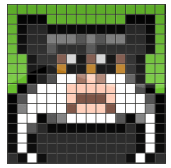
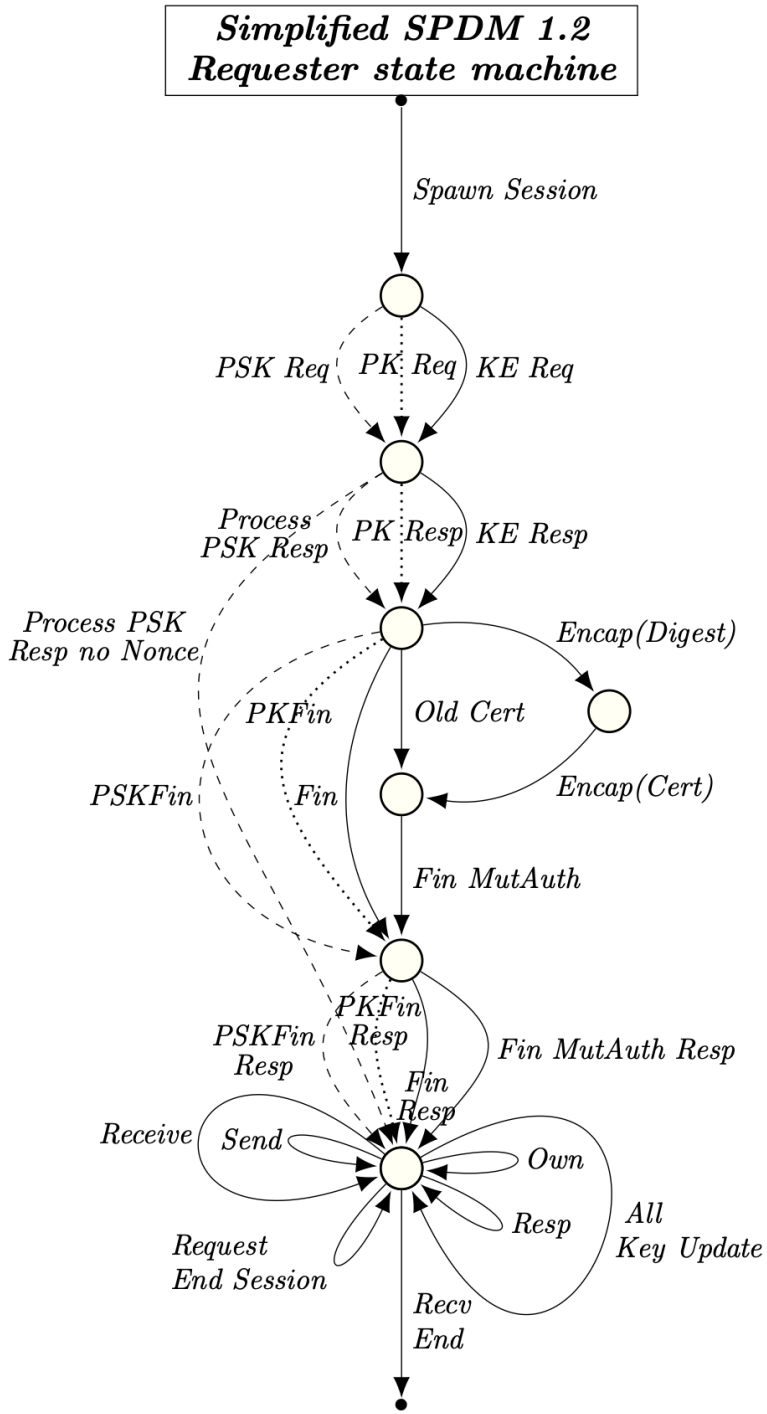
CISPA
HELMHOLTZ CENTER FOR INFORMATION SECURITY



Protocol analysis using Tam

Cas Cremers

May 4, CAPS workshop, Eurocrypt 2025



Tamarin prover

Tamarin prover

Constraint solver

Tamarin prover

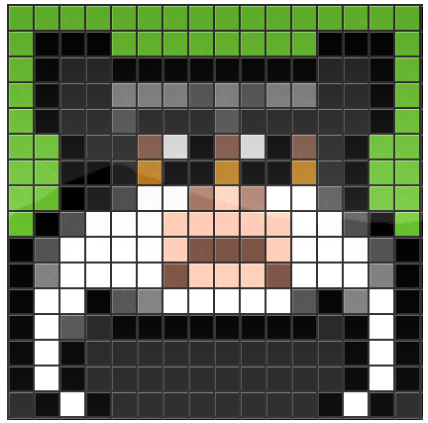
Theorem Prover

Constraint solver

Tamarin prover

Emperor Tamarin

Tamarin prover



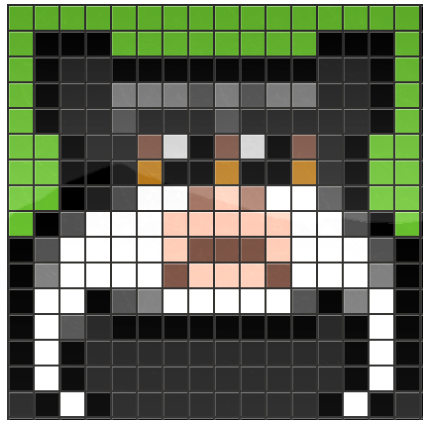
`tamarin-prover.com`

Development started around 2010 at ETH

Current development core:

- CISPA (Cas Cremers)
- ETH Zurich (David Basin, Ralf Sasse)
- INRIA (Jannik Dreier)

Tamarin prover



`tamarin-prover.com`

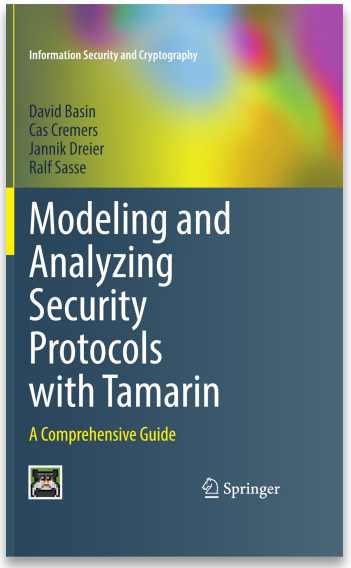
Development started around 2010 at ETH

Current development core:

- CISPA (Cas Cremers)
- ETH Zurich (David Basin, Ralf Sasse)
- INRIA (Jannik Dreier)

Open-source development, with

- Manual
- Online tutorials
- Active mailing list
- Syntax highlighting (vim, VSC, …)
- Upcoming **book**! *Free for download*
  “Modeling and Analyzing Security Protocols with Tamarin: A Comprehensive Guide”



Attacks Tamarin can find or prove absent

- Large attack scenarios (TLS 1.3 Rev 10: 18 messages)
- Cross-protocol attacks
- Unintended state machine transitions
- Downgrade attacks
- Nonce-reuse attacks (eg WiFi with AES-GCM)
- Invalid Curve Points
- Small order points
- DSKS attacks (Duplicate Signature Key Selection)
- Length extension attacks
- Maliciously generated keys
- Hybrid schemes

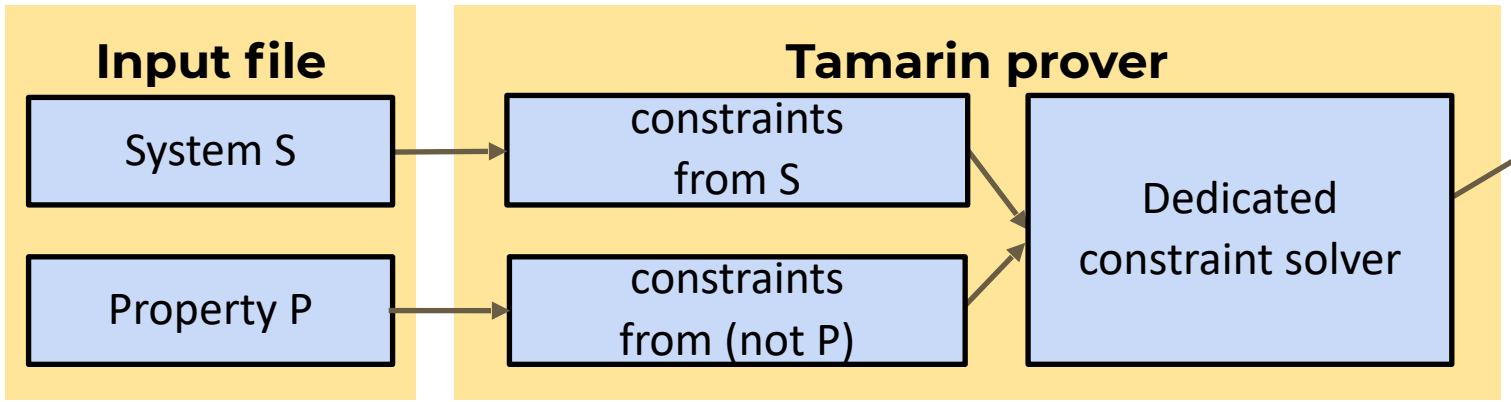
Attacks Tamarin can find or prove absent

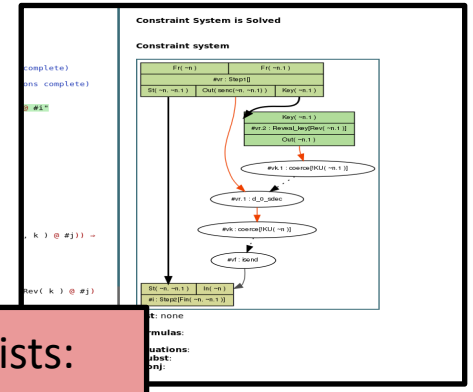
- Large attack scenarios (TLS 1.3 Rev 10: 18 messages)
- Cross-protocol attacks
- Unintended state machine transitions
- Downgrade attacks
- Nonce-reuse attacks (eg WiFi with AES-GCM)
- Invalid Curve Points
- Small order points
- DSKS attacks (Duplicate Signature
- Length extension attacks
- Maliciously generated keys
- Hybrid schemes

These attack types are NOT hard-coded!

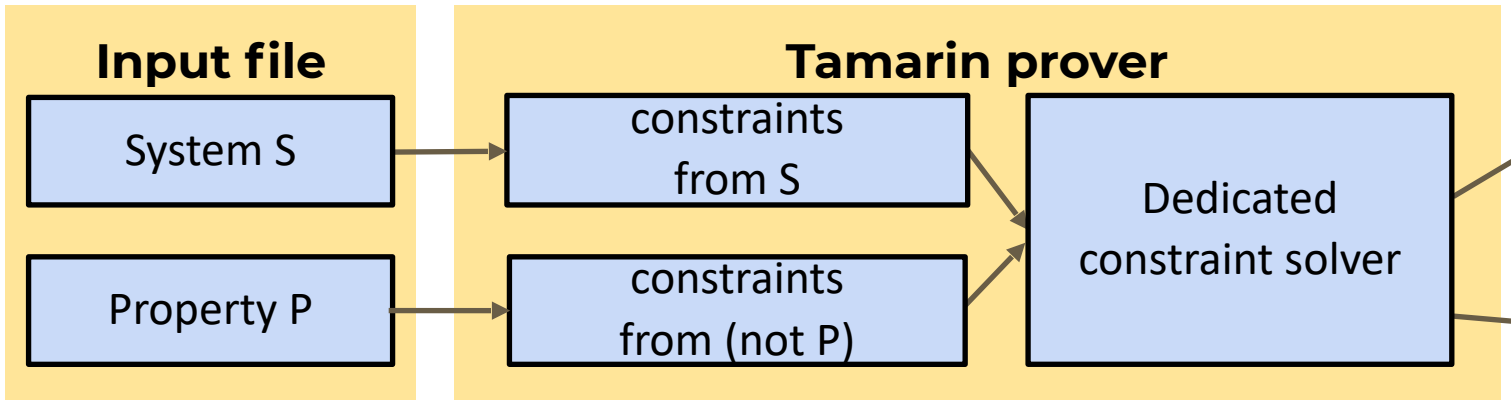
Tamarin finds these through systematic, accurate modeling of cryptography and protocols.

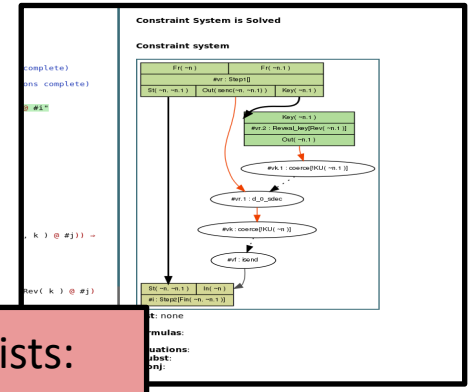
Tamarin prover: workflow



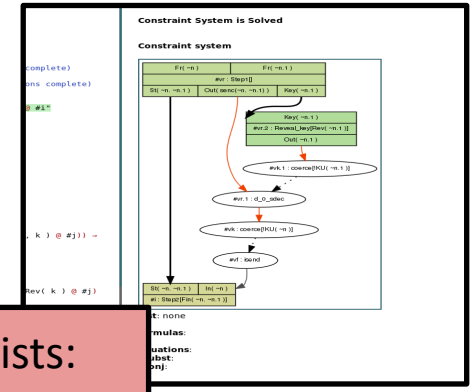


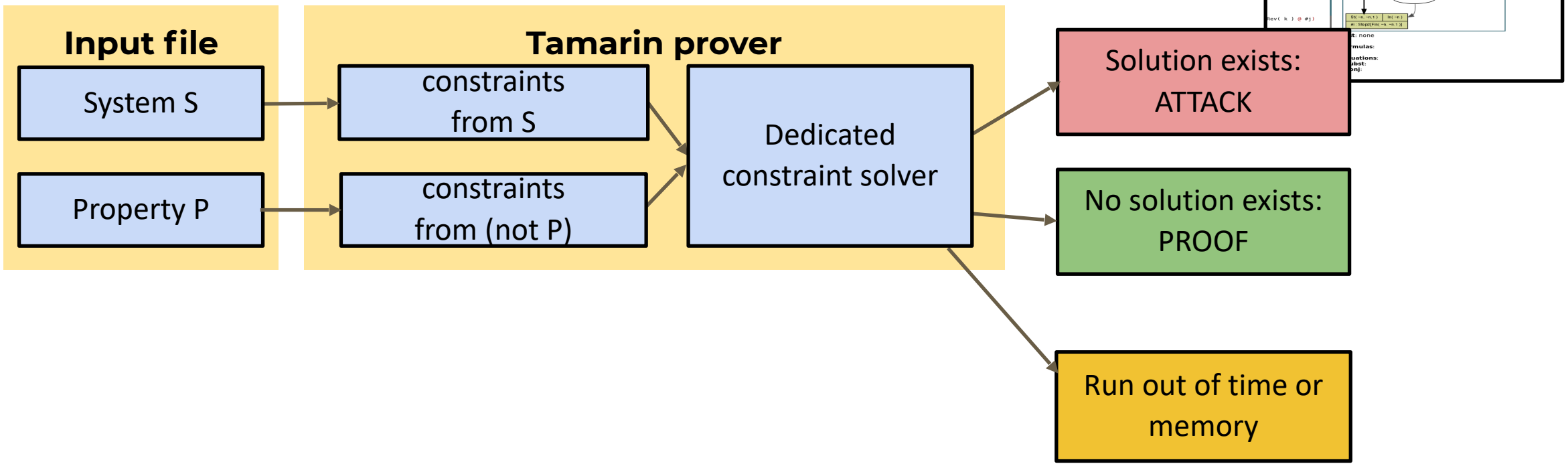
Tamarin prover: workflow



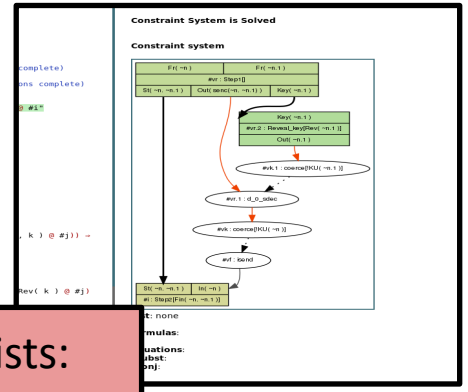


Tamarin prover: workflow





Tamarin prover: workflow



Input file

System S

Property P

Tamarin prover

constraints from S

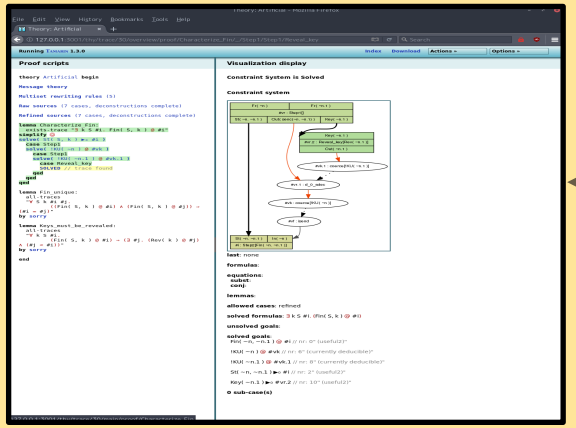
constraints from (not P)

Dedicated constraint solver

Solution exists: ATTACK

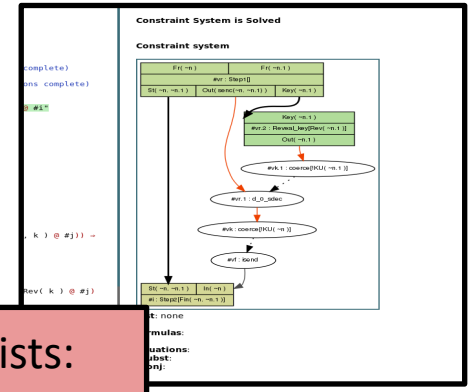
No solution exists: PROOF

Run out of time or memory



Interactive mode
(Inspect or guide partial proof)

Tamarin prover: workflow



Input file

System S

Property P

Tamarin prover

constraints from S

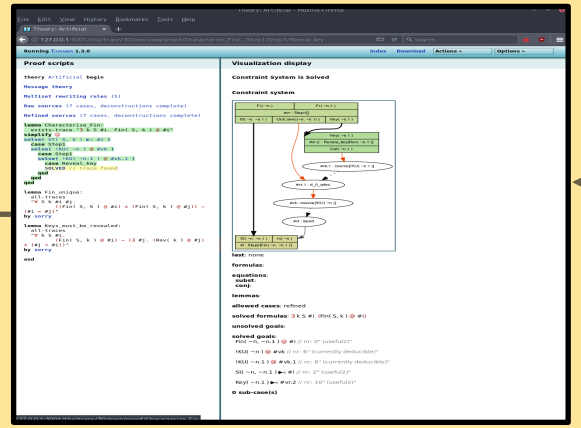
constraints from (not P)

Dedicated constraint solver

Solution exists: ATTACK

No solution exists: PROOF

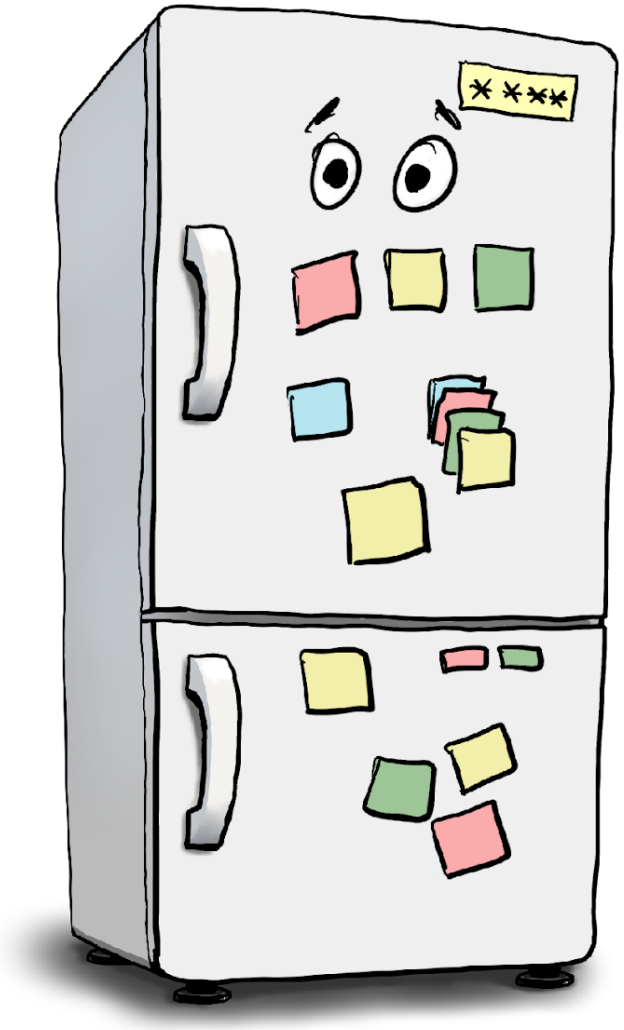
Run out of time or memory



Interactive mode
(Inspect or guide partial proof)

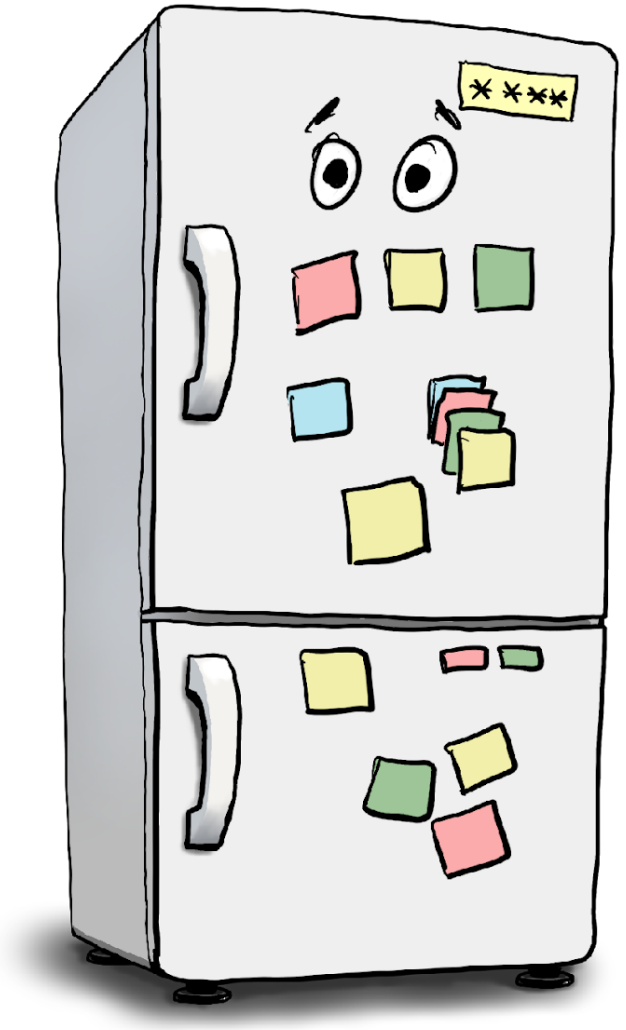
Provide **hints** for the prover (e.g. invariants)

