inc.
Intended status: Experimental — K. Oku
Expires: January 3, 2019 — Fastly
N. Sullivan
Cloudflare
C. Wood
Apple, Inc.
July 02, 2018

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

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

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

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

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

[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

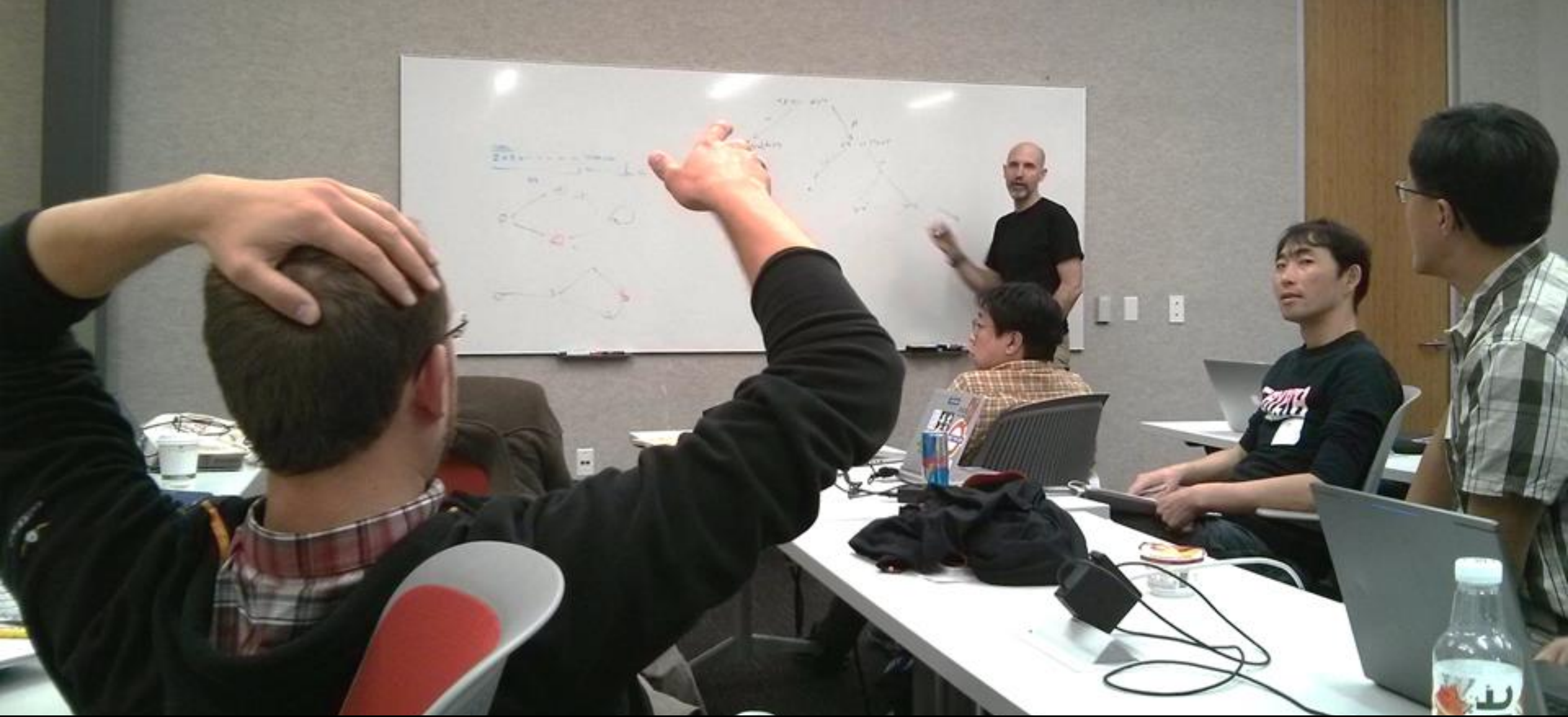
- Logical core of TLS 1.3 seems sound!
- We have built a transparent model others can build on (Github)
- Symbolic analysis
  - Complementary approach to other analysis methods
- Relatively fast turnaround and can directly produce attacks

