Modeling in Tamarin

Aurora Naska

Email: aurora.naska@cispa.de

Slides designed by Cas Cremers, David Basin, Jannik Dreier, and Ralf Sasse

Sources:
Tamarin picture used with chicken hat by Brocken Inaglory
All other Tamarin photographs by Martin Dehnel-Wild
Other photos, graphics, and chicken hats by Cas Cremers

Team



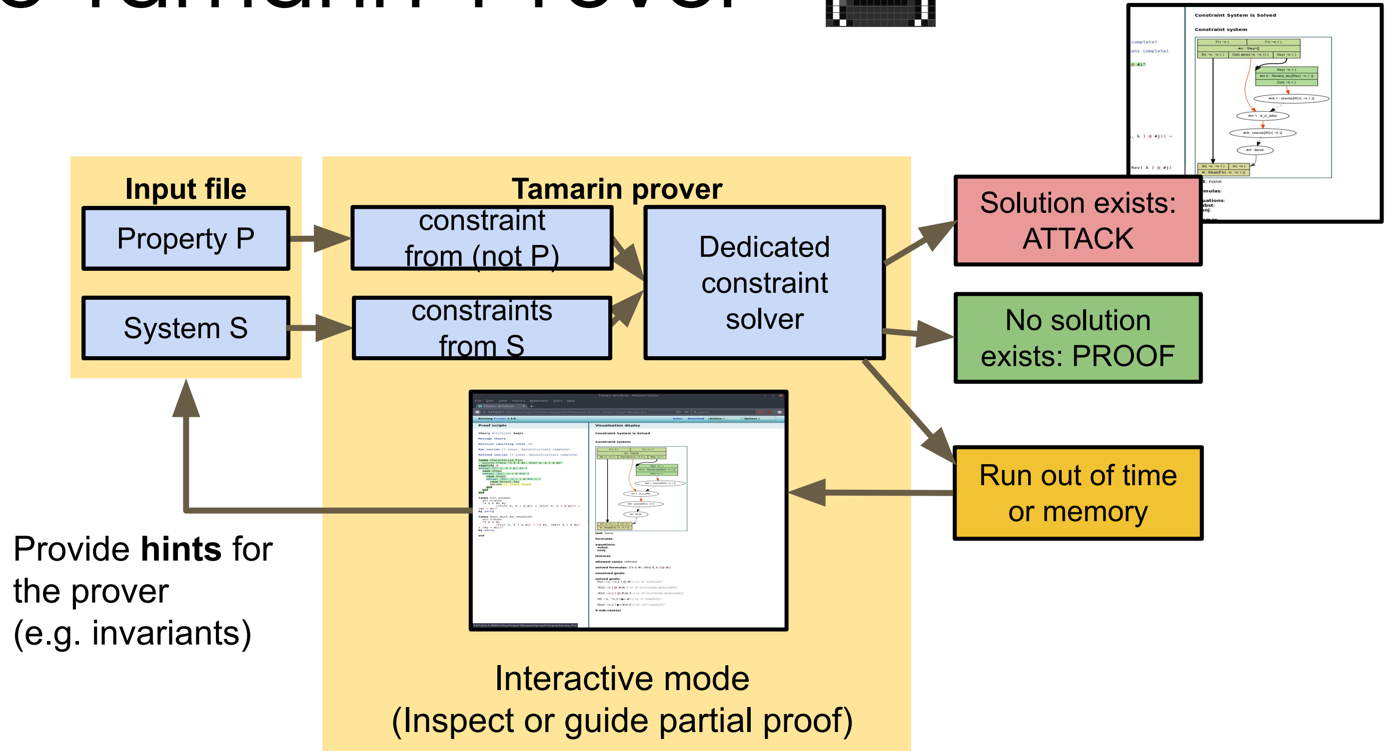


Alexander Dax Sofia Giampietro Xenia Hofmeier Niklas Medinger Aurora Naska

What to expect today?