Modeling in Tamarin: transition system



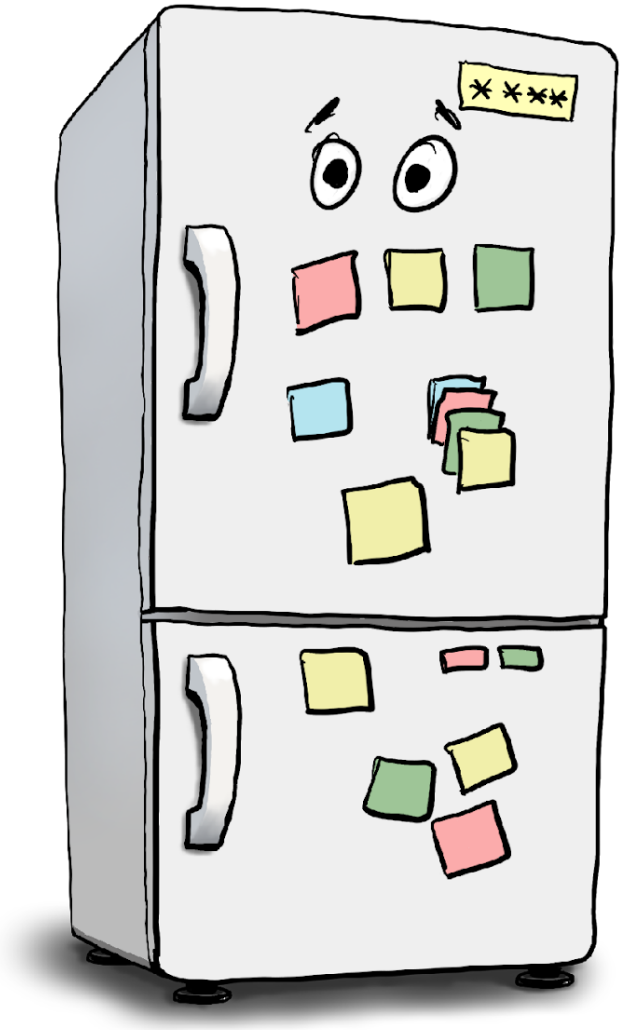
Modeling in Tamarin: transition system

- Basic ingredients:
  - **Terms** (think “messages”)
  - **Facts** (think “sticky notes on the fridge”)
  - Special facts: **Fr(t)**, **In(t)**, **Out(t)**, **K(t)**



Modeling in Tamarin: transition system

- Basic ingredients:
  - **Terms** (think “messages”)
  - **Facts** (think “sticky notes on the fridge”)
  - Special facts: **Fr(t)**, **In(t)**, **Out(t)**, **K(t)**
- State of system is a multiset of facts
  - **Initial state** is the empty multiset
  - **Rules** specify the transition rules (“moves”)

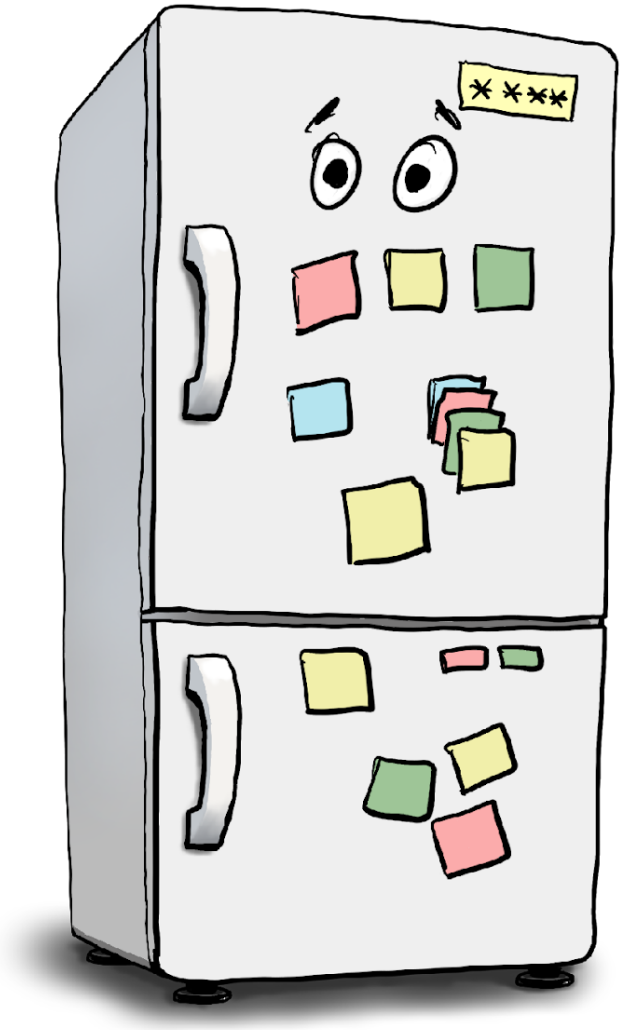


Modeling in Tamarin: transition system

- Basic ingredients:
  - **Terms** (think “messages”)
  - **Facts** (think “sticky notes on the fridge”)
  - Special facts: **Fr(t)**, **In(t)**, **Out(t)**, **K(t)**
- State of system is a multiset of facts
  - **Initial state** is the empty multiset
  - **Rules** specify the transition rules (“moves”)
- Rules are of the form:

  l --> r

  l --[a]-> r



The model

The model

- **Term algebra**
  - enc(_,_), dec(_,_), h(_,_), _^_, $_^{-1}$, _*_, 1, …

The model

- **Term algebra**
  - enc(_,_), dec(_,_), h(_,_), _^_, $_^{-1}$, _*_, 1, …
- **Equational theory**
  - dec(enc(m,k),k) $=_E$ m,
  - (x^y)^z $=_E$ x^(y*z),
  - $(x^{-1})^{-1} =_E$ x, ...

The model

- **Term algebra**
  - enc(_,_), dec(_,_), h(_,_), _^_, $_^{-1}$, _*_, 1, …
- **Equational theory**
  - dec(enc(m,k),k) $=_E$ m,
  - (x^y)^z $=_E$ x^(y*z),
  - $(x^{-1})^{-1} =_E$ x, ...
- **Facts**
  - F(t1,...,tn)

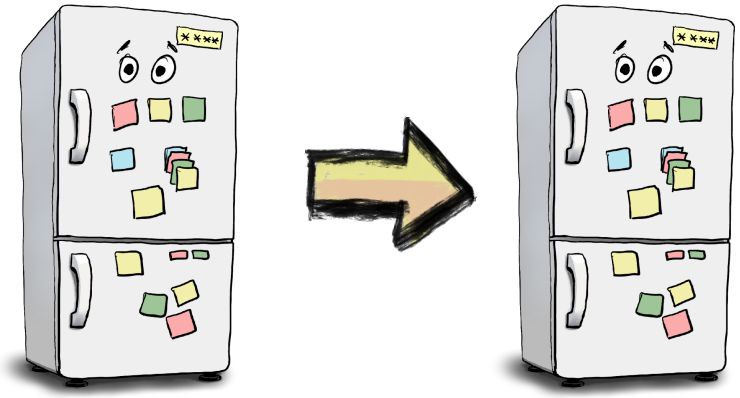
The model

- **Term algebra**
  - enc(_,_), dec(_,_), h(_,_), _^_, $_^{-1}$, _*_, 1, …
- **Equational theory**
  - dec(enc(m,k),k) $=_E$ m,
  - (x^y)^z $=_E$ x^(y*z),
  - $(x^{-1})^{-1} =_E$ x, ...
- **Facts**
  - F(t1,...,tn)
- **Transition system**
  - State: multiset of facts
  - Rules: l –[a]→ r

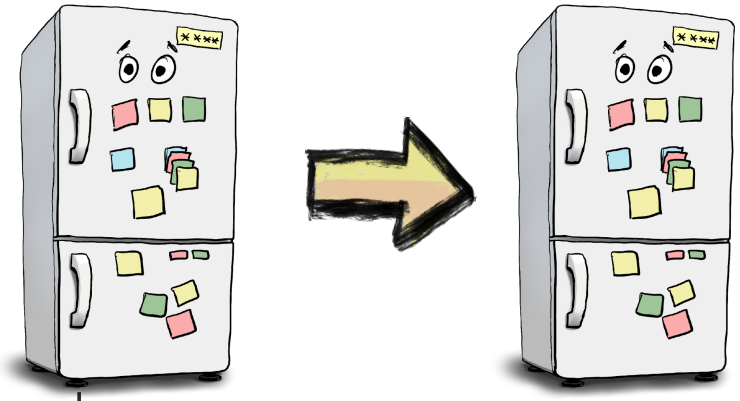
The model

- **Term algebra**
  - enc(_,_), dec(_,_), h(_,_), _^_, _$^{-1}$, _*_, 1, …
- **Equational theory**
  - dec(enc(m,k),k) $=_E$ m,
  - (x^y)^z $=_E$ x^(y*z),
  - $(x^{-1})^{-1} =_E$ x, ...
- **Facts**
  - F(t1,...,tn)
- **Transition system**
  - State: multiset of facts
  - Rules: l –[a]→ r
- **Tamarin-specific**
  - Built-in Dolev-Yao attacker rules:
    In( ),
    Out( ),
    K( )
  - Special **Fresh** rule:
    - [] --[]--> [Fr(**x**)]
      - Constraint on system such that **x** is unique

Semantics



Semantics

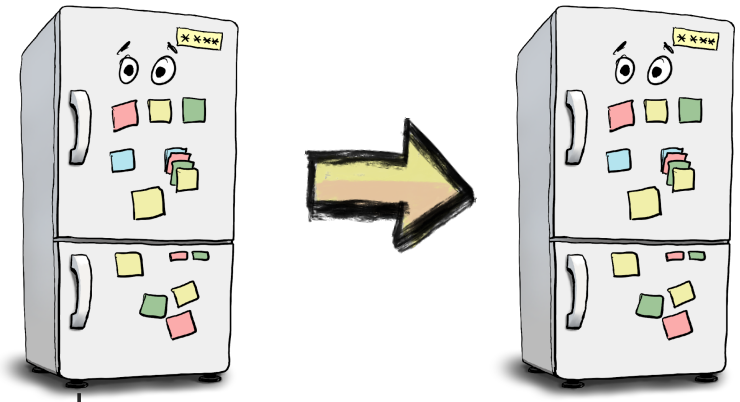


- **Transition relation**

  $S \text{ –}[a]\rightarrow_R ((S \setminus^{\#} I) \cup^{\#} r)$, where

  - $I \text{ –}[a]\rightarrow r$ is a ground instance of a rule in R, and
  - $I \subseteq^{\#} S$ w.r.t. the equational theory

Semantics



- **Transition relation**

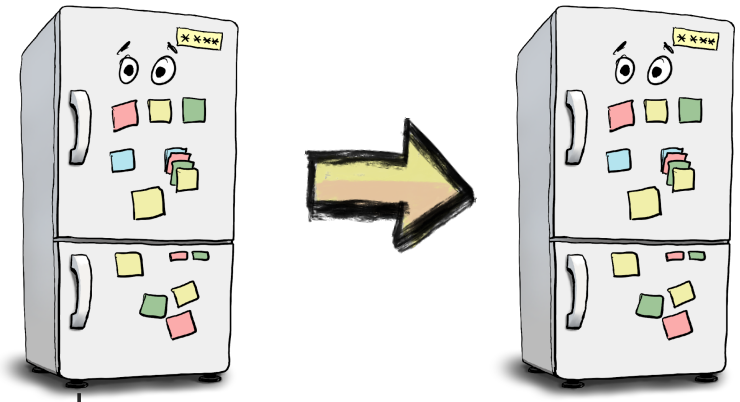
  $S -[a]\rightarrow_R ((S \setminus^\# I) \cup^\# r)$, where

  - $I -[a]\rightarrow r$ is a ground instance of a rule in R, and
  - $I \subseteq^\# S$ w.r.t. the equational theory

- **Executions**

  $-\text{Exec}(R) = \{ [\,] -[a_1]\rightarrow \ldots -[a_n]\rightarrow S_n$
  $\mid \forall n \,.\, \text{Fr}(n)$ appears only once on right-hand side of rule $\}$

Semantics



- **Transition relation**

  $S -[a]\rightarrow_R ((S \setminus^\# I) \cup^\# r)$, where

  - $I -[a]\rightarrow r$ is a ground instance of a rule in R, and
  - $I \subseteq^\# S$ w.r.t. the equational theory

- **Executions**

  $-Exec(R) = \{ [\,] -[a_1]\rightarrow \ldots -[a_n]\rightarrow S_n$
  $\mid \forall n . Fr(n)$ appears only once on right-hand side of rule $\}$

- **Traces**

  $-Traces(R) = \{ [a_1,\ldots,a_n] \mid [\,] -[a_1]\rightarrow \ldots -[a_n]\rightarrow S_n \in Exec(R) \}$

  −Property specification using first-order logic over traces

Semantics: example 1

‘c’	constant

Semantics: example 1

‘c’ constant

- **Rules**
  - rule 1: [ ] –[Init()]→ [A('5')]
  - rule 2: [A(x)] –[Step(x)]→ [B(x)]

Semantics: example 1

‘c’ constant

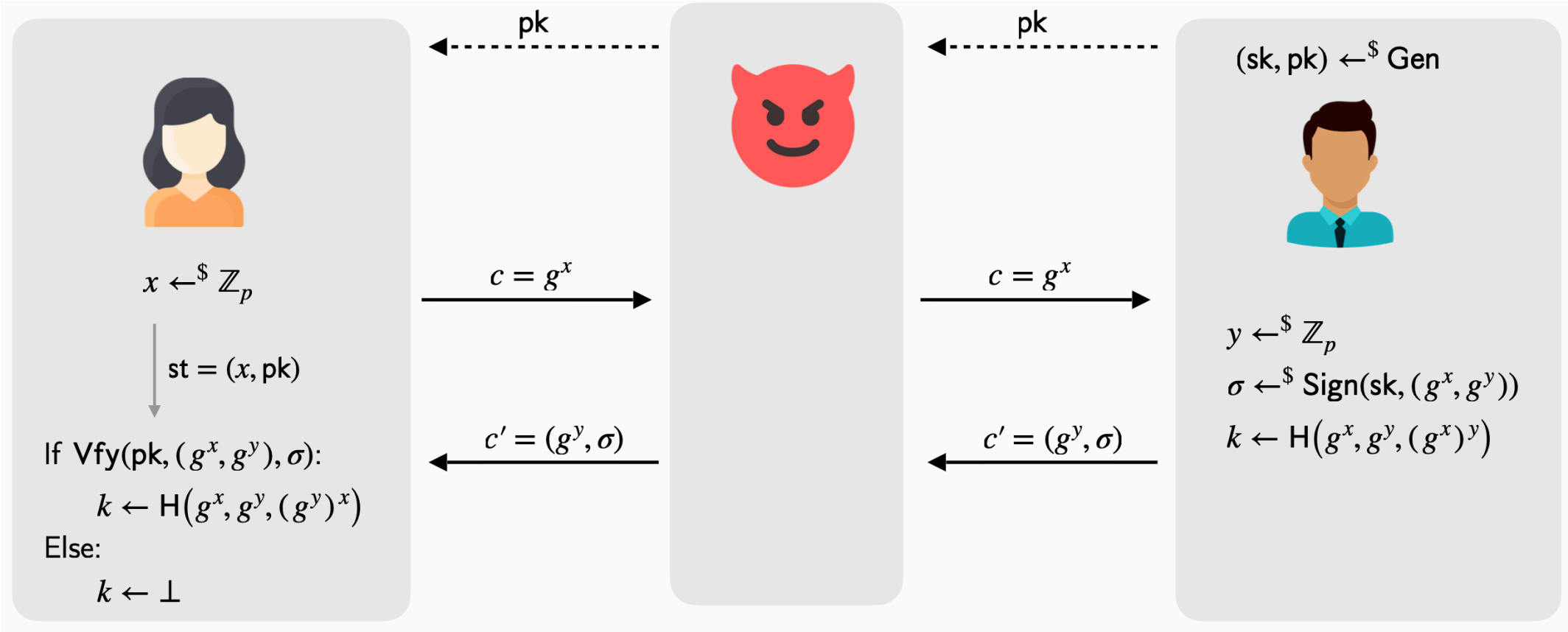
- **Rules**
  - rule 1: [ ] –[Init()]→ [A('5')]
  - rule 2: [A(x)] –[Step(x)]→ [B(x)]
- **Execution example**
  - [ ]
  - –[Init()]→ [A('5’)]
  - –[Init()]→ [A('5'), A('5')]
  - –[Step('5')]→ [A('5'), B('5')]

Semantics: example 1

'c' constant

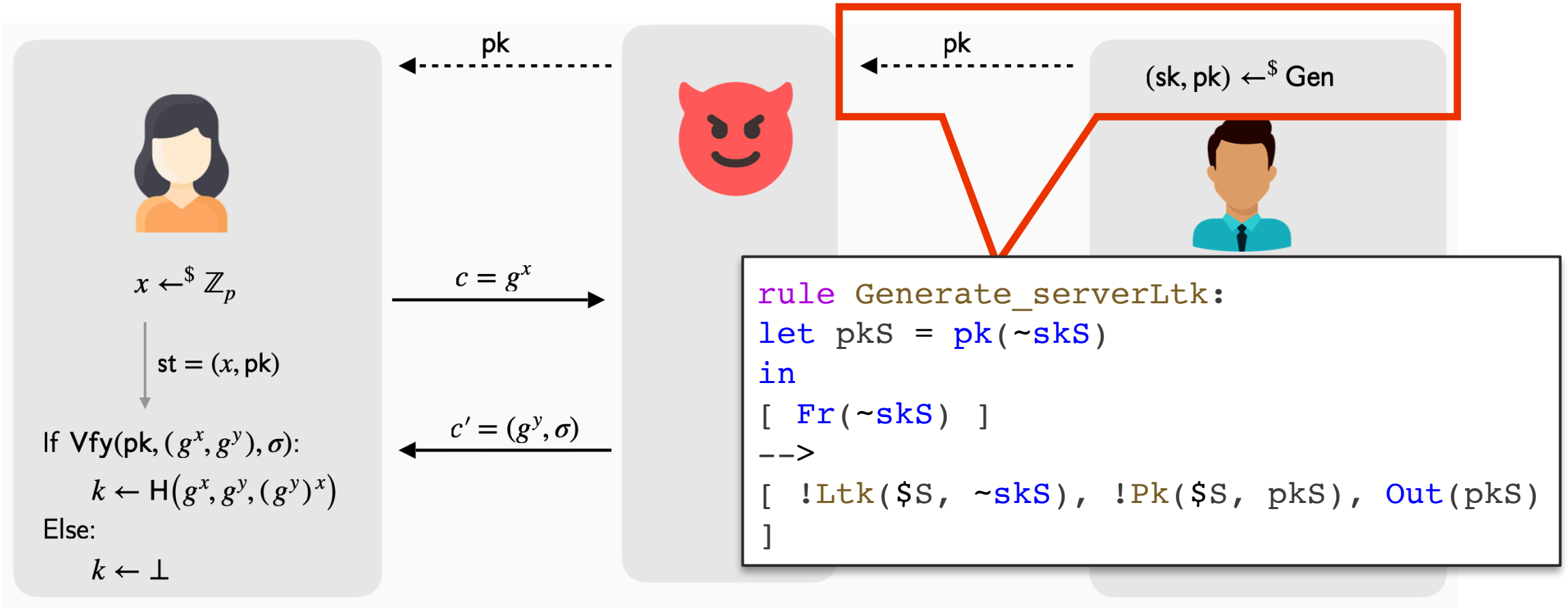
- **Rules**
  - rule 1: [ ] –[Init()]→ [A('5')]
  - rule 2: [A(x)] –[Step(x)]→ [B(x)]
- **Execution example**
  - [ ]
  - –[Init()]→ [A('5')]
  - –[Init()]→ [A('5'), A('5')]
  - –[Step('5')]→ [A('5'), B('5')]
- **Corresponding trace:** [Init(), Init(), Step('5')]