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

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

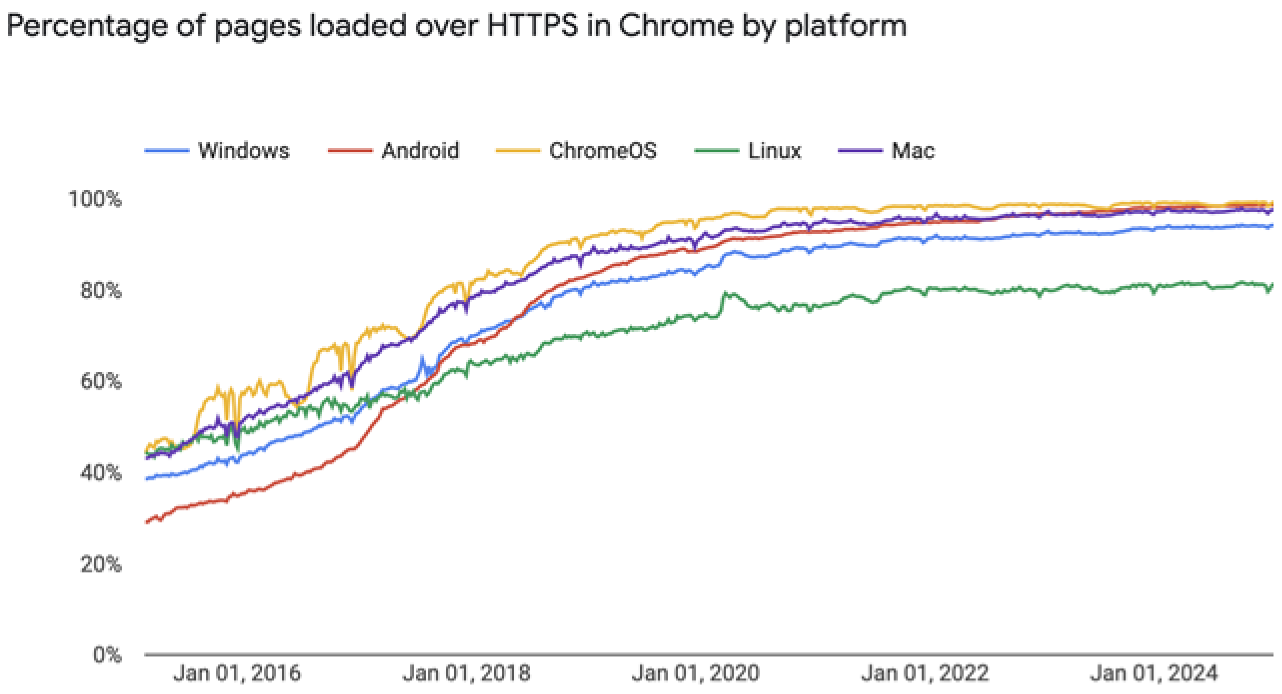
By now, this could really need an update with new Tamarin techniques!



May 2016

Mozilla HQ, Mountain View, CA, USA

HTTP vs HTTPS over time (2016–2024)



Source: Google Transparency Report (https://transparencyreport.google.com/https/overview)

The future

- Then:
    - IETF members were unsure if formal methods could yield benefits
    - TLS analysis was a substantial breakthrough for formal protocol analysis at the time
    - Caught critical potential flaw in TLS 1.3's development

- Now:
    - IETF now highly appreciates formal analysis, and would like to make it a requirement, but technology isn't there yet.
    - Our Tamarin analyis was done in 2015-2016. In 2024, Tamarin has many more advanced features. Open questions:
        - Could we re-do TLS 1.3 in a much nicer model?
        - Could we get to a point where proof is much more automated, and if we adapt it, to check?
        - (we didn't have time for this now)

Next lecture

The future of automated protocol analysis?

- Relation to cryptography
  - More detailed models of cryptographic primitives?
- Scaling analysis to larger systems

Reading material

Recommended reading: [Bas+24, Ch. 19 & Ch. 16, Pat+]

If you are interested in our scientific papers related to TLS 1.3:
https://tls13tamarin.github.io/TLS13Tamarin/

[Bas+24] D. Basin, C. Cremers, J. Dreier, and R. Sasse. **Modeling and Analyzing Security Protocols with Tamarin: A Comprehensive Guide.** Draft v0.7. Jan. 2025.

[PM16] K.G. Paterson and T. v.d. Merwe. **Reactive and Proactive Standardisation of TLS.** SSR 2016.
https://pure.royalholloway.ac.uk/ws/portalfiles/portal/27884959/paper.pdf