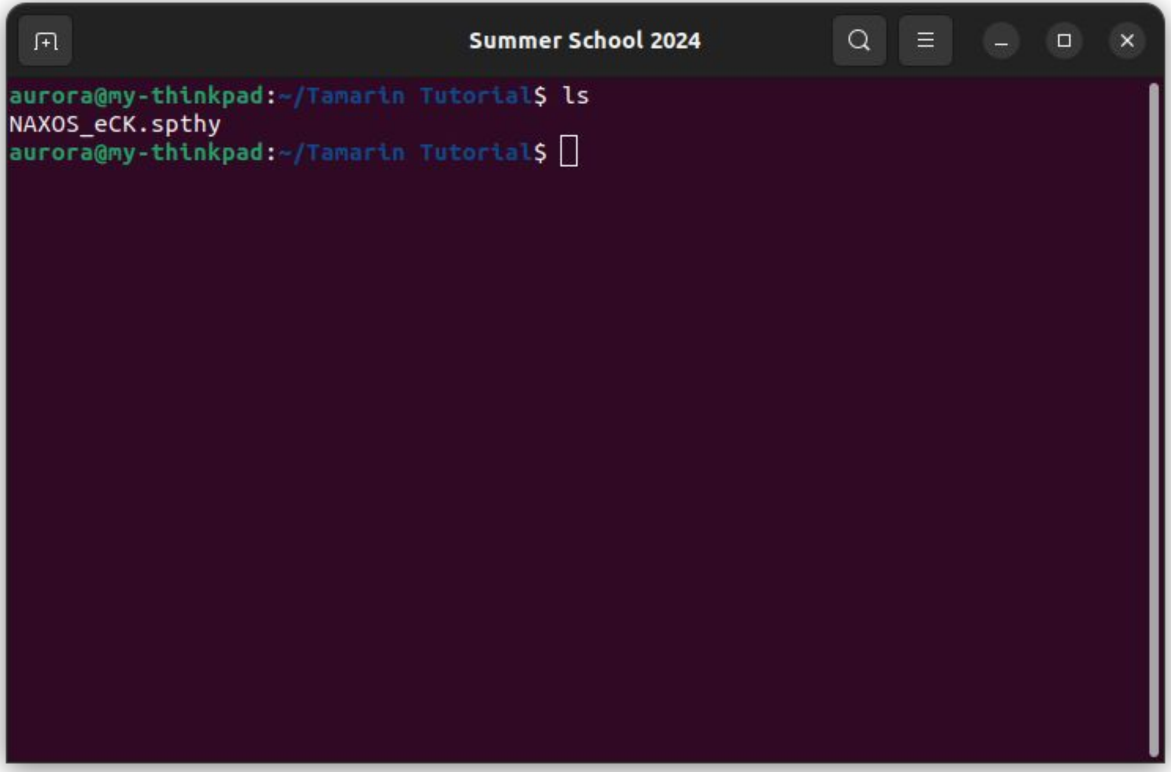
- **Morning Sessions**
  - S1: Introduction and small demo of the tool
  - S2: Getting started with incremental exercises
- **Afternoon Sessions**
  - S3: Advanced Features and Exercises
  - S4: Research

The Tamarin Prover

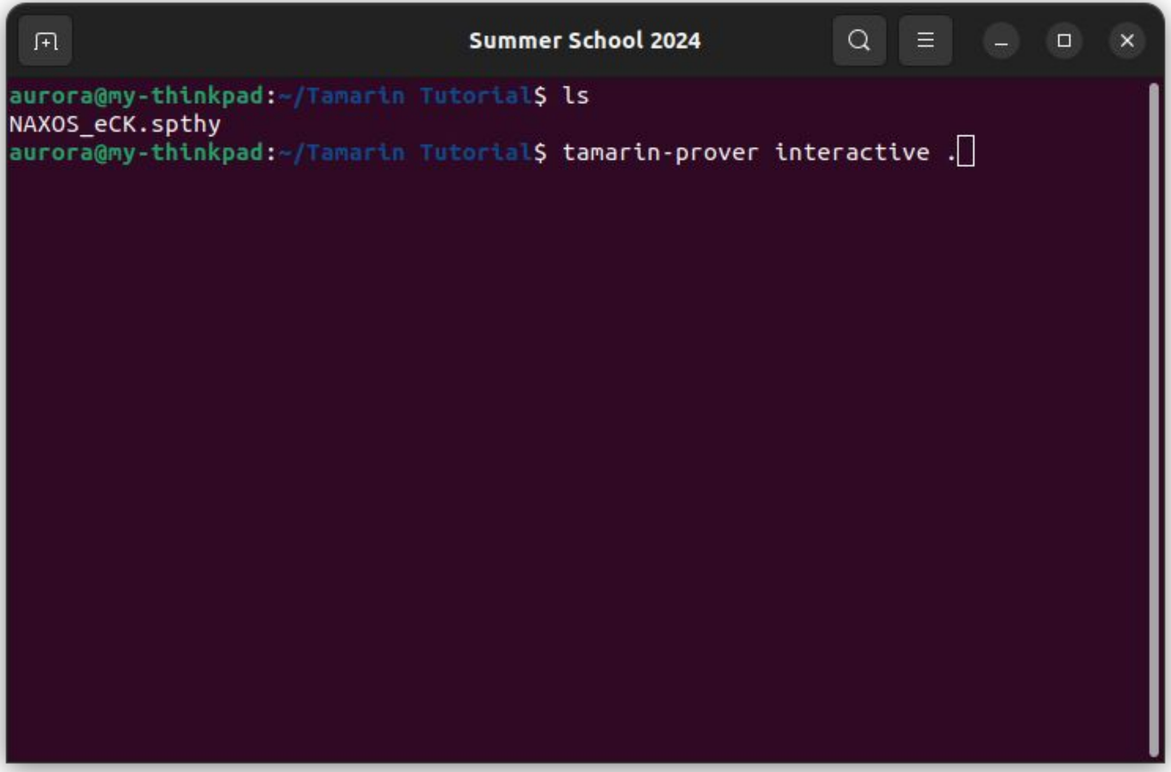


Demo

Demo


Summer School 2024
aurora@my-thinkpad:~/Tamarin Tutorial$ ls
NAXOS_eCK.spthy
aurora@my-thinkpad:~/Tamarin Tutorial$

Demo


Summer School 2024
aurora@my-thinkpad:~/Tamarin Tutorial$ ls
NAXOS_eCK.spthy
aurora@my-thinkpad:~/Tamarin Tutorial$ tamarin-prover interactive .