Unilateral signed Diffie-Hellman



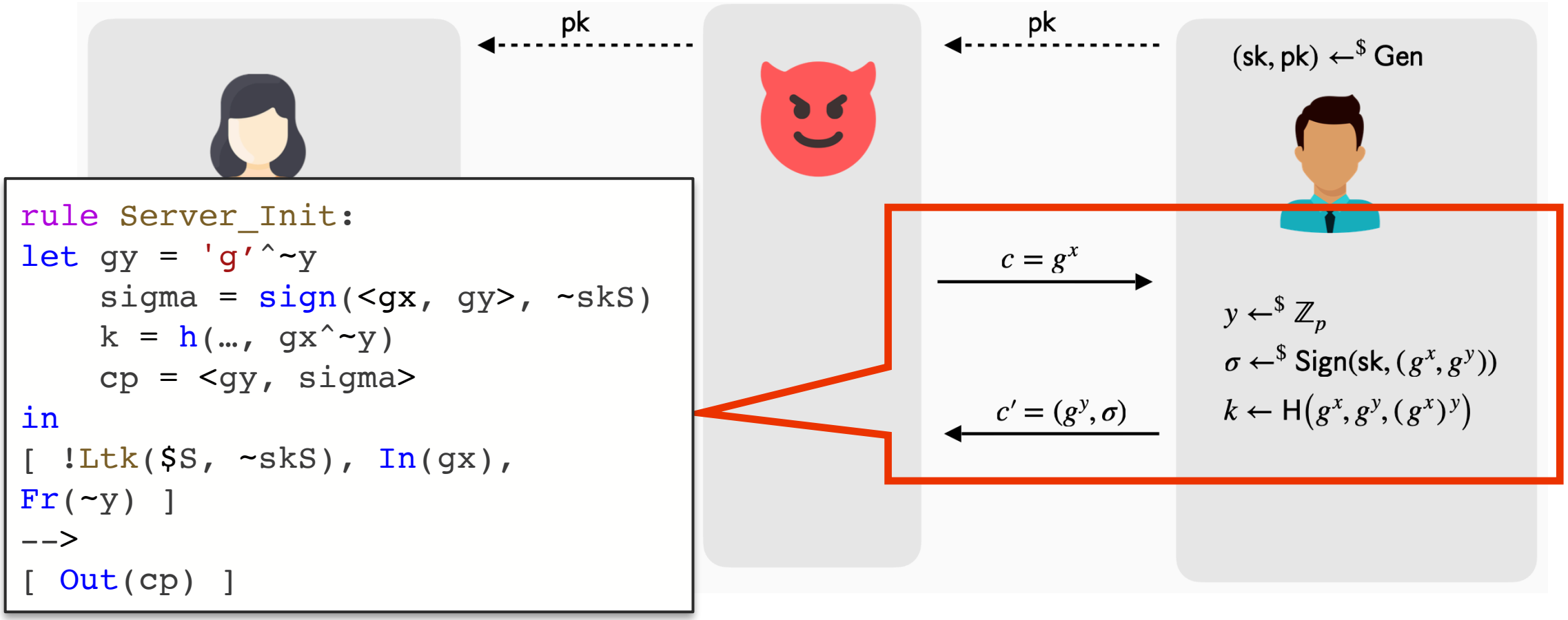
Key setup

‘c’	constant
~x	fresh type
$x	public type



Server

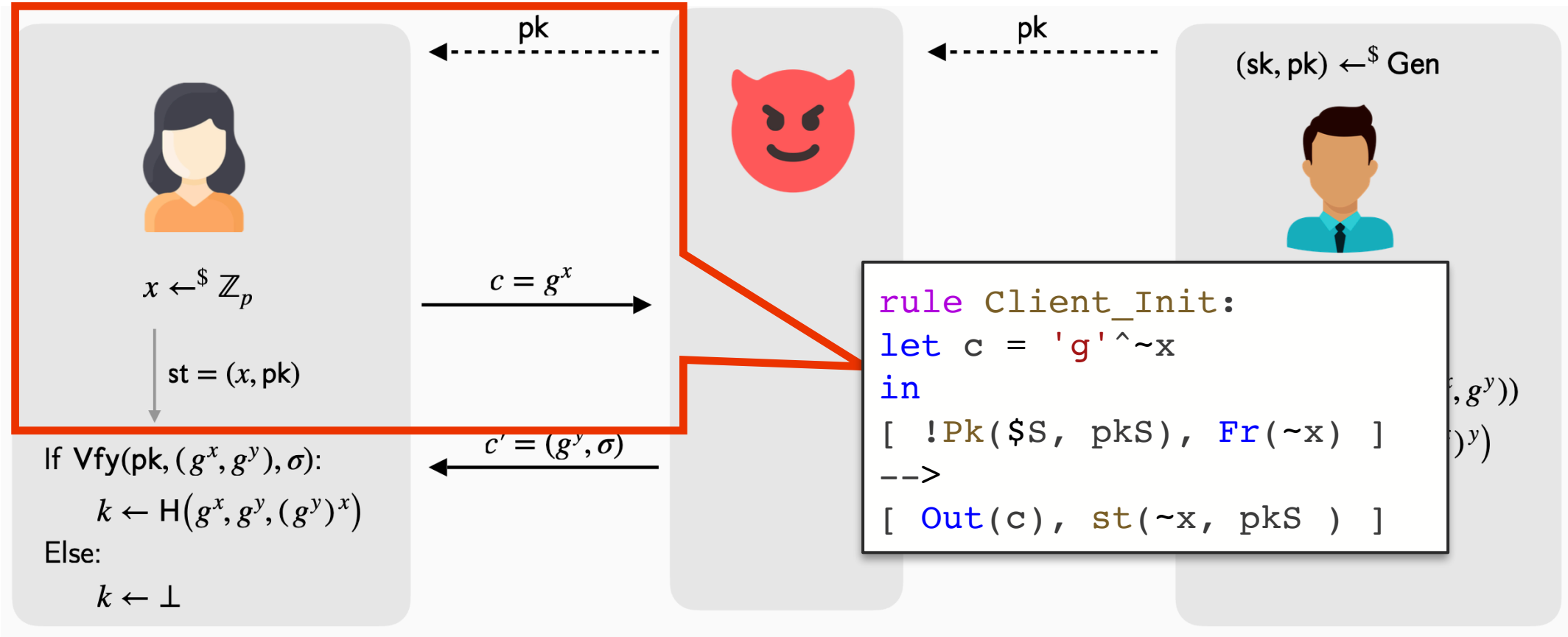
‘c’	constant
~x	fresh type
$x	public type




```
rule Server_Init:
let gy = 'g'^~y
    sigma = sign(<gx, gy>, ~skS)
    k = h(…, gx^~y)
    cp = <gy, sigma>
in
[ !Ltk($S, ~skS), In(gx),
Fr(~y) ]
-->
[ Out(cp) ]
```

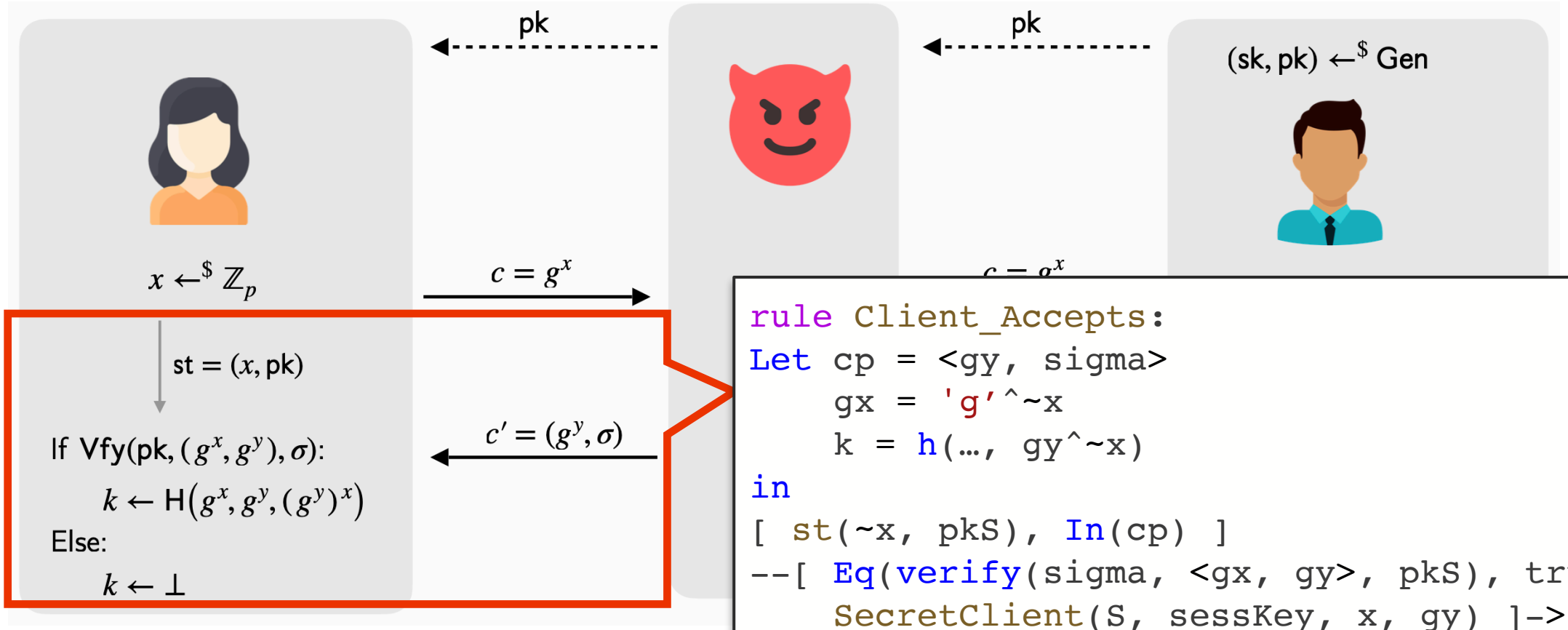
Client (1/2)

‘c’	constant
~x	fresh type
$x	public type



Client (2/2)

‘c’	constant
~x	fresh type
$x	public type

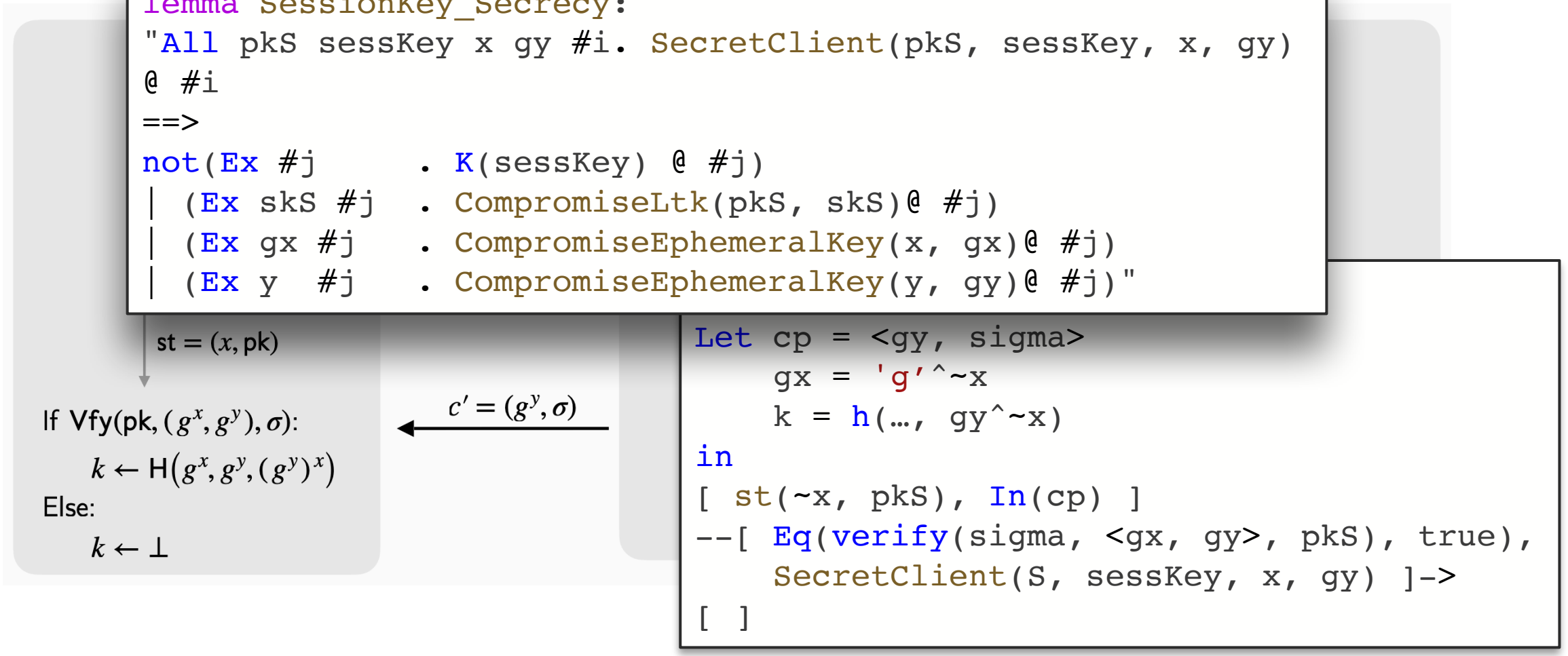



```
rule Client_Accepts:
Let cp = <gy, sigma>
    gx = 'g'^~x
    k = h(…, gy^~x)
in
[ st(~x, pkS), In(cp) ]
--[ Eq(verify(sigma, <gx, gy>, pkS), true),
    SecretClient(S, sessKey, x, gy) ]->
[ ]
```

Security properties

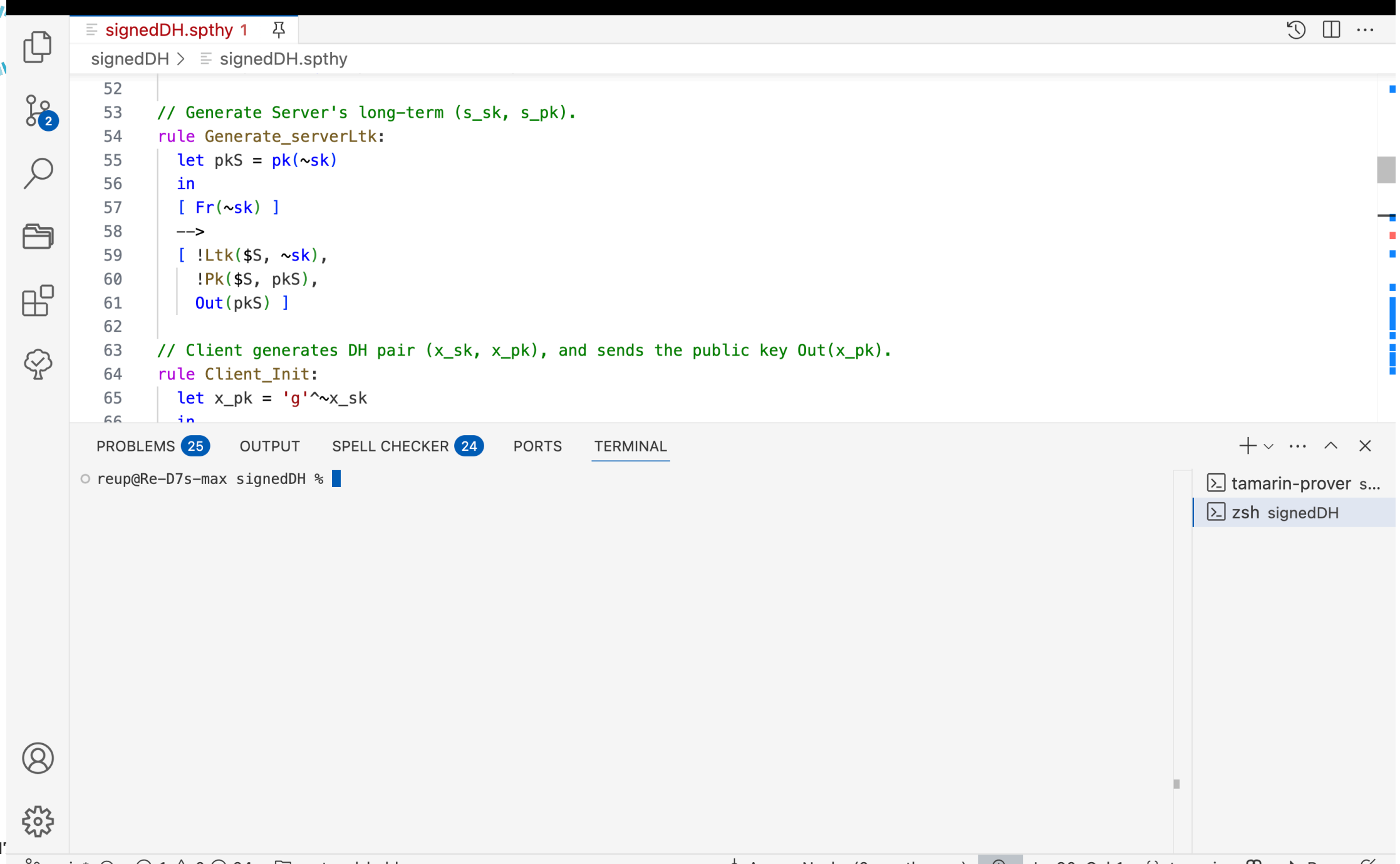
‘c’	constant
~x	fresh type
$x	public type

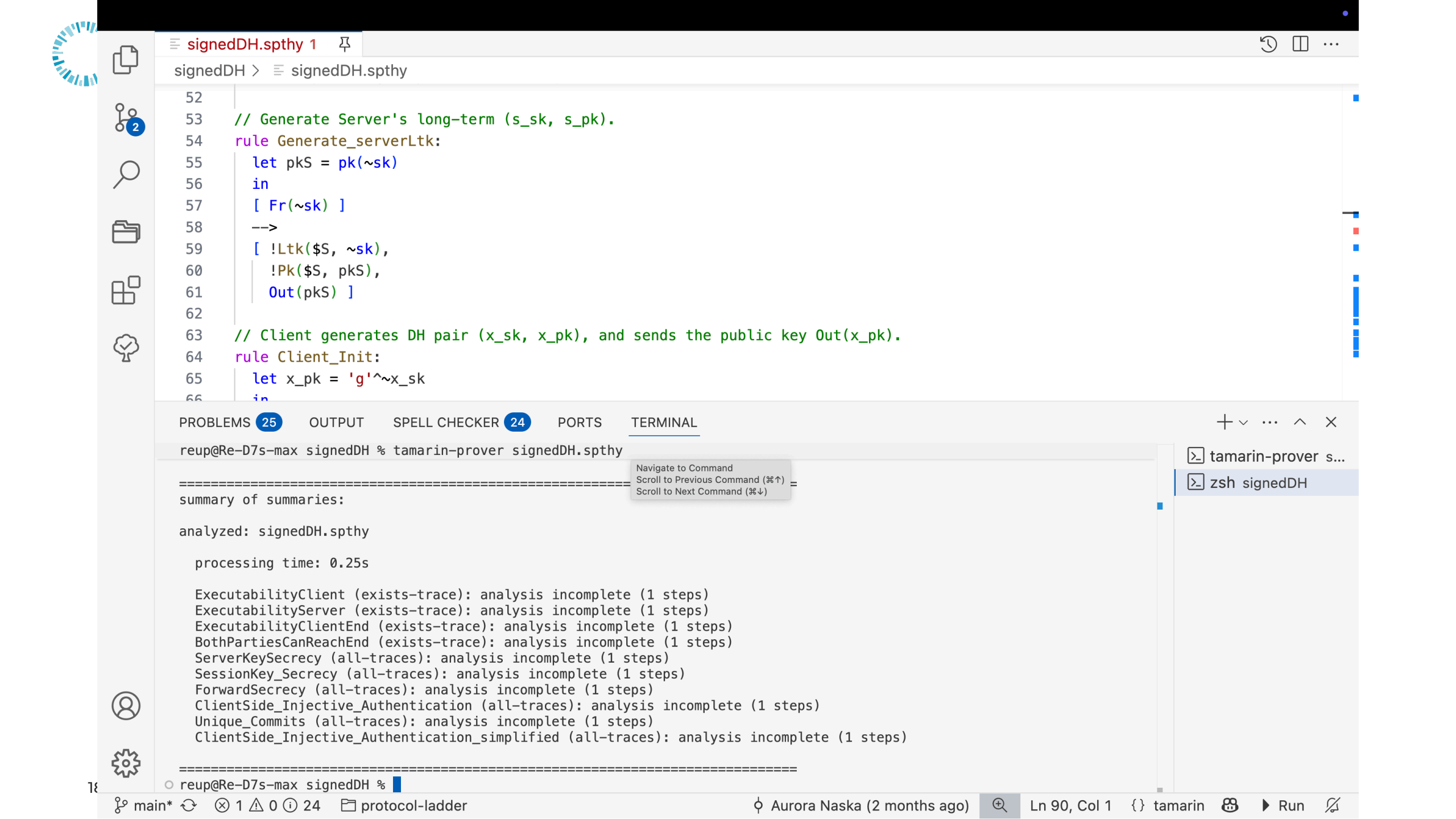
```
lemma SessionKey_Secrecy:
"All pkS sessKey x gy #i. SecretClient(pkS, sessKey, x, gy)
@ #i
==>
not(Ex #j      . K(sessKey) @ #j)
| (Ex skS #j   . CompromiseLtk(pkS, skS)@ #j)
| (Ex gx #j    . CompromiseEphemeralKey(x, gx)@ #j)
| (Ex y  #j    . CompromiseEphemeralKey(y, gy)@ #j)"
```