Demo

Summer School 2024

```
maude tool: 'maude'
 checking version: 3.1. OK.
 checking installation: OK.
GraphViz tool: 'dot'
 checking version: dot - graphviz version 2.43.0 (0). OK.
 checking PNG support: OK.
The server is starting up on port 3001.
Browse to http://127.0.0.1:3001 once the server is ready.

Loading the security protocol theories './*.spthy' ...

[Theory NAXOS_eCK] Theory loaded
[Theory NAXOS_eCK] Theory translated
[Theory NAXOS_eCK] Derivation checks started
[Theory NAXOS_eCK] Derivation checks ended
[Theory NAXOS_eCK] Theory closed
Finished loading theories ... server ready at

    http://127.0.0.1:3001

31/May/2024:13:09:46 +0200 [Info#yesod-core] Application launched @(yesod-core-1
.6.25.1-GqSGJZtNivBrk7KOeu9nl:Yesod.Core.Dispatch src/Yesod/Core/Dispatch.hs:188
:10)
```

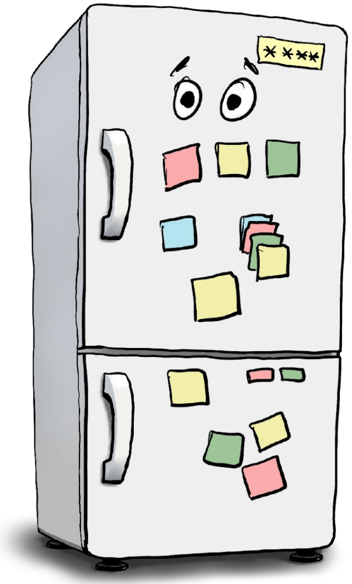
Modeling in Tamarin

Modeling in Tamarin

- 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

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

- **Transition relation**

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

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

- **Executions**

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

- **Traces**

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

'c' constant

Semantics: example 1

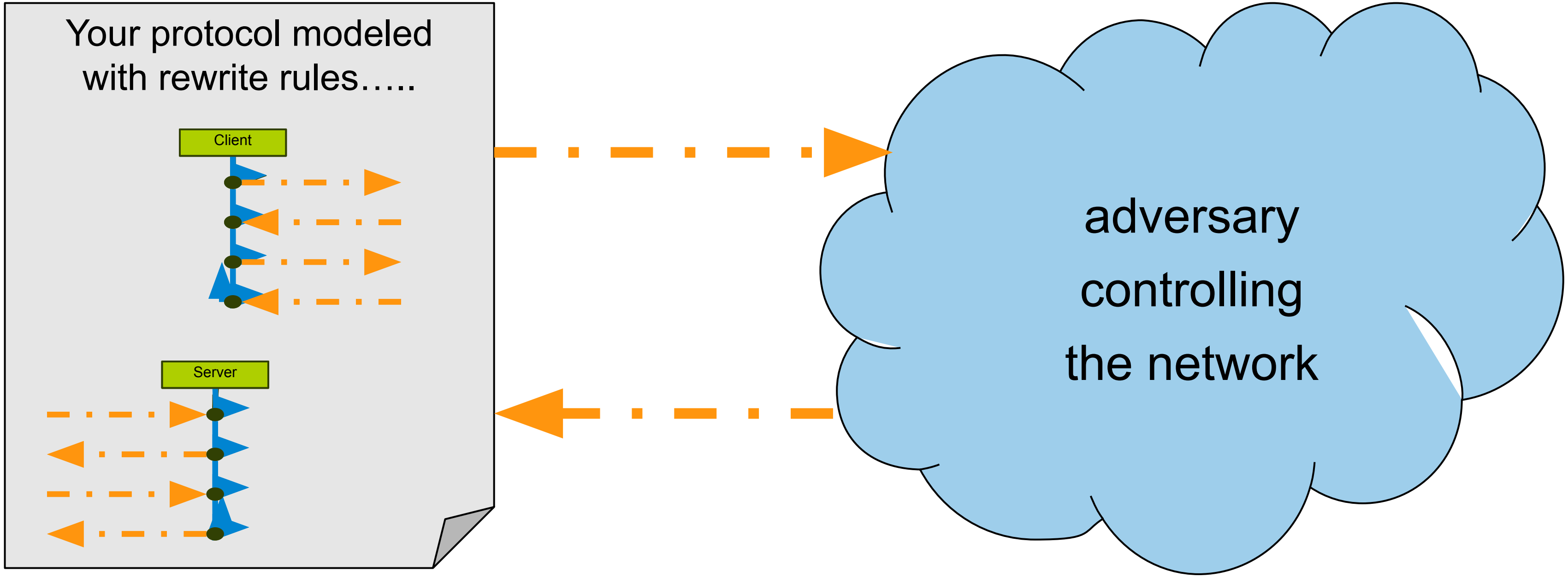
- **Rules**
  - rule 1: [ ] –[Init()]→ [A('5')]
  - rule 2: [A(x)] –[Step(x)]→ [B(x)]
- **Execution example**
  - [ ]
  - –[Init()]→ [A('5')]
  - –[Init()]→ [A('5'), A('5')]
  - –[Step('5')]→ [A('5'), B('5')]
- **Corresponding trace:** [Init(), Init(), Step('5')]

‘c’ constant

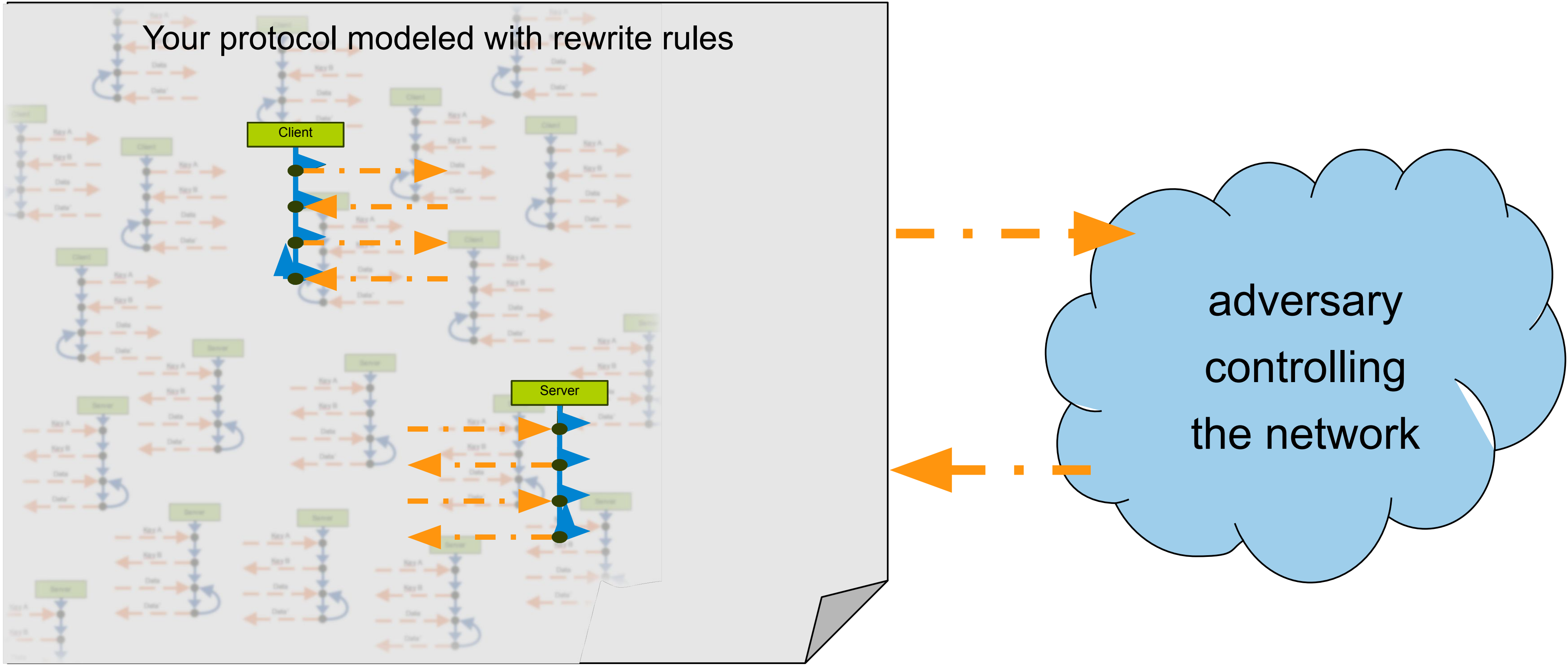
Semantics: example 2 (persistent facts)

- **Rules**
  - rule 1: [ ] –[Init()]→ [!C('ok'), D('1')]
  - rule 2: [!C(x), D(y)] –[Step(x,y)]→ [D(h(y))]
- **Execution example**
  - [ ]
  - –[Init()]→ [!C('ok'), D('1')]
  - –[Step('ok','1')]→ [!C('ok'), D(h('1'))]
  - –[Step('ok',h('1'))]→ [!C('ok'), D(h(h('1')))]
- **Corresponding trace:** [Init(), Step('ok', '1'), Step('ok', h('1'))]