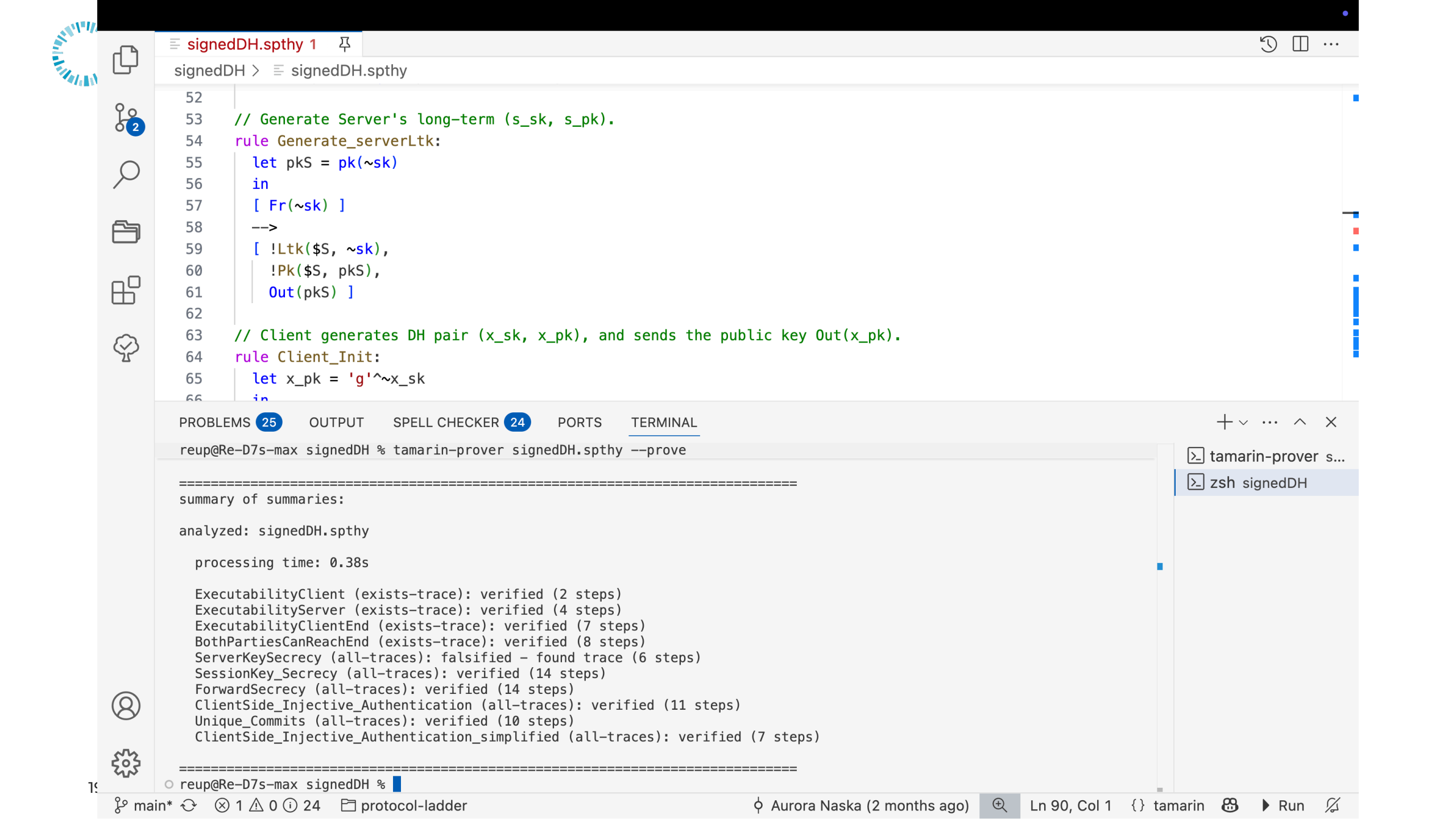



```
Let cp = <gy, sigma>
    gx = 'g'^~x
    k = h(…, gy^~x)
in
[ st(~x, pkS), In(cp) ]
--[ Eq(verify(sigma, <gx, gy>, pkS), true),
    SecretClient(S, sessKey, x, gy) ]->
[ ]
```

DEMO (command line)


signedDH.spthy 1
signedDH > signedDH.spthy
52
53 // Generate Server's long-term (s_sk, s_pk).
54 rule Generate_serverLtk:
55 let pkS = pk(~sk)
56 in
57 [Fr(~sk)]
58 -->
59 [!Ltk($S, ~sk),
60 !Pk($S, pkS),
61 Out(pkS)]
62
63 // Client generates DH pair (x_sk, x_pk), and sends the public key Out(x_pk).
64 rule Client_Init:
65 let x_pk = 'g'^~x_sk
PROBLEMS 25
OUTPUT
SPELL CHECKER 24
PORTS
TERMINAL
reup@Re-D7s-max signedDH %
tamarin-prover s...
zsh signedDH
main*
1 0 24
protocol-ladder
Aurora Naska (2 months ago)
Ln 90, Col 1
tamarin
Run


signedDH.spthy 1
signedDH > signedDH.spthy
52
53 // Generate Server's long-term (s_sk, s_pk).
54 rule Generate_serverLtk:
55 let pkS = pk(~sk)
56 in
57 [Fr(~sk)]
58 -->
59 [!Ltk($S, ~sk),
60 !Pk($S, pkS),
61 Out(pkS)]
62
63 // Client generates DH pair (x_sk, x_pk), and sends the public key Out(x_pk).
64 rule Client_Init:
65 let x_pk = 'g'^~x_sk
PROBLEMS 25
OUTPUT
SPELL CHECKER 24
PORTS
TERMINAL
reup@Re-D7s-max signedDH % tamarin-prover signedDH.spthy
Navigate to Command
Scroll to Previous Command (⌘↑)
Scroll to Next Command (⌘↓)
summary of summaries:
analyzed: signedDH.spthy
processing time: 0.25s
ExecutabilityClient (exists-trace): analysis incomplete (1 steps)
ExecutabilityServer (exists-trace): analysis incomplete (1 steps)
ExecutabilityClientEnd (exists-trace): analysis incomplete (1 steps)
BothPartiesCanReachEnd (exists-trace): analysis incomplete (1 steps)
ServerKeySecrecy (all-traces): analysis incomplete (1 steps)
SessionKey_Secrecy (all-traces): analysis incomplete (1 steps)
ForwardSecrecy (all-traces): analysis incomplete (1 steps)
ClientSide_Injective_Authentication (all-traces): analysis incomplete (1 steps)
Unique_Commits (all-traces): analysis incomplete (1 steps)
ClientSide_Injective_Authentication_simplified (all-traces): analysis incomplete (1 steps)
reup@Re-D7s-max signedDH %
tamarin-prover s...
zsh signedDH
main*
1 0 24
protocol-ladder
Aurora Naska (2 months ago)
Ln 90, Col 1
tamarin
Run


signedDH.spthy 1
signedDH > signedDH.spthy
52
53 // Generate Server's long-term (s_sk, s_pk).
54 rule Generate_serverLtk:
55 let pkS = pk(~sk)
56 in
57 [Fr(~sk)]
58 -->
59 [!Ltk($S, ~sk),
60 !Pk($S, pkS),
61 Out(pkS)]
62
63 // Client generates DH pair (x_sk, x_pk), and sends the public key Out(x_pk).
64 rule Client_Init:
65 let x_pk = 'g'^~x_sk
PROBLEMS 25
OUTPUT
SPELL CHECKER 24
PORTS
TERMINAL
reup@Re-D7s-max signedDH % tamarin-prover signedDH.spthy --prove
summary of summaries:
analyzed: signedDH.spthy
processing time: 0.38s
ExecutabilityClient (exists-trace): verified (2 steps)
ExecutabilityServer (exists-trace): verified (4 steps)
ExecutabilityClientEnd (exists-trace): verified (7 steps)
BothPartiesCanReachEnd (exists-trace): verified (8 steps)
ServerKeySecrecy (all-traces): falsified - found trace (6 steps)
SessionKey_Secrecy (all-traces): verified (14 steps)
ForwardSecrecy (all-traces): verified (14 steps)
ClientSide_Injective_Authentication (all-traces): verified (11 steps)
Unique_Commits (all-traces): verified (10 steps)
ClientSide_Injective_Authentication_simplified (all-traces): verified (7 steps)
reup@Re-D7s-max signedDH %
tamarin-prover s...
zsh signedDH
main*
1 0 24
protocol-ladder
Aurora Naska (2 months ago)
Ln 90, Col 1
tamarin
Run

DEMO (GUI)

Proof scripts

```
lemma SessionKey_Secrecy:
  all-traces
  "∀ S sessKey x_sk y_pk #i.
         (SecretClient( S, sessKey, x_sk, y_pk ) @ #i) ⇒
         ((((¬(∃ #j. K( sessKey ) @ #j)) ∨
            (∃ s_sk #j. CompromiseLtk( S, s_sk ) @ #j)) ∨
           (∃ x_pk #j. CompromiseEphemeralKey( x_sk, x_pk
          (∃ y_sk #j. CompromiseEphemeralKey( y_sk, y_pk )
edit lemma or delete lemma
by sorry /* removed */

add lemma

lemma ForwardSecrecy:
  all-traces
  "∀ S sessKey x_sk y_pk #i #k.
         ((SecretClient( S, sessKey, x_sk, y_pk ) @ #i) ∧
          (K( sessKey ) @ #k)) ⇒
         (((∃ s_sk #j. (CompromiseLtk( S, s_sk ) @ #j) ∧ (
           (∃ x_pk #j. CompromiseEphemeralKey( x_sk, x_pk
          (∃ y_sk #j. CompromiseEphemeralKey( y_sk, y_pk )
edit lemma or delete lemma
by sorry

add lemma

lemma ClientSide_Injective_Authentication:
  all-traces
  "∀ S x_pk y_pk s_pk serverSign sessKey #i.
         (ClientAccepts( S, x_pk, y_pk, s_pk, serverSign,
         ((∃ #j.
            ((#j < #i) ∧
             (ServerAccepts( S, x_pk, y_pk, s_pk, serverSi
            (¬(∃ #t.
                (ClientAccepts( S, x_pk, y_pk, s_pk, serve
                (¬(#t = #i))))) ∨
          (∃ s_sk #j. CompromiseLtk( S, s_sk ) @ #j))"
edit lemma or delete lemma
```

Lemma: SessionKey_Secrecy

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
s. autoprove (S. for all solutions) for all lemmas

Constraint system

last: none

formulas:

```
∃ S sessKey x_sk y_pk #i.
  (SecretClient( S, sessKey, x_sk, y_pk ) @ #i)
∧
  (∃ #j. (K( sessKey ) @ #j)) ∧
  (∀ s_sk #j. (CompromiseLtk( S, s_sk ) @ #j) ⇒ ⊥) ∧
  (∀ x_pk #j. (CompromiseEphemeralKey( x_sk, x_pk ) @ #j) ⇒ ⊥) ∧
  (∀ y_sk #j. (CompromiseEphemeralKey( y_sk, y_pk ) @ #j) ⇒ ⊥)
```

subterms:

equations:
  subst:
  conj:

lemmas: ∀ x y #i. (Eq(x, y) @ #i) ⇒ x = y

allowed cases: refined

solved formulas:

Proof scripts

```
lemma SessionKey_Secrecy:
  all-traces
  "∀ S sessKey x_sk y_pk #i.
        (SecretClient( S, sessKey, x_sk, y_pk ) @ #i) ⇒
        ((((¬(∃ #j. K( sessKey ) @ #j)) ∨
           (∃ s_sk #j. CompromiseLtk( S, s_sk ) @ #j)) ∨
          (∃ x_pk #j. CompromiseEphemeralKey( x_sk, x_pk
         (∃ y_sk #j. CompromiseEphemeralKey( y_sk, y_pk )
edit lemma or delete lemma
simplify
solve( StateC( $C, ~x_sk ) ▶₀ #i )
  case Client_Init
  solve( !Pk( $S, pk(x) ) ▶₁ #i )
    case Generate_serverLtk
    solve( !KU( sign(<'g'^~x_sk, y_pk>, ~sk) ) @ #vk.2 )
      case Server_Init
      solve( !KU( h('g'^(~x_sk*~y_sk)) ) @ #vk.3 )
        case c_h
        solve( !KU( 'g'^(~x_sk*~y_sk) ) @ #vk.5 )
          case Client_Init
          solve( !KU( ~y_sk ) @ #vk.6 )
            case Compromise_EphemeralKey
            by contradiction /* from formulas */
          qed
        next
          case Server_Init
          solve( !KU( ~x_sk ) @ #vk.6 )
            case Compromise_EphemeralKey
            by contradiction /* from formulas */
          qed
        next
          case c_exp
          solve( !KU( ~x_sk ) @ #vk.8 )
            case Compromise_EphemeralKey
            by contradiction /* from formulas */
          qed
        qed
      qed
    next
```

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
s. autoprove (S. for all solutions) for all lemmas

Constraint system

last: none

formulas:
∃ S sessKey x_sk y_pk #i #k.
(SecretClient(S, sessKey, x_sk, y_pk) @ #i) ∧ (K(sessKey) @ #k)
∧
(∀ s_sk #j. (CompromiseLtk(S, s_sk) @ #j) ⇒ ¬(#j < #i)) ∧
(∀ x_pk #j. (CompromiseEphemeralKey(x_sk, x_pk) @ #j) ⇒ ⊥) ∧
(∀ y_sk #j. (CompromiseEphemeralKey(y_sk, y_pk) @ #j) ⇒ ⊥)

subterms:

equations:
subst:
conj:

lemmas: ∀ x y #i. (Eq(x, y) @ #i) ⇒ x = y

allowed cases: refined

solved formulas:

Proof scripts

```
lemma ServerKeySecrecy:
  all-traces
  "∀ S sessKey #i.
         (SecretServer( S, sessKey ) @ #i) ⇒
         (((¬(∃ #j. K( sessKey ) @ #j)) ∨
           (∃ s_sk #j. CompromiseLtk( S, s_sk ) @ #j)) ∨
          (∃ eKey ePubKey #j.
            CompromiseEphemeralKey( eKey, ePubKey ) @ #j))
edit lemma or delete lemma
simplify
solve( !Ltk( $S, ~s_sk ) ▶₀ #i )
  case Generate_serverLtk
  solve( splitEqs(0) )
    case split_case_1
    solve( !KU( h(x_pk^~y_sk) ) @ #vk.1 )
      case c_h
      solve( !KU( x_pk^~y_sk ) @ #vk.2 )
        case Server_Init
        SOLVED // trace found
      qed
    qed
  qed
qed

add lemma

lemma SessionKey_Secrecy:
  all-traces
  "∀ S sessKey x_sk y_pk #i.
         (SecretClient( S, sessKey, x_sk, y_pk ) @ #i) ⇒
         ((((¬(∃ #j. K( sessKey ) @ #j)) ∨
            (∃ s_sk #j. CompromiseLtk( S, s_sk ) @ #j)) ∨
           (∃ x_pk #j. CompromiseEphemeralKey( x_sk, x_pk
          (∃ y_sk #j. CompromiseEphemeralKey( y_sk, y_pk )
edit lemma or delete lemma
by sorry /* removed */

add lemma
```

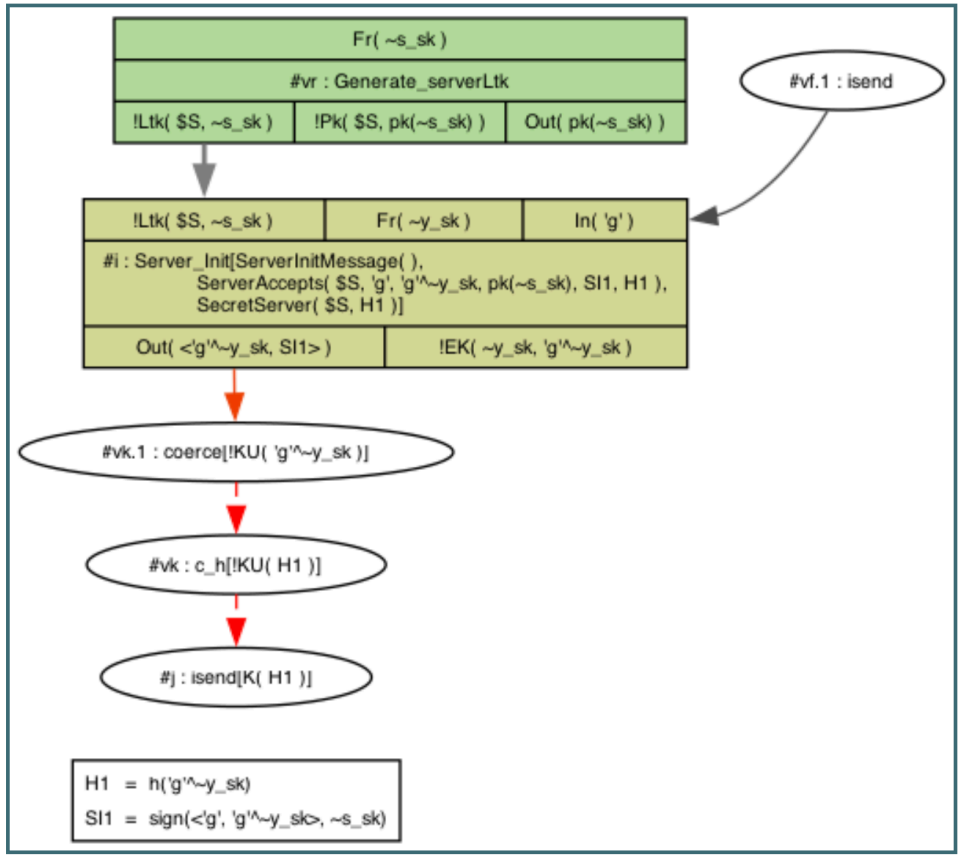
Visualization display

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

1. SOLVED // trace found

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

Constraint system



Proof scripts

```
lemma ForwardSecrecy:
  all-traces
  "∀ S sessKey x_sk y_pk #i #k.
         ((SecretClient( S, sessKey, x_sk, y_pk ) @ #i) ∧
          (K( sessKey ) @ #k)) ⇒
         (((∃ s_sk #j. (CompromiseLtk( S, s_sk ) @ #j) ∧ (
           (∃ x_pk #j. CompromiseEphemeralKey( x_sk, x_pk
           (∃ y_sk #j. CompromiseEphemeralKey( y_sk, y_pk )
edit lemma or delete lemma
simplify
solve( StateC( $C, ~x_sk ) ▶₀ #i )
  case Client_Init
  solve( !Pk( $S, pk(x) ) ▶₁ #i )
    case Generate_serverLtk
    solve( !KU( sign(<'g'^~x_sk, y_pk>, ~sk) ) @ #vk.2 )
      case Server_Init
      solve( !KU( h('g'^(~x_sk*~y_sk)) ) @ #vk.3 )
        case c_h
        solve( !KU( 'g'^(~x_sk*~y_sk) ) @ #vk.5 )
          case Client_Init
          solve( !KU( ~y_sk ) @ #vk.6 )
            case Compromise_EphemeralKey
            by contradiction /* from formulas */
          qed
        next
          case Server_Init
          solve( !KU( ~x_sk ) @ #vk.6 )
            case Compromise_EphemeralKey
            by contradiction /* from formulas */
          qed
        next
          case c_exp
          solve( !KU( ~x_sk ) @ #vk.8 )
            case Compromise_EphemeralKey
            by contradiction /* from formulas */
          qed
        qed
      qed
    next
```

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
s. autoprove (S. for all solutions) for all lemmas

Constraint system

last: none

formulas:

∃ S x_pk y_pk s_pk serverSign sessKey #i.
(ClientAccepts(S, x_pk, y_pk, s_pk, serverSign, sessKey) @ #i)
∧
(∀ #j.
(ServerAccepts(S, x_pk, y_pk, s_pk, serverSign, sessKey) @ #j)
⇒
((¬(#j < #i)) ∨
(∃ #t.
(ClientAccepts(S, x_pk, y_pk, s_pk, serverSign, sessKey) @ #t)
∧
¬(#t = #i)))) ∧
(∀ s_sk #j. (CompromiseLtk(S, s_sk) @ #j) ⇒ ⊥)

subterms:

equations:
subst:
conj:

But Tamarin can do much more...

- Modern models high double-digits number of rules, covering all modes of complex protocols in one go

Some examples:

- IETF TLS 1.3
- SPDM 1.2
- 5G-AKA
- EMV (Chip and pin)
- Noise protocol suite
- Apple iMessage PQ3

But Tamarin can do much more…

- Modern models high double-digits number of rules, covering all modes of complex protocols in one go

Some examples:

- IETF TLS 1.3
- SPDM 1.2
- 5G-AKA
- EMV (Chip and pin)
- Noise protocol suite
- Apple iMessage PQ3

Tamarin found 18-messageattack on draft 10+ that breaks post-handshake authentication

But Tamarin can do much more…

- Modern models high double-digits number of rules, covering all modes of complex protocols in one go

Some examples:

- IETF TLS 1.3
- SPDM 1.2
- 5G-AKA
- EMV (Chip and pin)
- Noise protocol suite
- Apple iMessage PQ3

Tamarin found 18-messageattack on draft 10+ that breaks post-handshake authentication

Tamarin found cross-protocol attack that breaks psk authenticaton

But Tamarin can do much more…

- Modern models high double-digits number of rules, covering all modes of complex protocols in one go

Some examples:

- IETF TLS 1.3
- SPDM 1.2
- 5G-AKA
- EMV (Chip and pin)
- Noise protocol suite
- Apple iMessage PQ3

Tamarin found 18-messageattack on draft 10+ that breaks post-handshake authentication

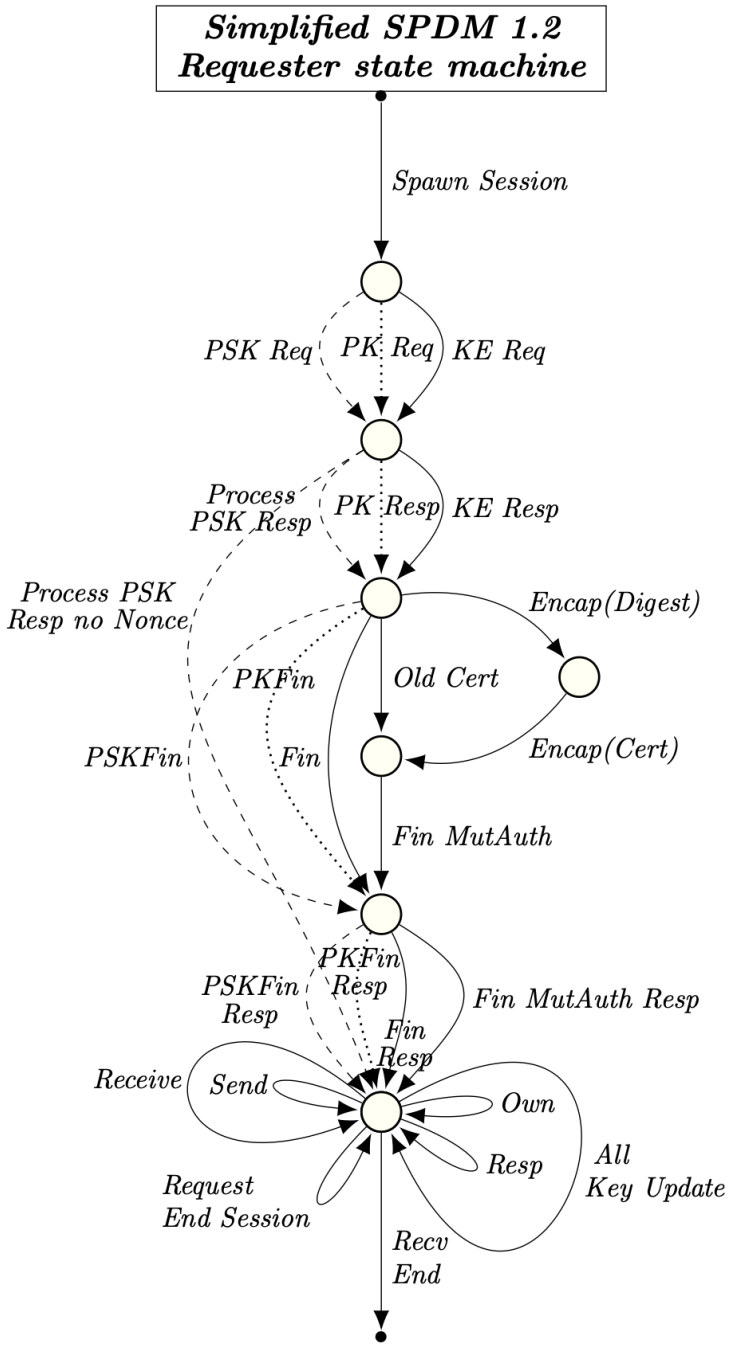
Tamarin found cross-protocol attack that breaks psk authenticaton