Tamarin tackles complex interaction with adversary



Tamarin tackles complex interaction with adversary



The NAXOS protocol

The Naxos protocol

lkA	A's long-term priv. key
g^lkA	A's long-term pub. key
eskA	A's eph. priv. key

$\boxed{\mathrm{I}}$ $\boxed{\mathrm{R}}$

Fresh esk_I

$ex_I = h1(esk_I, lk_I)$

$hk_I = g^{ex_I}$ $\xrightarrow{hk_I}$ receive X

Fresh esk_R

$ex_R = h1(esk_R, lk_R)$

receive Y $\xleftarrow{hk_R}$ $hk_R = g^{ex_R}$

The Naxos protocol

lkA	A's long-term priv. key
g^lkA	A's long-term pub. key
eskA	A's eph. priv. key

$$
\begin{array}{lcl}
\boxed{\mathrm{I}} & & \boxed{\mathrm{R}} \\
\boxed{\begin{array}{l} \text{Fresh } esk_I \\ ex_I = h1(esk_I, lk_I) \\ hk_I = g^{ex_I} \end{array}} & \xrightarrow{hk_I} & \text{receive } X \\
& & \text{Fresh } esk_R \\
& & ex_R = h1(esk_R, lk_R) \\
\text{receive } Y & \xleftarrow{hk_R} & hk_R = g^{ex_R}
\end{array}
$$

The Naxos protocol

lkA	A's long-term priv. key
g^lkA	A's long-term pub. key
eskA	A's eph. priv. key

$$\boxed{\mathrm{I}} \qquad\qquad\qquad\qquad \boxed{\mathrm{R}}$$

Fresh esk_I

$ex_I = h1(esk_I, lk_I)$

$hk_I = g^{ex_I}$

$$\xrightarrow{hk_I}$$

receive X

Fresh esk_R

$ex_R = h1(esk_R, lk_R)$

$hk_R = g^{ex_R}$

$$\xleftarrow{hk_R}$$

receive Y

The Naxos protocol

lkA	A's long-term priv. key
g^lkA	A's long-term pub. key
eskA	A's eph. priv. key

$\boxed{\mathrm{I}}$ $\qquad\qquad\qquad\qquad$ $\boxed{\mathrm{R}}$

Fresh esk_I

$ex_I = h1(esk_I, lk_I)$

$hk_I = g^{ex_I}$ $\quad \xrightarrow{hk_I} \quad$ receive X

Fresh esk_R

$ex_R = h1(esk_R, lk_R)$

receive Y $\quad \xleftarrow{hk_R} \quad$ $hk_R = g^{ex_R}$

$$\boxed{key = h2(g^{(ex_R)(lk_I)}, g^{(ex_I)(lk_R)}, g^{(ex_I)(ex_R)}, I, R)}$$

Modeling Naxos

lkA	A's long-term priv. key
g^lkA	A's long-term pub. key
eskA	A's eph. priv. key

$\boxed{\mathrm{I}}$

$$\text{Fresh } \ esk_I$$

$$ex_I = h1(esk_I, lk_I)$$

$$hk_I = g^{ex_I} \qquad \xrightarrow{\quad hk_I \quad}$$

Modeling Naxos

lkA	A's long-term priv. key
g^lkA	A's long-term pub. key
eskA	A's eph. priv. Key
'c'	constant
~t	t has type fresh

$\boxed{\mathrm{I}}$

Fresh esk_I

$ex_I = h1(esk_I, lk_I)$

$hk_I = g^{ex_I}$ $\xrightarrow{\quad hk_I \quad}$

```
rule Init_1:
  let exI = h1(<~eskI, ~lkI >)
      hkI = 'g'^exI
  in
   [ Fr( ~eskI ) ] --> [ Out( hkI) ]
```

Modeling Naxos

lkA	A's long-term priv. key
g^lkA	A's long-term pub. key
eskA	A's eph. priv. Key
'c'	constant
~t	t has type fresh
$t	t has type public
!F	F is persistent

$\boxed{\mathrm{I}}$

Fresh esk_I

$ex_I = h1(esk_I, lk_I)$

$hk_I = g^{ex_I}$ $\xrightarrow{\quad hk_I \quad}$

```
rule generate_ltk:
   let pkA = 'g'^~lkA
   in
   [ Fr(~lkA) ] --> [ !Ltk( $A, ~lkA ), !PK( $A, pkA), Out(pkA) ]

rule Init_1:
  let exI = h1(<~eskI, ~lkI >)
      hkI = 'g'^exI
  in
   [ Fr( ~eskI ) ] --> [ Out( hkI) ]
```

Modeling Naxos

lkA	A's long-term priv. key
g^lkA	A's long-term pub. key
eskA	A's eph. priv. Key
'c'	constant
~t	t has type fresh
$t	t has type public
!F	F is persistent

$\boxed{\mathrm{I}}$

Fresh esk_I

$$ex_I = h1(esk_I, lk_I)$$

$$hk_I = g^{ex_I} \qquad \xrightarrow{\quad hk_I \quad}$$

```
rule generate_ltk:
   let pkA = 'g'^~lkA
   in
   [ Fr(~lkA) ] --> [ !Ltk( $A, ~lkA ), !PK( $A, pkA), Out(pkA) ]

rule Init_1:
  let exI = h1(<~eskI, ~lkI >)
      hkI = 'g'^exI
  in
   [ Fr( ~eskI ), !Ltk( $I, ~lkI ) ] --> [ Out( hkI) ]
```

Modeling Naxos

lkA	A's long-term priv. key
g^lkA	A's long-term pub. key
eskA	A's eph. priv. Key
'c'	constant
~t	t has type fresh
$t	t has type public
!F	F is persistent

$\boxed{\mathrm{I}}$

Fresh esk_I

$ex_I = h1(esk_I, lk_I)$

$hk_I = g^{ex_I}$ $\xrightarrow{hk_I}$

receive Y $\longleftarrow$

```
rule generate_ltk:
   let pkA = 'g'^~lkA
   in
   [ Fr(~lkA) ] --> [ !Ltk( $A, ~lkA ), !PK( $A, pkA), Out(pkA) ]

rule Init_1:
  let exI = h1(<~eskI, ~lkI >)
      hkI = 'g'^exI
  in
   [ Fr( ~eskI ), !Ltk( $I, ~lkI ) ] --> [ Out( hkI) ]

rule Init_2:
   [ In( Y ) ] --> []
```

Modeling Naxos

lkA	A's long-term priv. key
g^lkA	A's long-term pub. key
eskA	A's eph. priv. Key
'c'	constant
~t	t has type fresh
$t	t has type public
!F	F is persistent

$\boxed{\mathrm{I}}$

Fresh esk_I

$ex_I = h1(esk_I, lk_I)$

$hk_I = g^{ex_I}$ $\qquad \xrightarrow{\quad hk_I \quad}$

receive Y $\qquad \longleftarrow$

```
rule generate_ltk:
   let pkA = 'g'^~lkA
   in
   [ Fr(~lkA) ] --> [ !Ltk( $A, ~lkA ), !PK( $A, pkA), Out(pkA) ]

rule Init_1:
  let exI = h1(<~eskI, ~lkI >)
      hkI = 'g'^exI
  in
   [ Fr( ~eskI ), !Ltk( $I, ~lkI ) ] --> [ Out( hkI), Init_1( ~eskI, $I, $R, ~lkI ,hkI)]

rule Init_2:
   [ In( Y ) ] --> []
```

Modeling Naxos

lkA	A's long-term priv. key
g^lkA	A's long-term pub. key
eskA	A's eph. priv. Key
'c'	constant
~t	t has type fresh
$t	t has type public
!F	F is persistent

$\boxed{\mathrm{I}}$

Fresh esk_I

$ex_I = h1(esk_I, lk_I)$

$hk_I = g^{ex_I}$ $\xrightarrow{hk_I}$

receive Y $\longleftarrow$

```
rule generate_ltk:
   let pkA = 'g'^~lkA
   in
   [ Fr(~lkA) ] --> [ !Ltk( $A, ~lkA ), !PK( $A, pkA), Out(pkA) ]

rule Init_1:
  let exI = h1(<~eskI, ~lkI >)
      hkI = 'g'^exI
  in
   [ Fr( ~eskI ), !Ltk( $I, ~lkI ) ] --> [ Out( hkI), Init_1( ~eskI, $I, $R, ~lkI ,hkI)]

rule Init_2:
   [ Init_1( ~eskI, $I, $R, ~lkI ,hkI), In( Y ) ] --> []
```

Property specification

- `l --[ a ]-> r`
- Actions stored as (action) trace

  Additionally: adversary knows facts K()

lkA	A's long-term priv. key
g^lkA	A's long-term pub. key
eskA	A's eph. priv. Key
'c'	constant
~t	t has type fresh
$t	t has type public
!F	F is persistent

Property specification

- `l --[ a ]-> r`
- Actions stored as (action) trace

    Additionally: adversary knows facts K()

lkA	A’s long-term priv. key
g^lkA	A’s long-term pub. key
eskA	A’s eph. priv. Key
‘c’	constant
~t	t has type fresh
$t	t has type public
!F	F is persistent

```
rule Init_2:
  let exI = h1(< ~eskI, ~lkI >),
      key  = h2(< Y^~lkI, pkR^exI, Y^exI, $I, $R >)
  in
   [ Init_1( ~eskI, $I, $R, ~lkI , hkI), In( Y ), !Pk($R,pkR) ]
  --[ ]-->
   []
```

Property specification

lkA	A's long-term priv. key
g^lkA	A's long-term pub. key
eskA	A's eph. priv. Key
'c'	constant
~t	t has type fresh
$t	t has type public
!F	F is persistent