We can automate “the strongest property” that holds for each noise pattern

Complexity example: SPDM 1.2

- Simplified picture:
- Actual model nearly 70 rules

- Tamarin finds a cross-protocol attack that breaks PSK authentication
  - Works on reference implementation and Intel's rust implementation
  - CVE with score 9.0 (critical)
- Prove fixed version



Recent developments: Blurring the lines

- We are moving beyond traditional symbolic/Dolev-Yao features

A good starting point is:
https://ia.cr/2019/779

Recent developments: Blurring the lines

- We are moving beyond traditional symbolic/Dolev-Yao features

- *"Perfect cryptography"*
  and
  "one symbolic model for each cryptographic primitive"

- *"Symbolic models explicitly specify the possible adversary operations"*

A good starting point is:
https://ia.cr/2019/779

Recent developments: Blurring the lines

- We are moving beyond traditional symbolic/Dolev-Yao features

- *"Perfect cryptography"*
  and
  "one symbolic model for each cryptographic primitive"

- New: a range of symbolic models for
  - Signatures
  - AEADs
  - Hash functions
  - DH/EC
  - KEMs

- *"Symbolic models explicitly specify the possible adversary operations"*

A good starting point is:
https://ia.cr/2019/779

Recent developments: Blurring the lines

- We are moving beyond traditional symbolic/Dolev-Yao features

- *"Perfect cryptography"*
  and
  "one symbolic model for each cryptographic primitive"

- New: a range of symbolic models for
  - Signatures
  - AEADs
  - Hash functions
  - DH/EC
  - KEMs

- *"Symbolic models explicitly specify the possible adversary operations"*

- We can avoid this by trace restrictions:
  Instead of explicitly specifying allowed transitions, generalize and restrict

A good starting point is:
https://ia.cr/2019/779

Reality, models, and gaps

Reality

Reality, models, and gaps

Reality

Computational

Reality, models, and gaps

Reality

Computational

Symbolic

Reality, models, and gaps

Reality

Computational

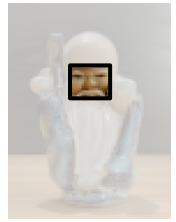
Symbolic

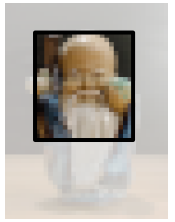
“SoK: Computer-Aided Cryptography”, BBBBCLP IEEE S&P 2021. IACR eprint 2019/1393

The need to create breathing room

Multiple approaches for provable security

In practice, the results are incomparable

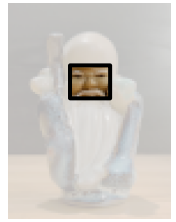
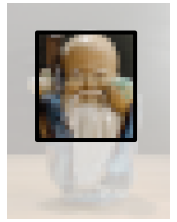




The need to create breathing room

Multiple approaches for provable security

In practice, the results are incomparable

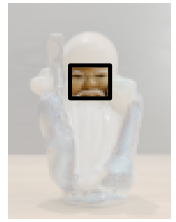
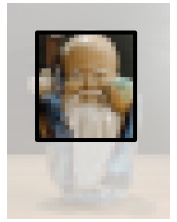




- *We should not fight this but embrace it*
  - *Study* multiple approaches, don't discard
  - Need space and time for new ideas to come to fruition

The need to create breathing room

Multiple approaches for provable security

In practice, the results are incomparable

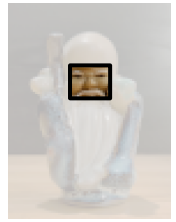
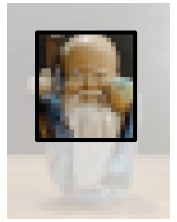




- *We should not fight this but embrace it*
  - *Study* multiple approaches, don't discard
  - Need space and time for new ideas to come to fruition
    - **Reviewing**
      - "you didn't do everything"
      - "use other method instead"

The need to create breathing room

Multiple approaches for provable security

In practice, the results are incomparable

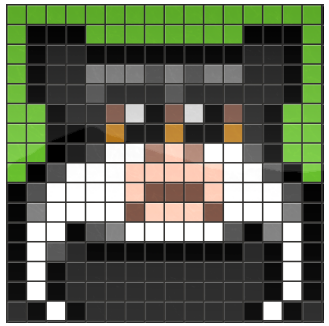
- *We should not fight this but embrace it*
  - *Study* multiple approaches, don't discard
  - Need space and time for new ideas to come to fruition
    - **Reviewing**
      - "you didn't do everything"
      - "use other method instead"
    - **Writing**:
      - Framing of results:
      - "formally *verified*" "*provably secure*" unhelpful

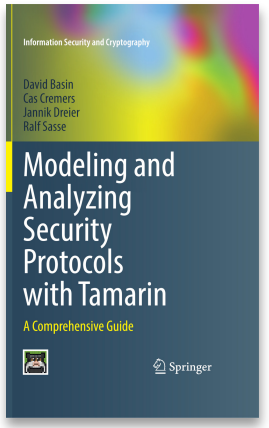
Conclusions

Conclusions: Tamarin

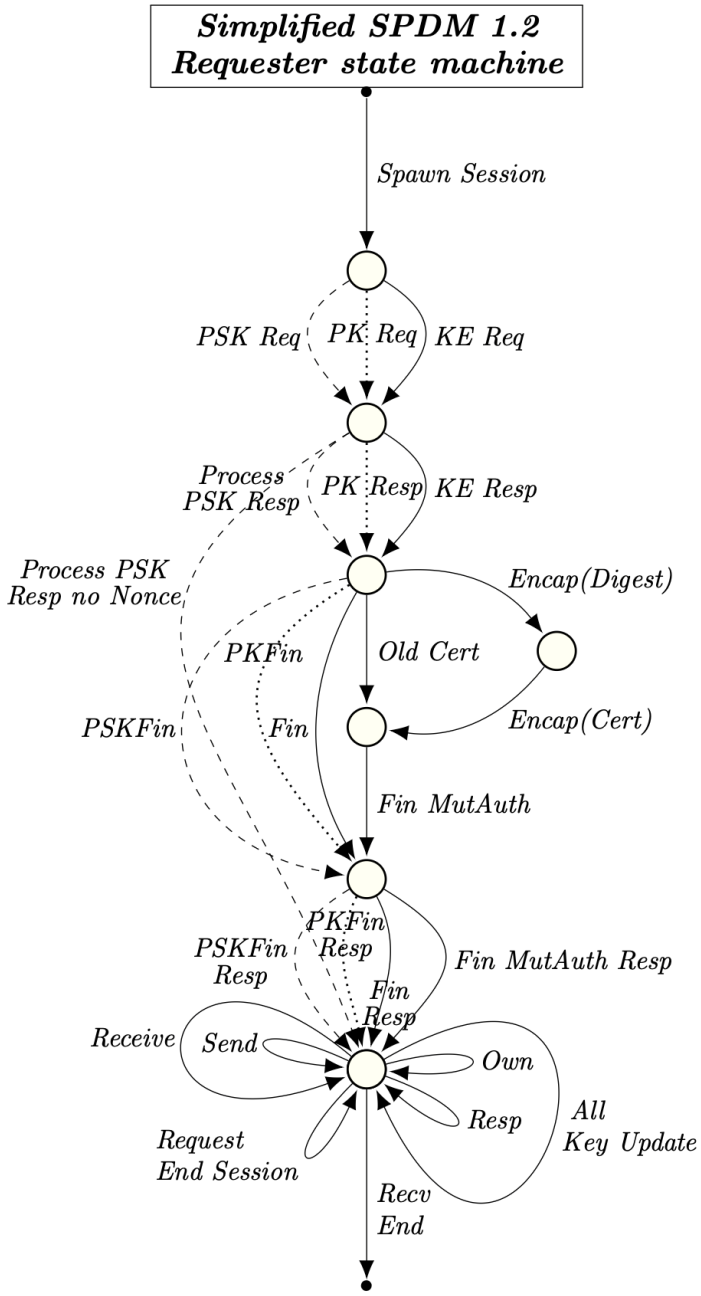
- Tamarin is a mature tool
- capable of dealing with highly complex models
- Offers state-of-the art features
- Proofs or attack-finding
- GUI enables guiding/inspecting partial proofs
- Active development, user community, tutorials, book (soon: lecture slides based on book)

tamarin-prover.com





- IETF TLS 1.3
- SPDM 1.2
- 5G-AKA
- EMV (Chip and pin)
- Noise protocol suite
- Apple iMessage PQ3



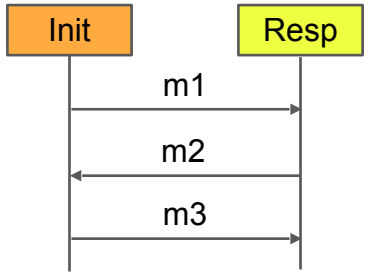
Cas Cremers – cremers@cispa.de

CISPA
HELMHOLTZ CENTER FOR INFORMATION SECURITY

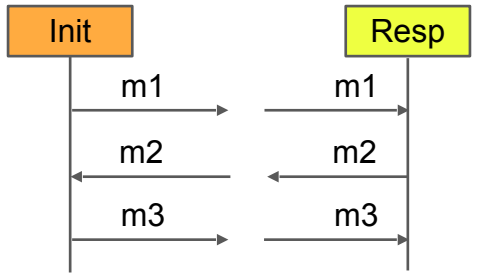
Backup slides

Tamarin use in industry

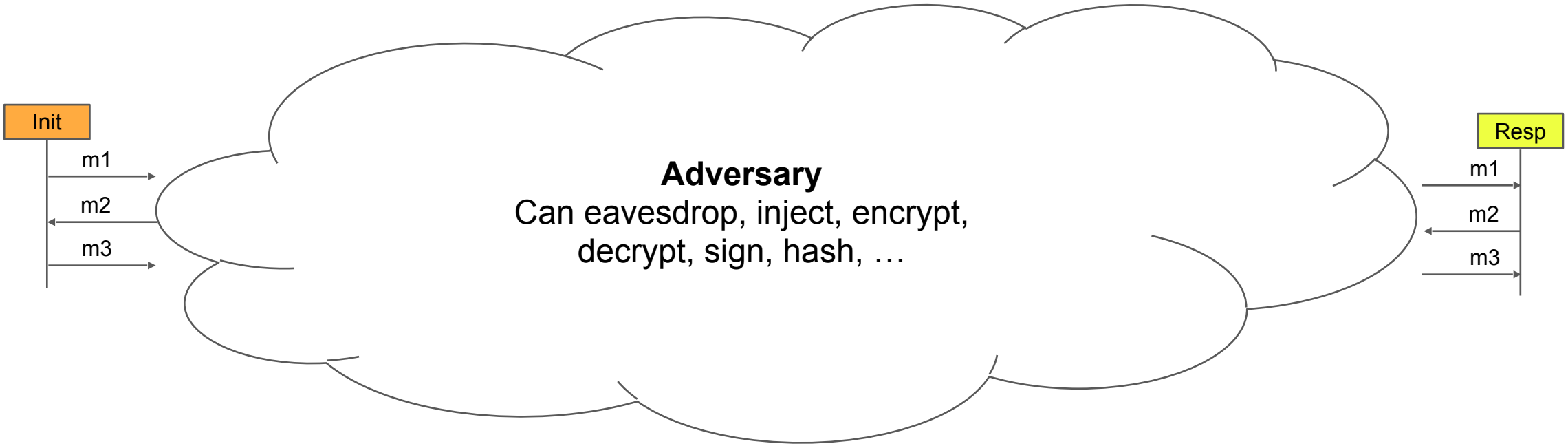
Protocols and threat models



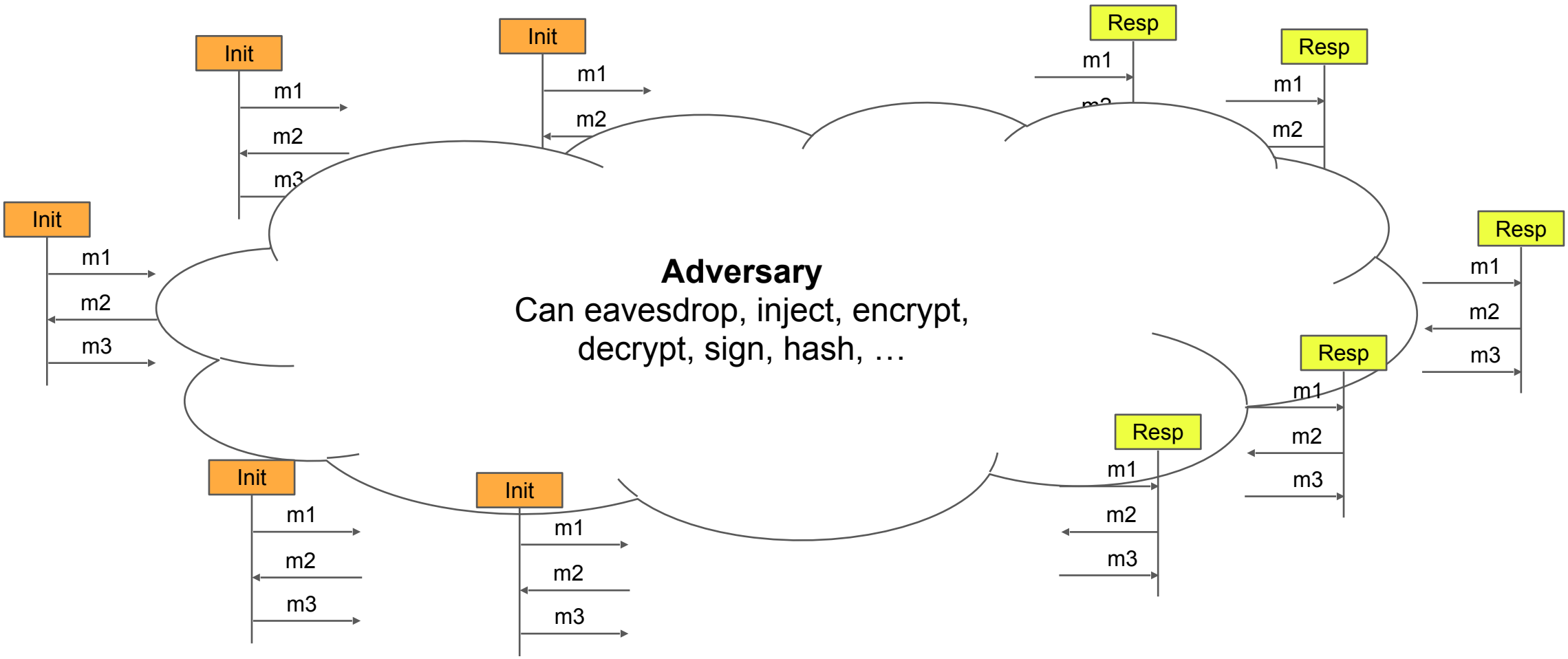
Protocols and threat models



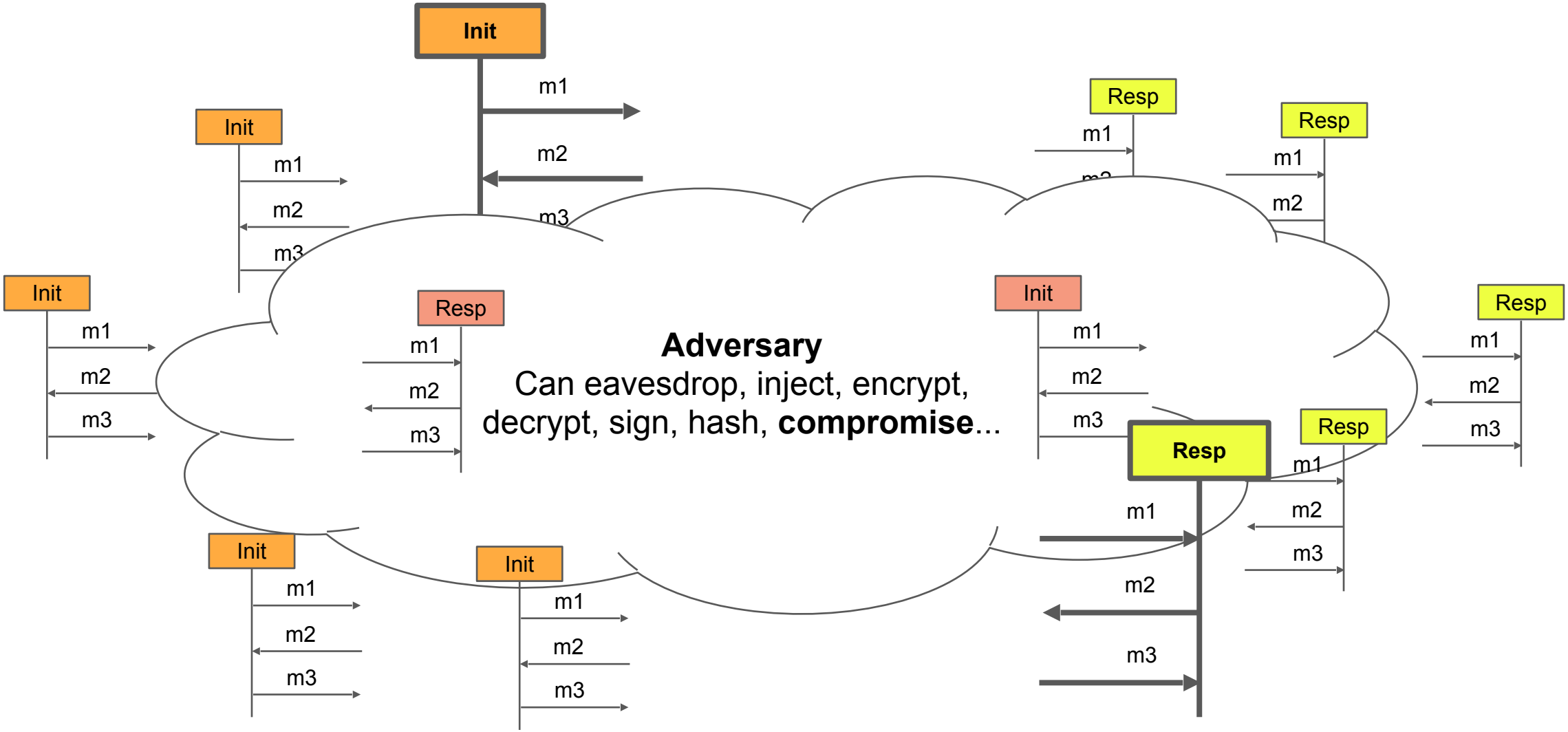
Protocols and threat models



Protocols and threat models



Protocols and threat models



SPDM: Security Protocol and Data Model

- Industry support behind this protocol:
  - DMTF- Distributed Management Task Force
  - Supported by other standards groups

SPDM: Security Protocol and Data Model

- Industry support behind this protocol:
  - DMTF- Distributed Management Task Force
  - Supported by other standards groups
- Two party protocol for secure communication over the wire:
  - authentication of hardware identities
  - measurement for firmware identities
  - session key exchange protocols to enable confidentiality & integrity