- `l --[ a ]-> r`
- Actions stored as (action) trace

  Additionally: adversary knows facts K()

```
rule Init_2:
  let exI = h1(< ~eskI, ~lkI >),
      key  = h2(< Y^~lkI, pkR^exI, Y^exI, $I, $R >)
  in
   [ Init_1( ~eskI, $I, $R, ~lkI , hkI), In( Y ), !Pk($R,pkR) ]
   --[ Accept(~eskI, $I, $R, key) ]-->
   []
```

Property specification

- `l --[ a ]-> r`
- Actions stored as (action) trace

  Additionally: adversary knows facts K()

lkA	A's long-term priv. key
g^lkA	A's long-term pub. key
eskA	A's eph. priv. Key
'c'	constant
~t	t has type fresh
$t	t has type public
!F	F is persistent

```
rule Init_2:
  let exI = h1(< ~eskI, ~lkI >),
      key  = h2(< Y^~lkI, pkR^exI, Y^exI, $I, $R >)
  in
   [ Init_1( ~eskI, $I, $R, ~lkI , hkI), In( Y ), !Pk($R,pkR) ]
   --[ Accept(~eskI, $I, $R, key) ]-->
   []


Lemma trivial_key_secrecy:
  "(All #i Test A B key. Accept(Test,A,B,key)@i => Not (Ex #j. K(key)@j ))"
```

Property specification

```
rule Ltk_reveal:
    [ !Ltk($A, lkA) ] --[ LtkRev($A) ]-> [ Out(lkA) ]
```

lkA	A’s long-term priv. key
g^lkA	A’s long-term pub. key
eskA	A’s eph. priv. Key
‘c’	constant
~t	t has type fresh
$t	t has type public
!F	F is persistent

Property specification

lkA	A's long-term priv. key
g^lkA	A's long-term pub. key
eskA	A's eph. priv. Key
'c'	constant
~t	t has type fresh
$t	t has type public
!F	F is persistent

```
rule Ltk_reveal:
   [ !Ltk($A, lkA) ] --[ LtkRev($A) ]-> [ Out(lkA) ]


lemma key_secrecy:
  /* If A and B are honest, the adversary doesn't learn the session key
   */
  "(All #i1 Test A B key.
    (
      Accept(Test, A, B, key) @ i1
      &
      not ( (Ex #ia . LtkRev( A ) @ ia )
          | (Ex #ib . LtkRev( B ) @ ib )
          )
    )
    ==> not (Ex #i2. K( key ) @ i2 )
  )"
```

eCK security model for key exchange

- Adversary can
  - learn **long-term keys**,
  - learn the **randomness** generated in sessions,
  - learn **session keys**

- But only as long as the Test session is ***clean***:
  - **No reveal of session key of** Test session or its **matching session**, and
  - No reveal of randomness of Test session as well as the long-term key of the actor, and
  - If there exists a matching session, then something is disallowed
  - If there is no matching session, then something else...

Specifying eCK

```
Lemma eCK_key_secrecy:
  "(All #i1 #i2 Test A B key.Accept(Test, A, B, key) @ i1
                               &K( key ) @ i2 ==>
  (




  )"
```

Specifying eCK

```
Lemma eCK_key_secrecy:
  "(All #i1 #i2 Test A B key.Accept(Test, A, B, key) @ i1
                                  &K( key ) @ i2 ==>
  (
      (Ex #i3. SesskRev( Test ) @ i3 )
    |




  )"
```

Specifying eCK

```
Lemma eCK_key_secrecy:
  "(All #i1 #i2 Test A B key.Accept(Test, A, B, key) @ i1
                                &K( key ) @ i2 ==>
  (
      (Ex #i3. SesskRev( Test ) @ i3 )
    | (Ex MatchingSession #i3 #i4 ms.
            ( Sid ( MatchingSession, ms ) @ i3
            & Match( Test, ms ) @ i4)
            & (Ex #i5. SesskRev( MatchingSession ) @ i5 ))

  )"
```

Specifying eCK

```
Lemma eCK_key_secrecy:
  "(All #i1 #i2 Test A B key.Accept(Test, A, B, key) @ i1
                               &K( key ) @ i2 ==>
  (
      (Ex #i3. SesskRev( Test ) @ i3 )
    | (Ex MatchingSession #i3 #i4 ms.
          ( Sid ( MatchingSession, ms ) @ i3
          & Match( Test, ms ) @ i4)
          & (Ex #i5. SesskRev( MatchingSession ) @ i5 ))
    | [ ...andsoforth... ]
  )"
```