SPDM

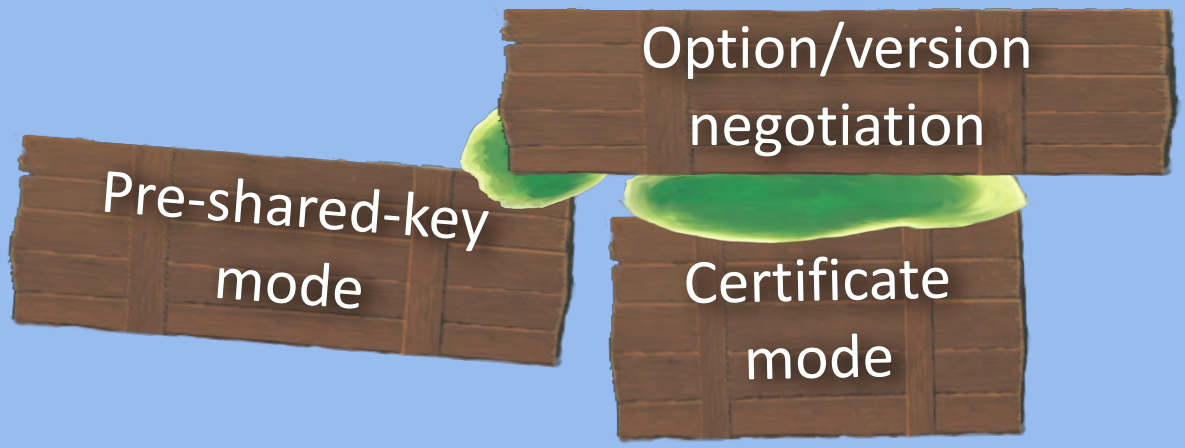
- Secure Protocol and Data Model

SPDM

- Secure Protocol and Data Model

SPDM

- Secure Protocol and Data Model

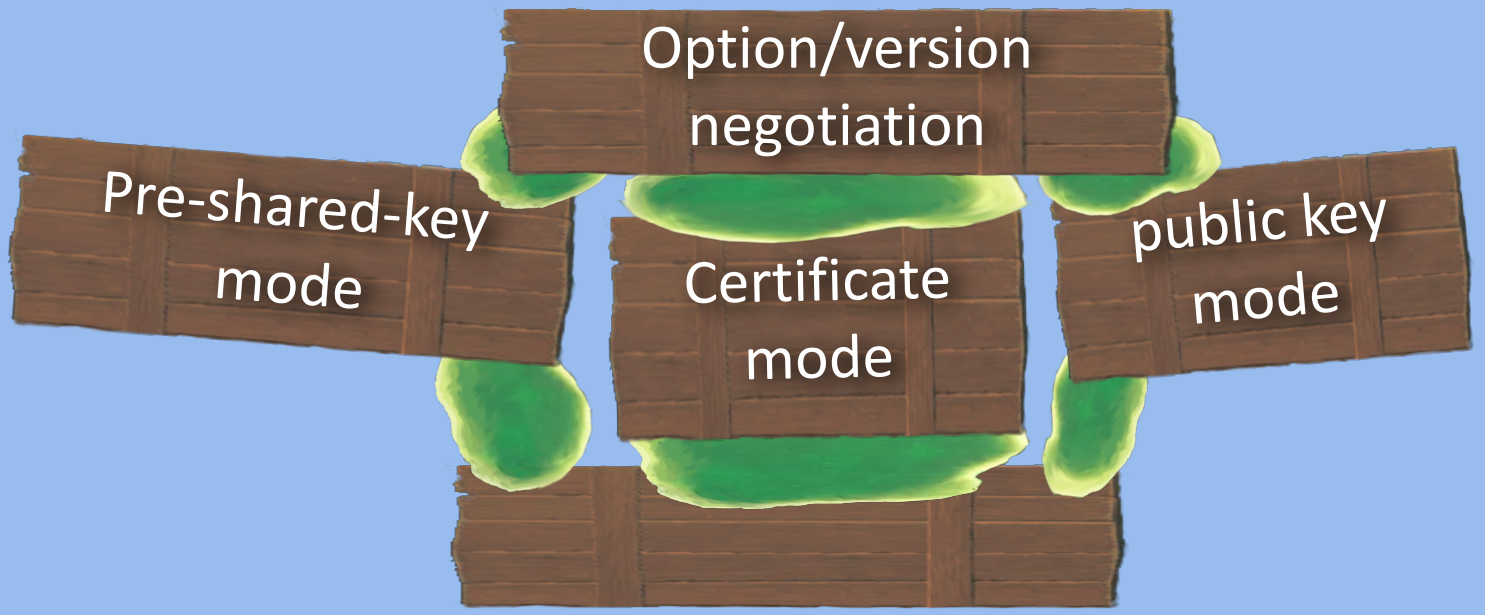


SPDM

- Secure Protocol and Data Model

SPDM

- Secure Protocol and Data Model



SPDM

- Secure Protocol and Data Model

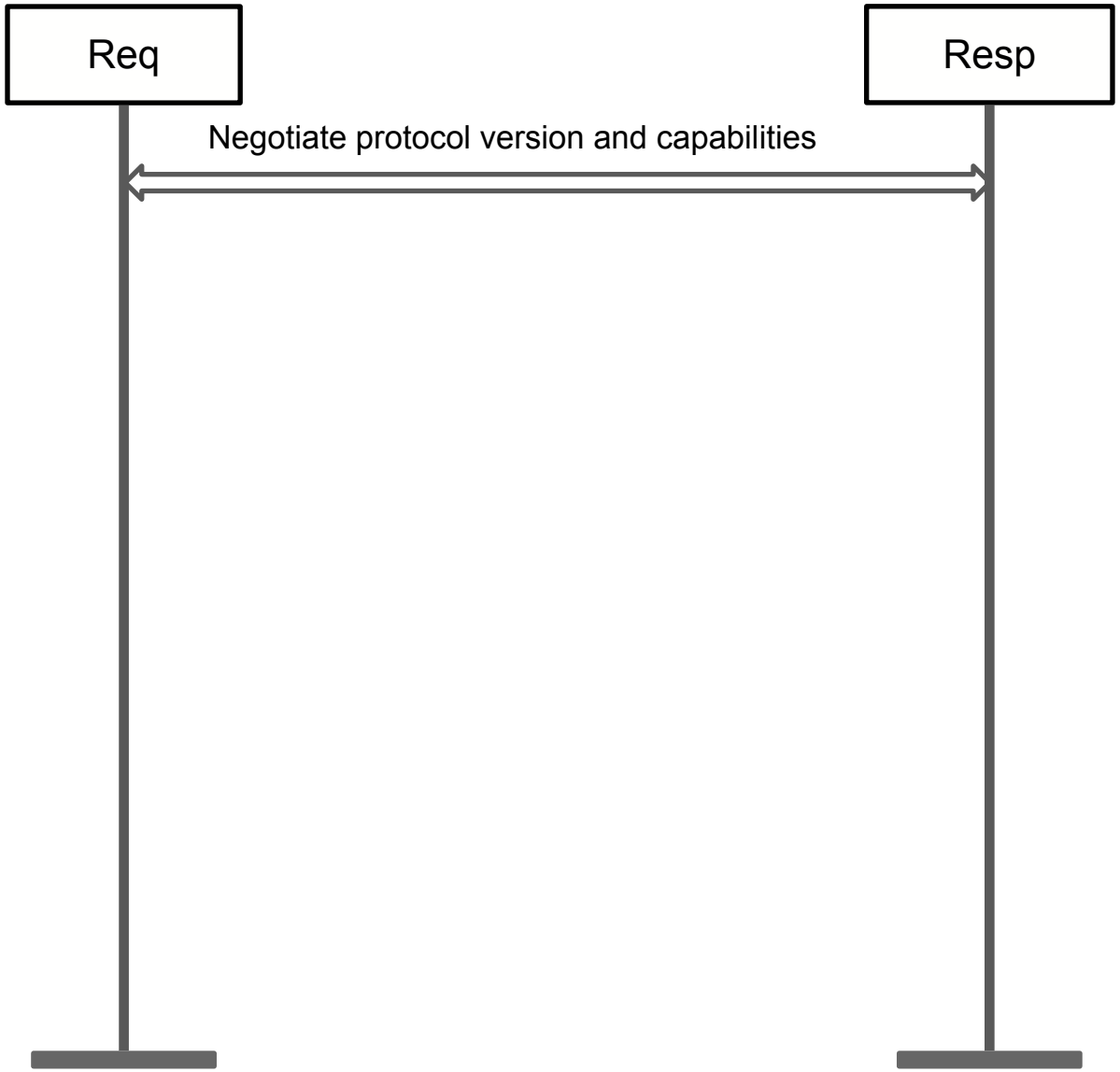
SPDM

- Secure Protocol and Data Model

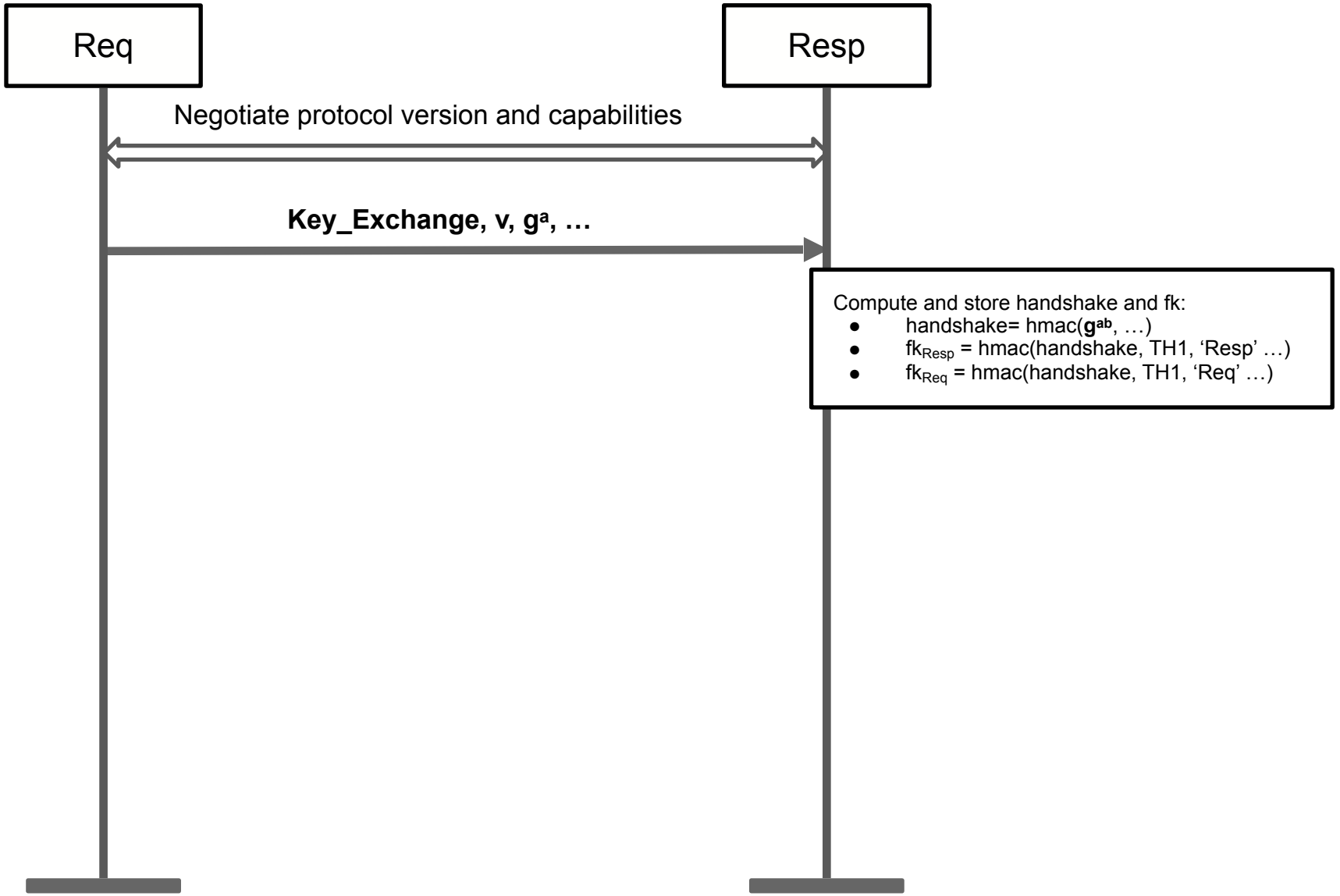
SPDM

- Secure Protocol and Data Model

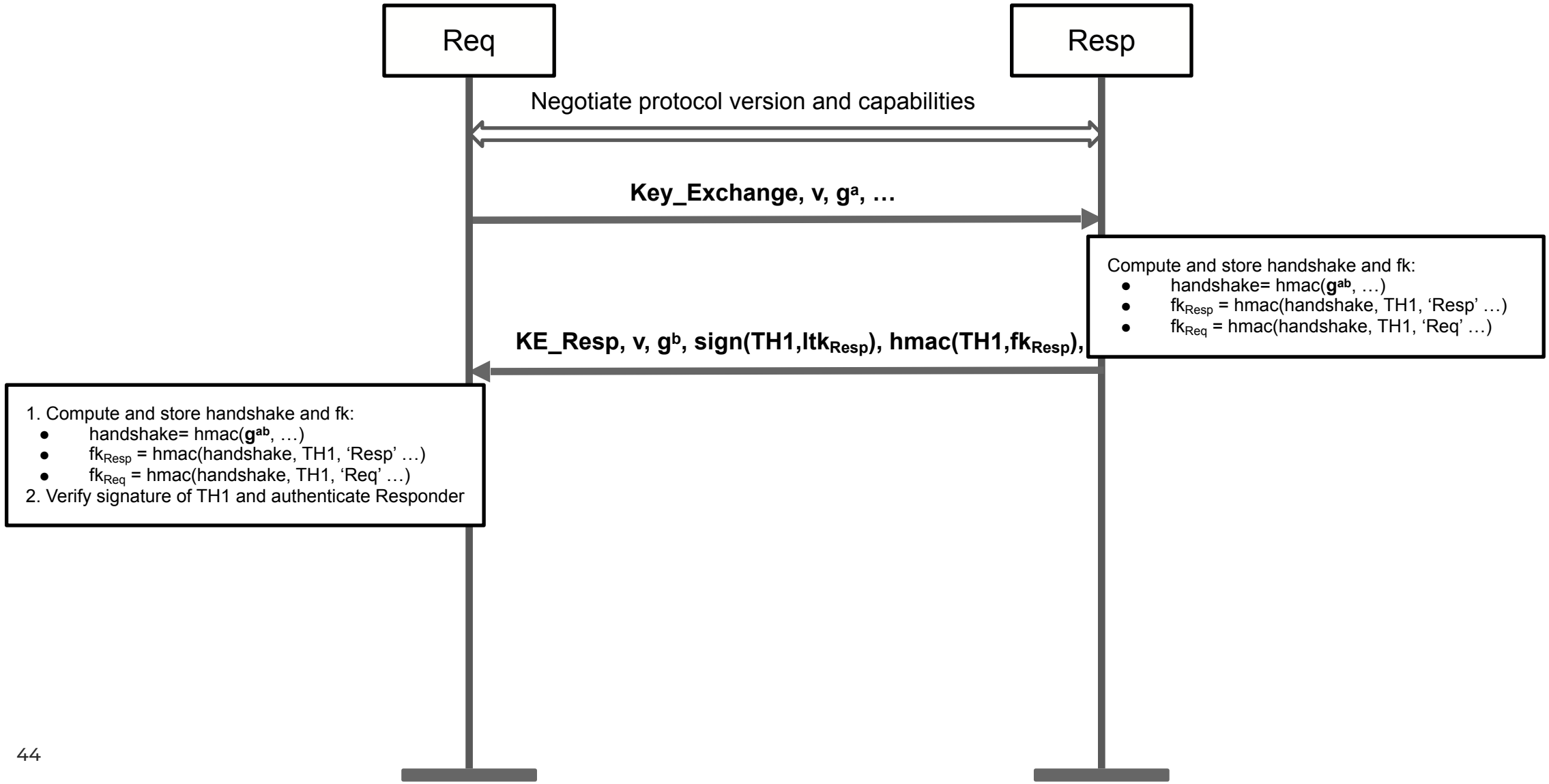
Key Exchange with Certificates



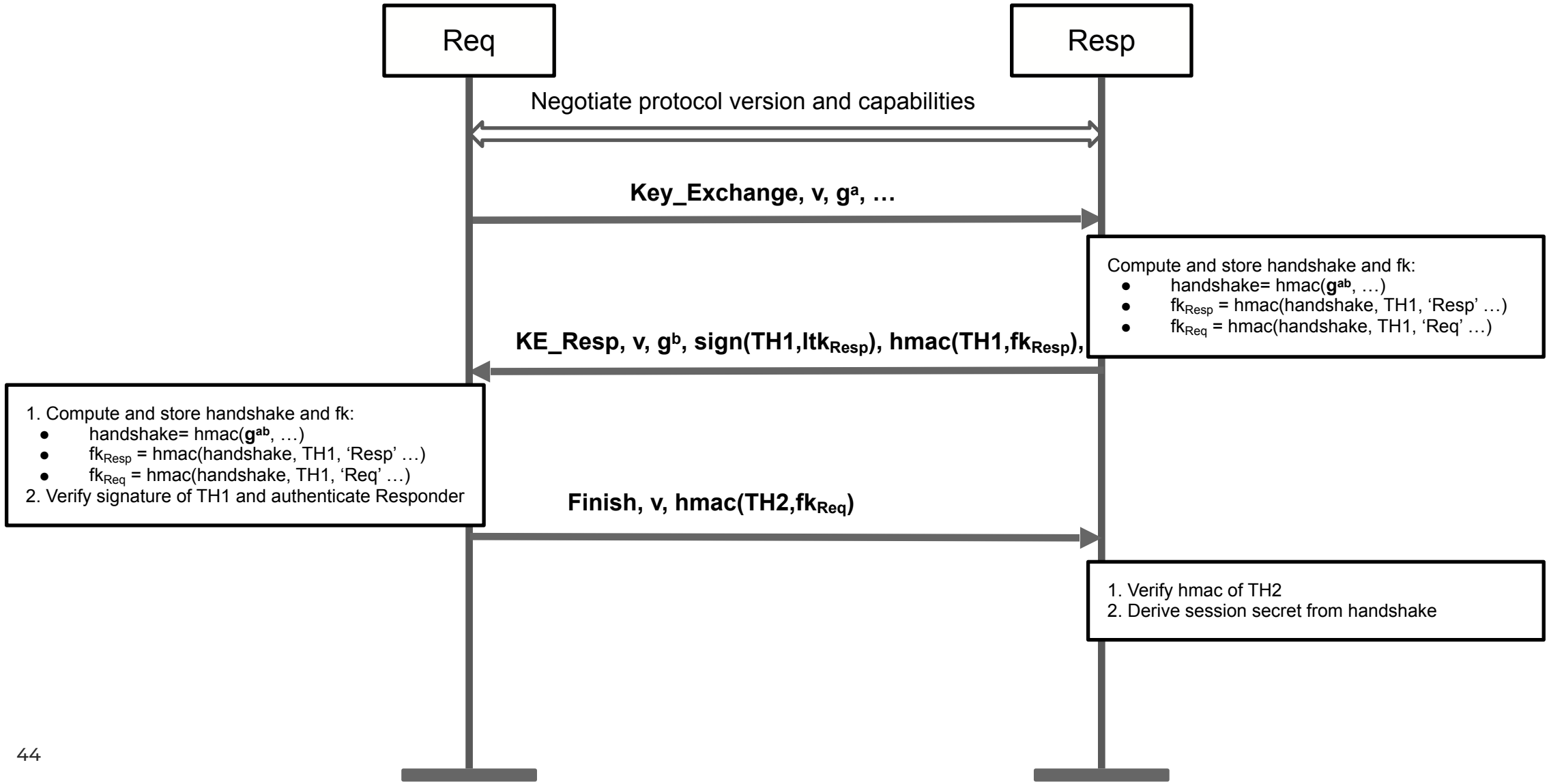
Key Exchange with Certificates



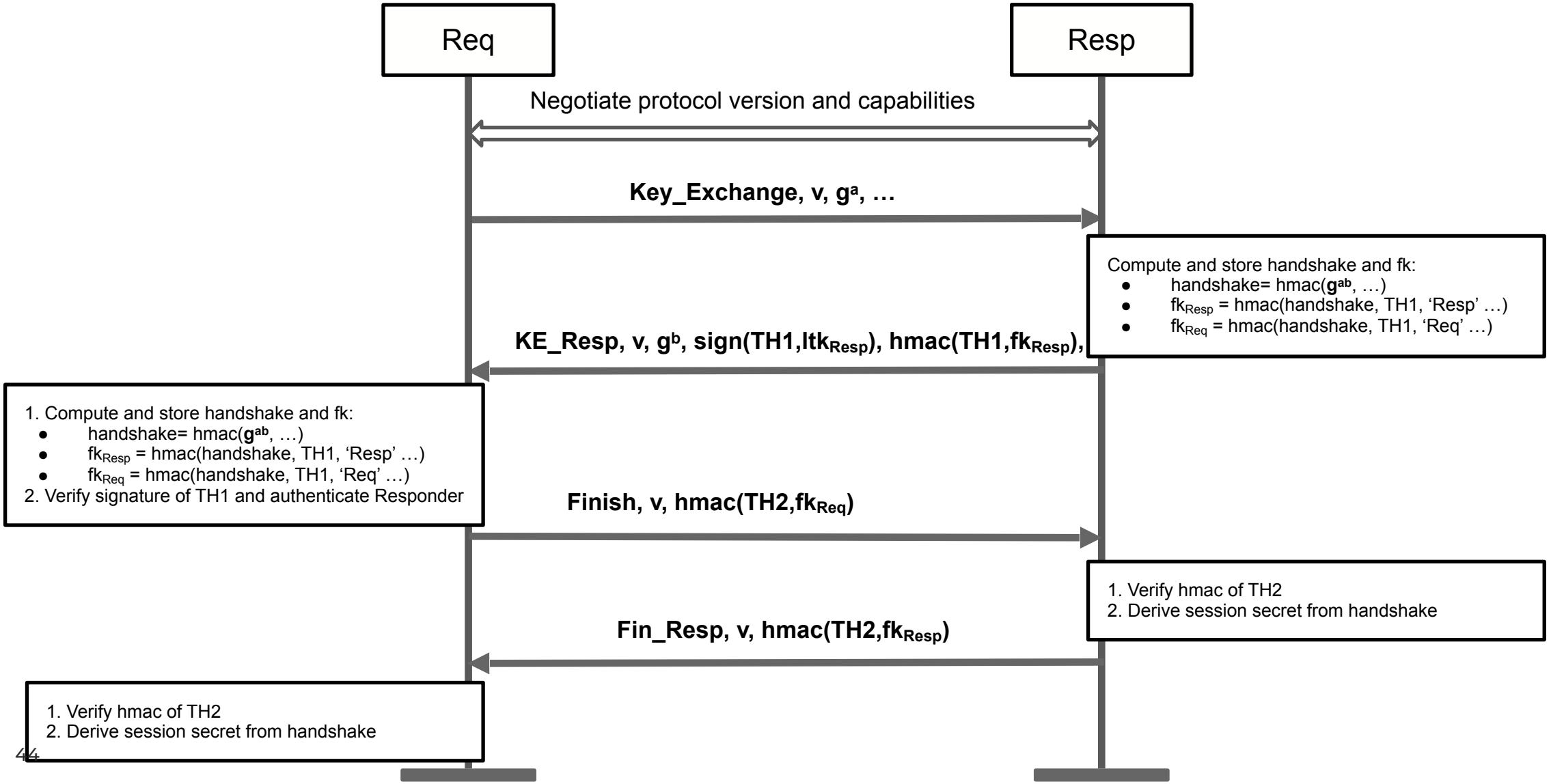
Key Exchange with Certificates



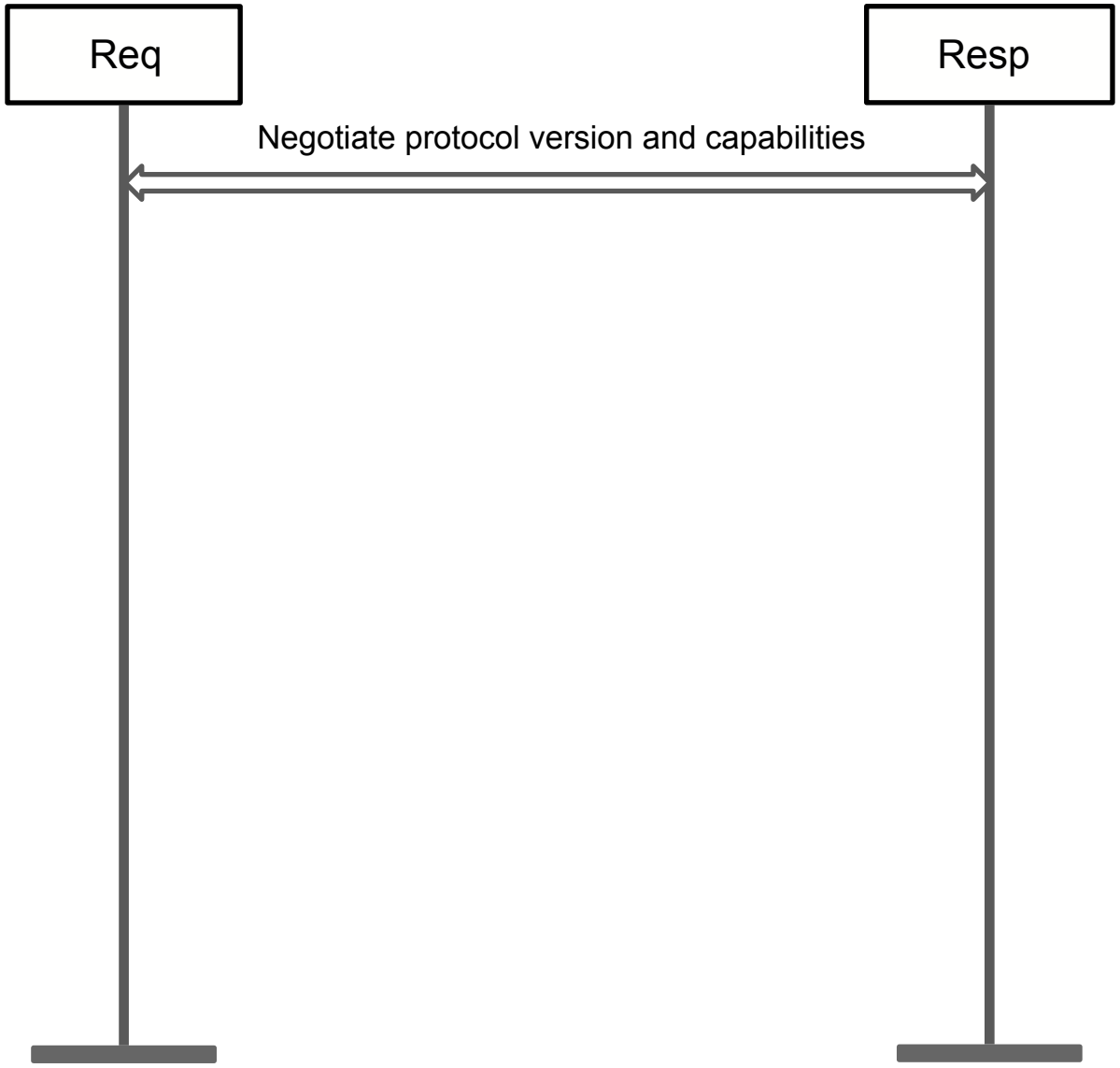
Key Exchange with Certificates



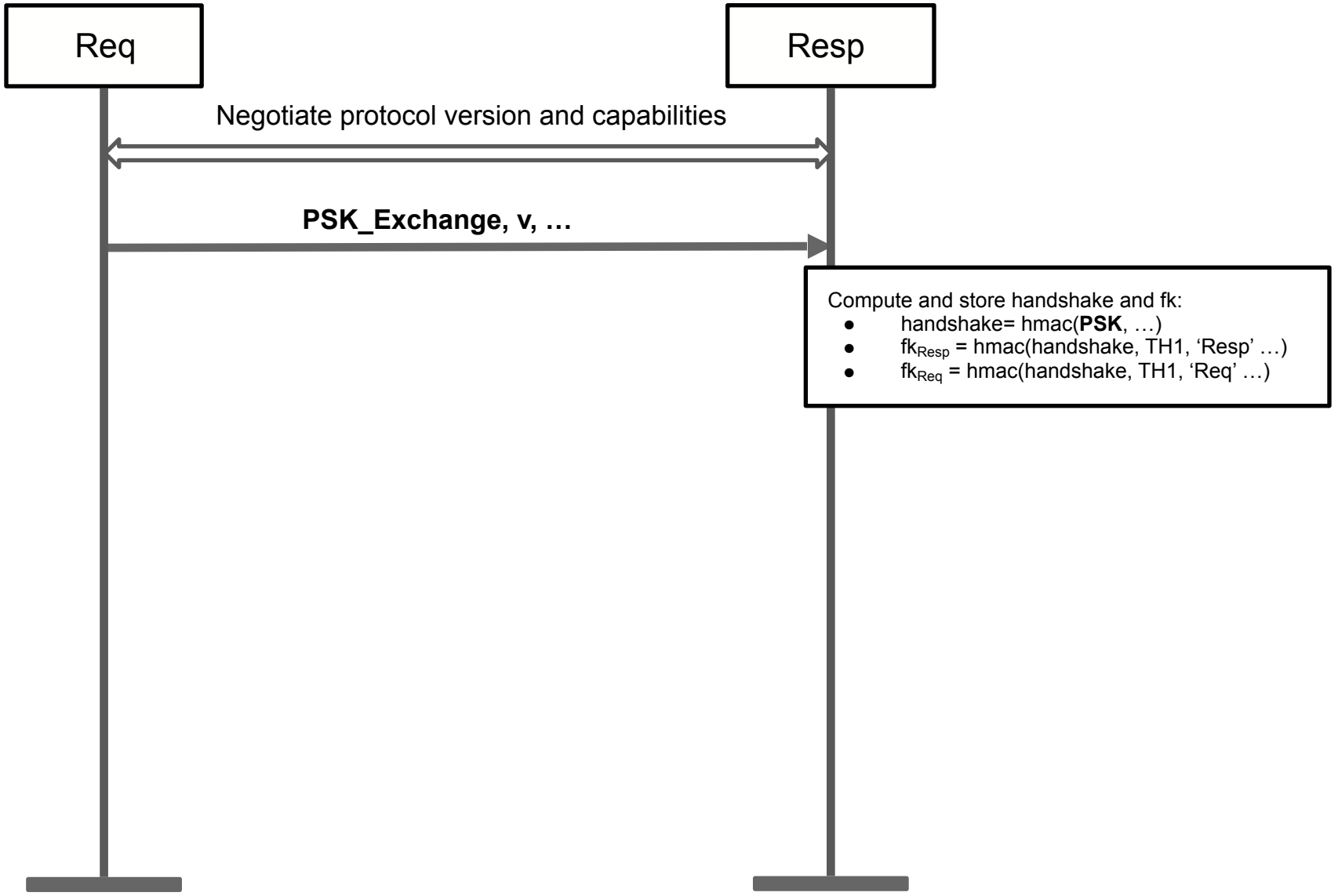
Key Exchange with Certificates



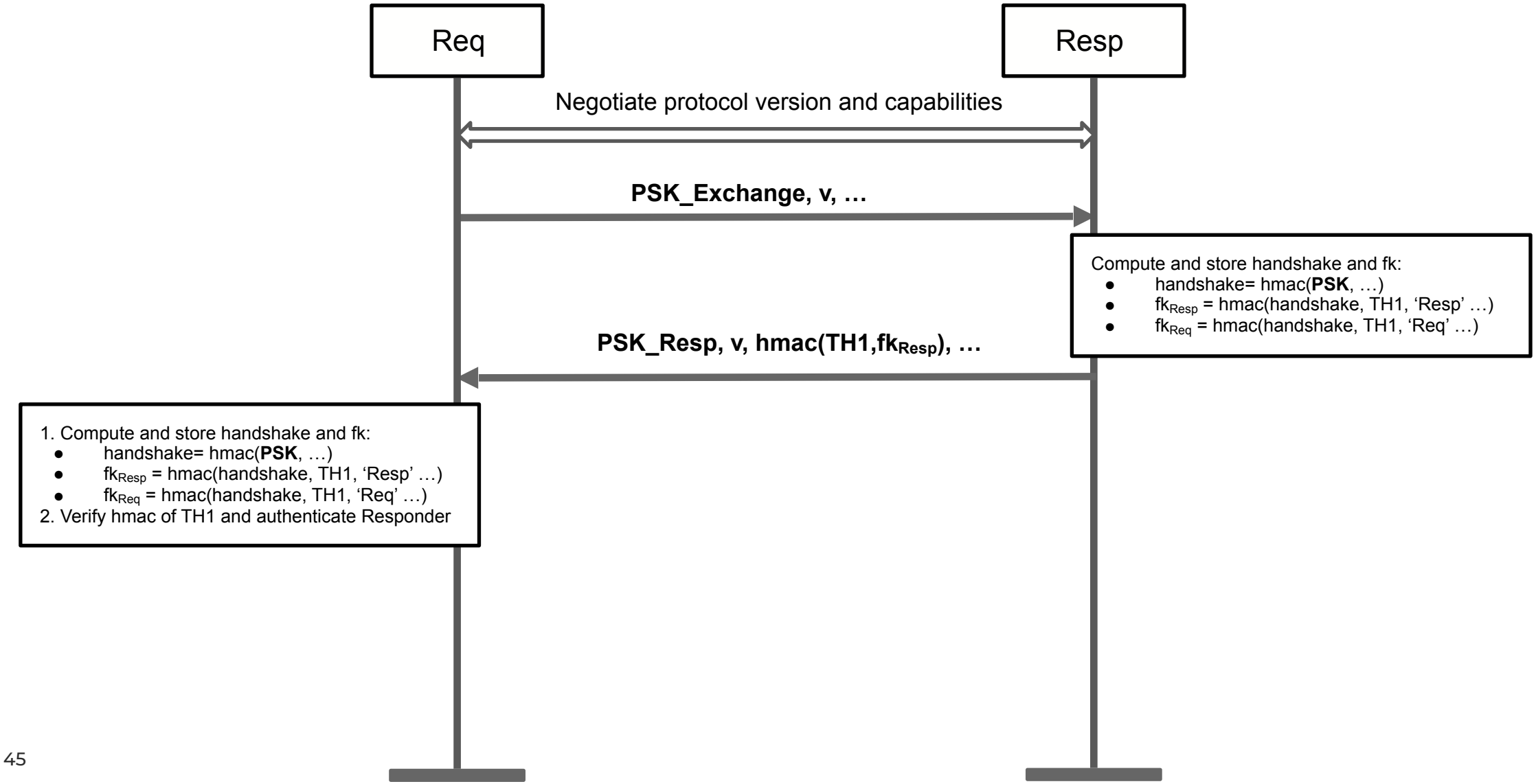
Key Exchange with Preshared Key PSK



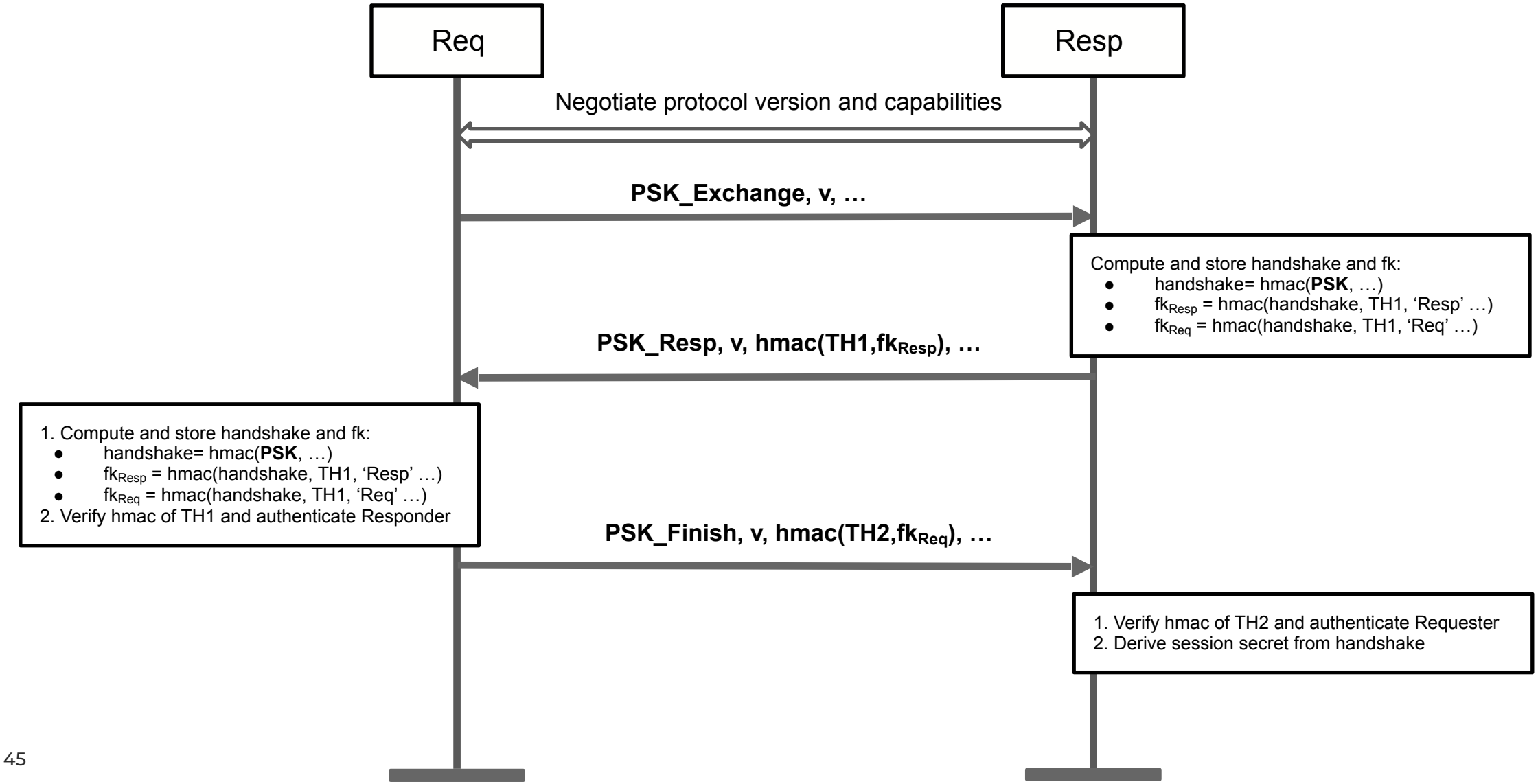
Key Exchange with Preshared Key PSK



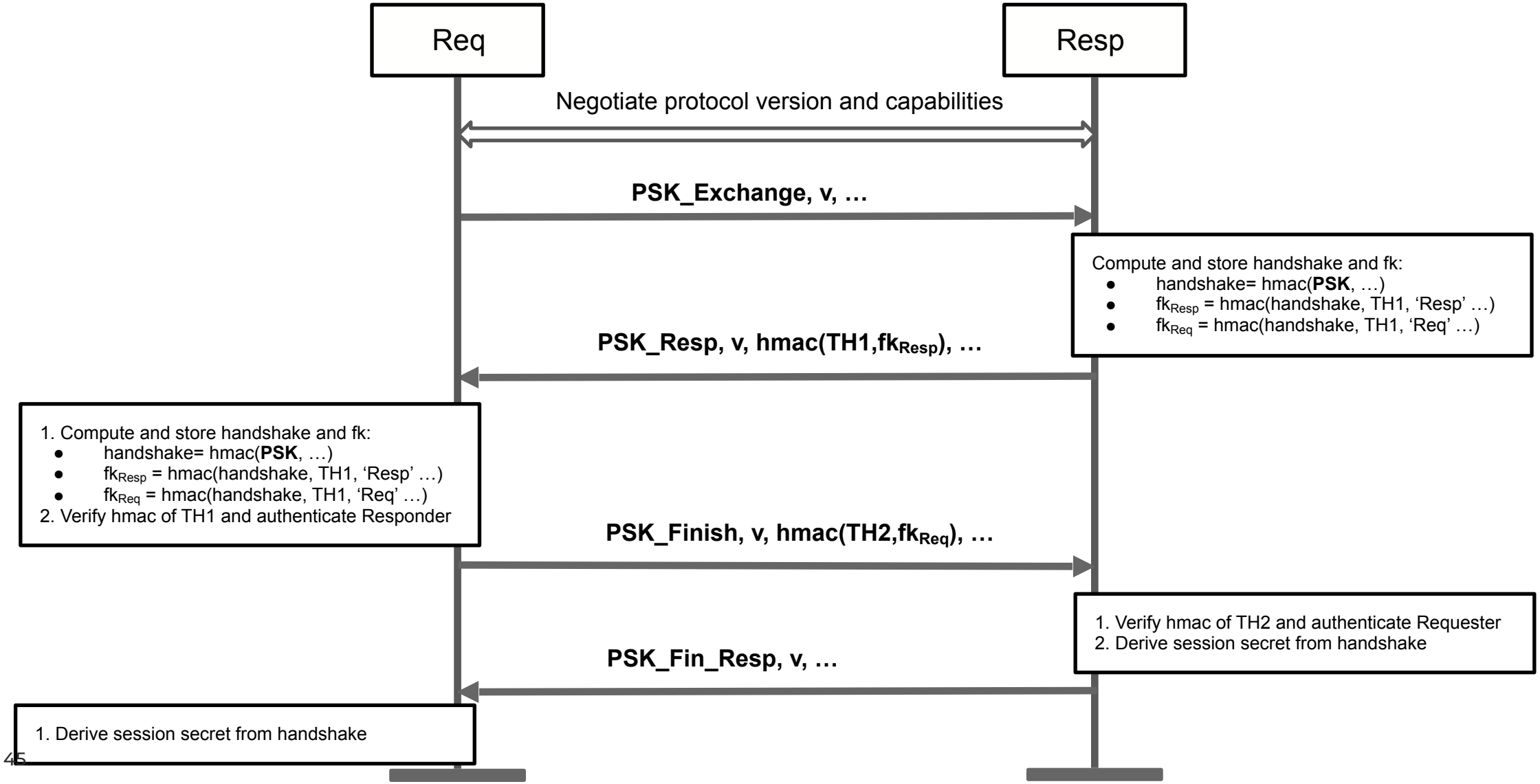
Key Exchange with Preshared Key PSK



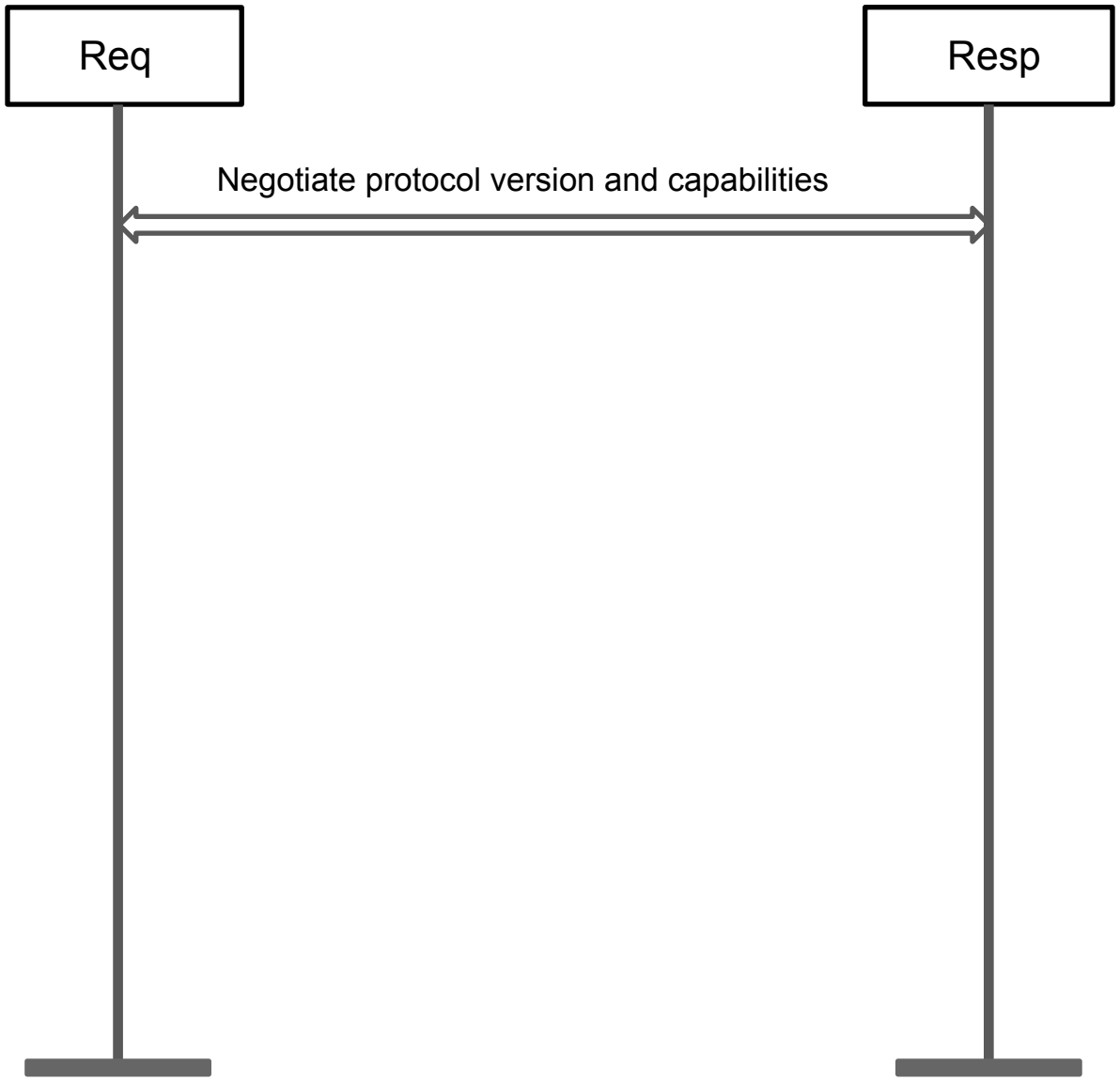
Key Exchange with Preshared Key PSK



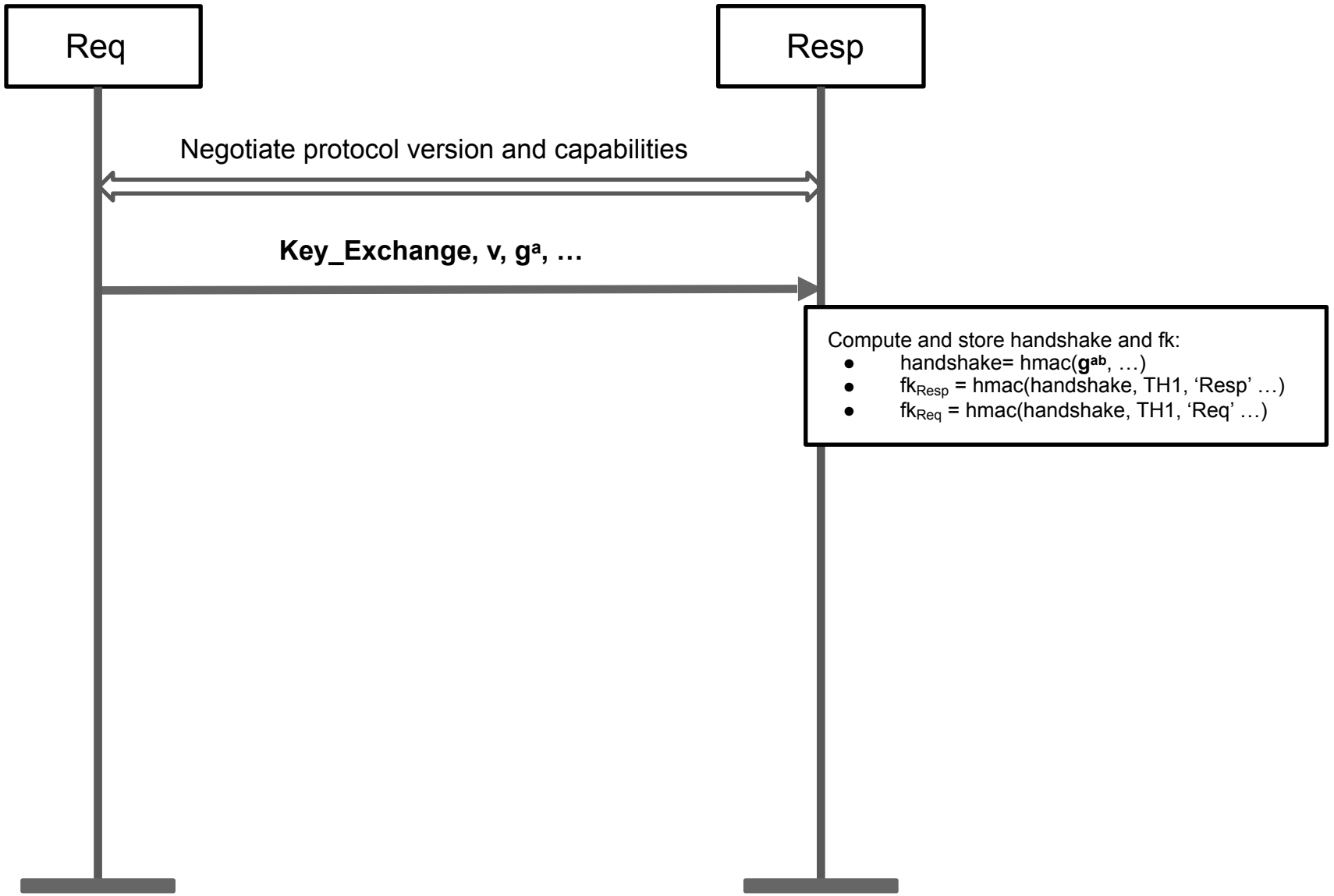
Key Exchange with Preshared Key PSK



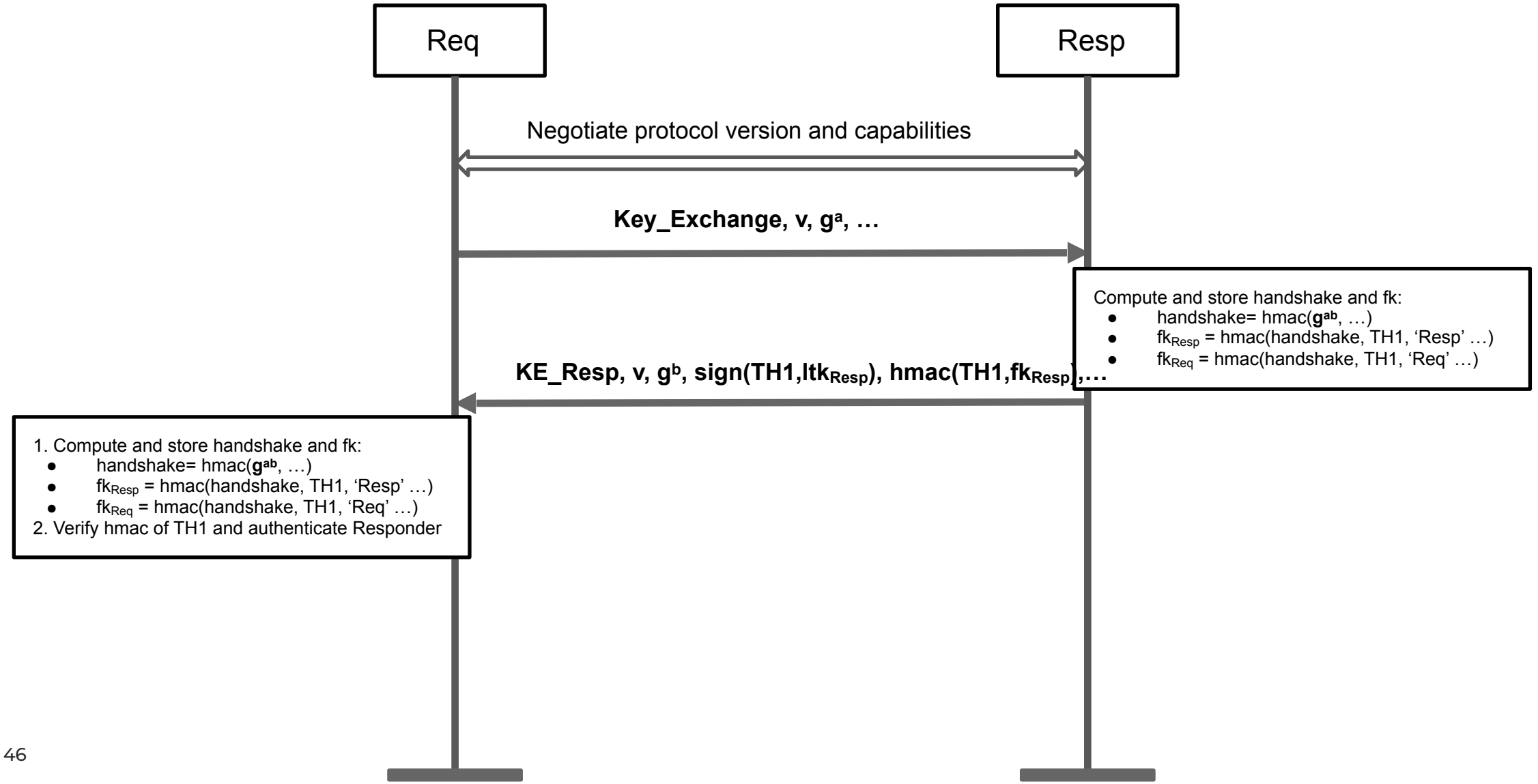
Mode Switch Attack (found by Tamarin)



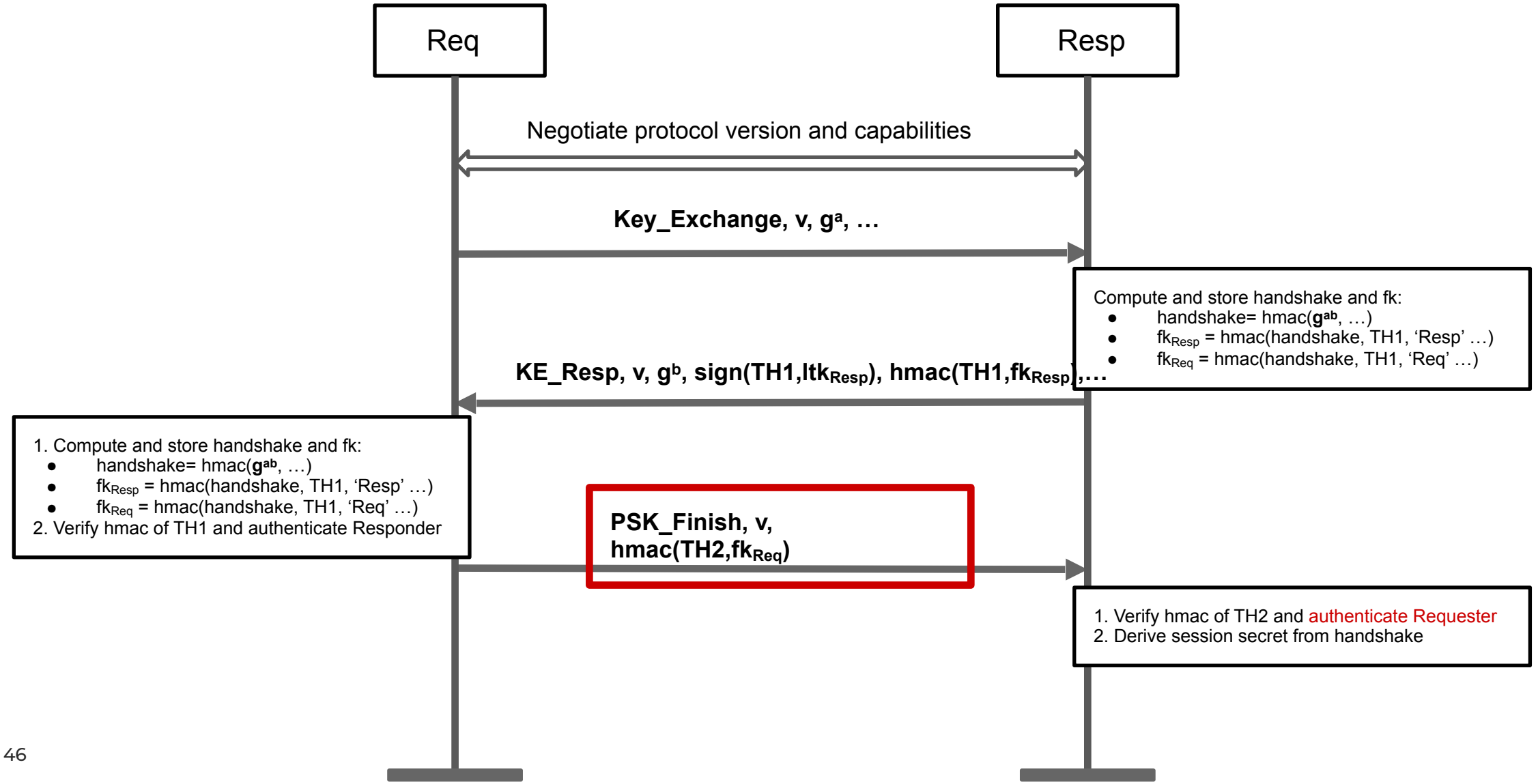
Mode Switch Attack (found by Tamarin)



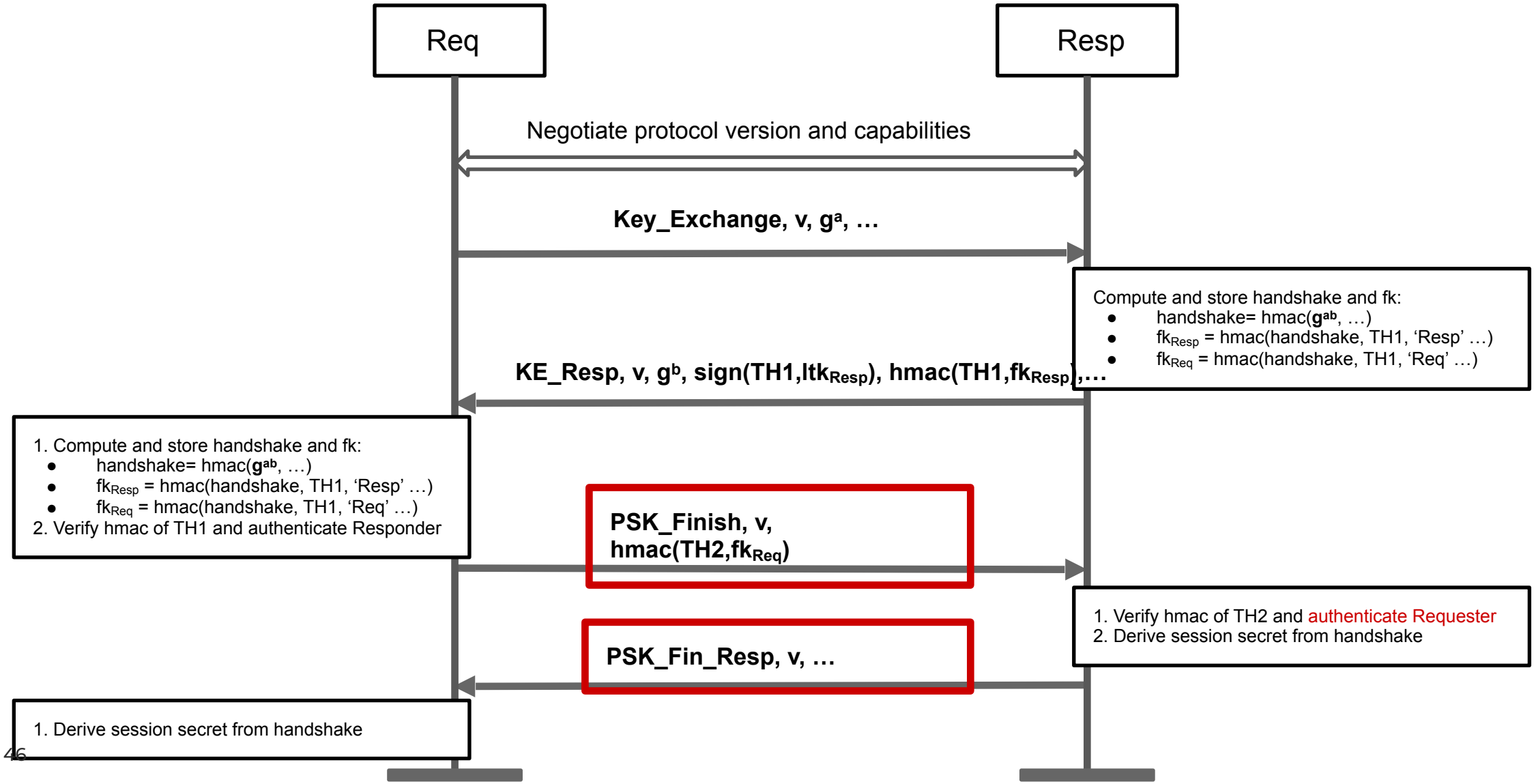
Mode Switch Attack (found by Tamarin)



Mode Switch Attack (found by Tamarin)



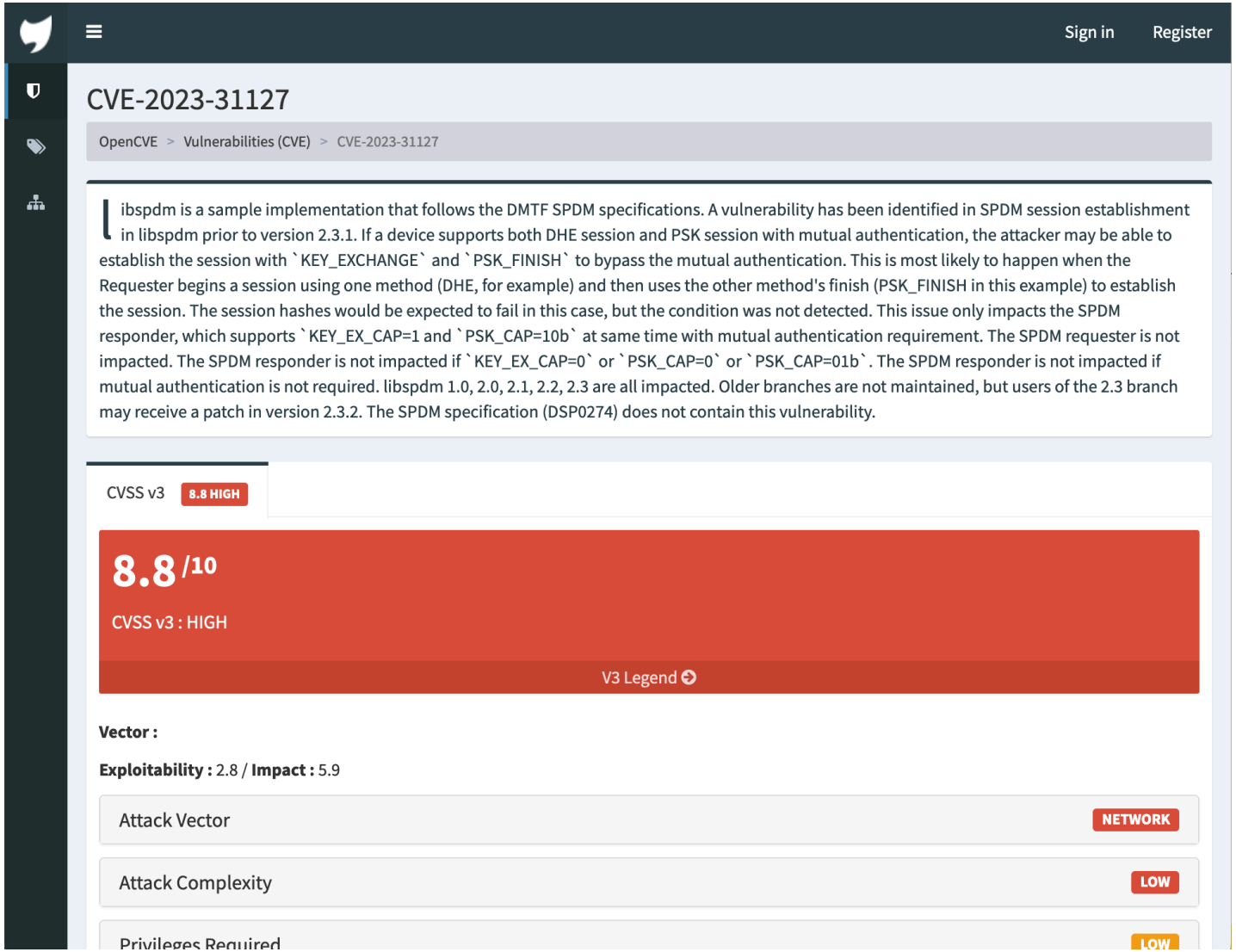
Mode Switch Attack (found by Tamarin)



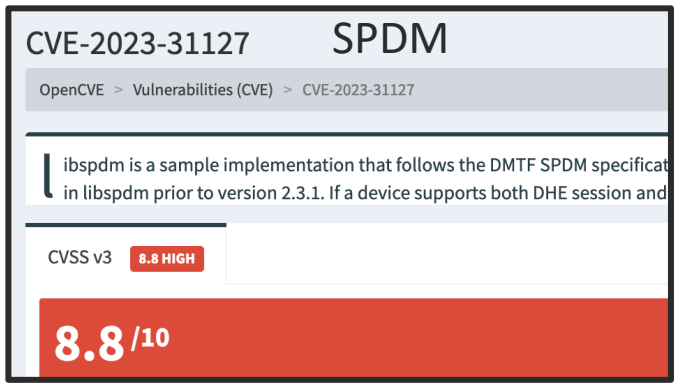
SPDM attack on PSK mode

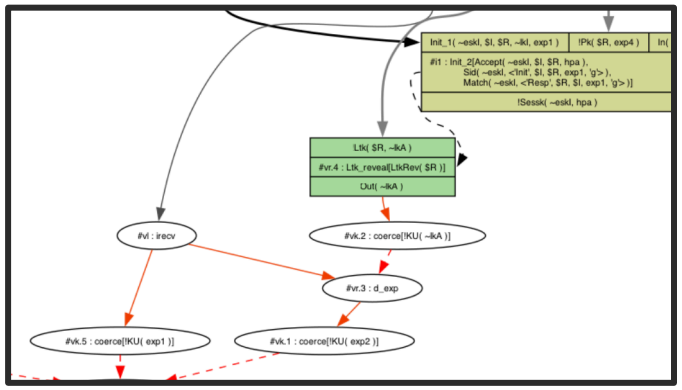
- Attack allows network attacker, without any knowledge, to get responder to accept in PSK mode
  - Breaks mutual authentication
  - Attacker learns key
- Automatically found by the Tamarin prover
- CVE for reference implementation
- Our fixes now in reference C implementation, Intel's Rust implementation, and the new standard

SPDM attack : outcome



Conclusions: the Tamarin Prover





Fine-grained models of cryptography

E.g. post-quantum KEMs

tamarin-prover.com

cremers@cispa.de

Open source on github

Mailing list

Manual & tutorials

Upcoming book

Examples

- Tamarin has been applied to a wide range of protocol problems
- Library of small examples
- Several larger case studies

- IETF TLS 1.3
- SPDM 1.2
- 5G-AKA
- Noise protocol suite
- Apple iMessage with PQ3
  https://security.apple.com/blog/imessage-pq3/

The Real End