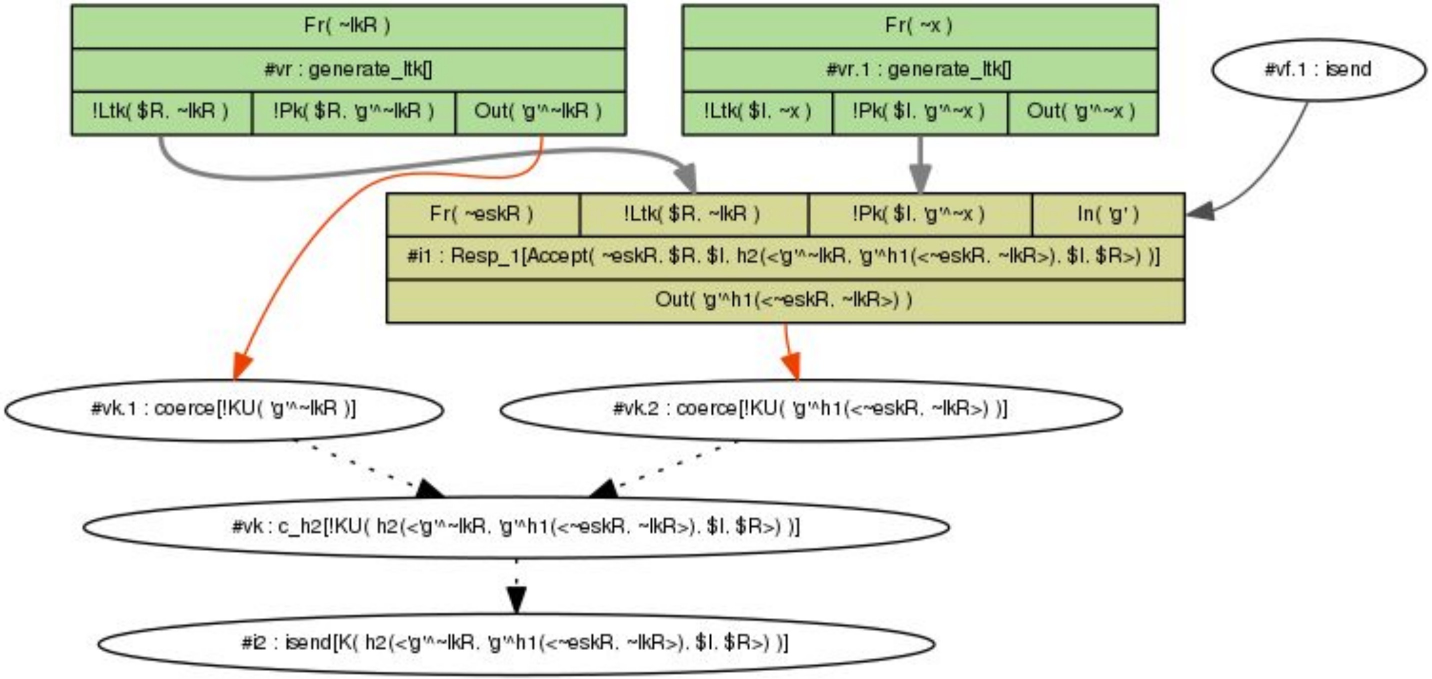
Specifying eCK

```
Lemma eCK_key_secrecy:
  "(All #i1 #i2 Test A B key.Accept(Test, A, B, key) @ i1
                              &K( key ) @ i2 ==>
  (
      (Ex #i3. SesskRev( Test ) @ i3 )
    | (Ex MatchingSession #i3 #i4 ms.
          ( Sid ( MatchingSession, ms ) @ i3
          & Match( Test, ms ) @ i4)
          & (Ex #i5. SesskRev( MatchingSession ) @ i5 ))
    | [ ...andsoforth... ]
  )"
```

If Test accepts and the adversary knows key, then the Test must not be fresh, i.e., "... **reveal of session key of Test session** or **its matching session**", or ...

Demo

Reading Tamarin’s graphs



Tamarin’s algorithm

Basic principles

- Backwards search using **constraint reduction rules** (>25!)
- Turn negation of formula into set of constraints
- Case distinctions
  - E.g.: Possible sources of a message or fact
- Each step of the Proof, solve goals:
  - Solve Premises based on Dependency Graphs
  - Deconstruction Chains
  - Solve Action goals, …

Lemmas

- When it doesn’t terminate…
- Guide the proof manually; export
- Write **lemmas**
  - “**Hints**” for the prover
    - They don't change the guarantees, only help tool in finding a proof
  - E.g. specify lemma that can be used to prune proof trees at multiple points

How do I know my model is correct?

- **It is easy to model something incorrectly**
- Executability: try to prove expected traces actually exist
- Break the protocol on purpose
- Much easier to check these things than in manual proofs!

Tamarin: Conclusions

- **Tamarin** offers **many unique features**
  - Unbounded analysis, flexible properties, equational theories, global state
  - Enables automated analysis in areas previously unexplored
- It has many **other features** I didn't touch on now
  - Induction, restrictions, reusable lemmas, heuristics, ...
  - Many new features planned!
- Tool and sources are **free**; development on Github

aurora.naska@cispa.de

What’s next

- Please install Tamarin if you have not done so yet.
- Start from files in Session 2:
  - https://github.com/sgiampietro/tamarin-tutorial
- Ask us for help!

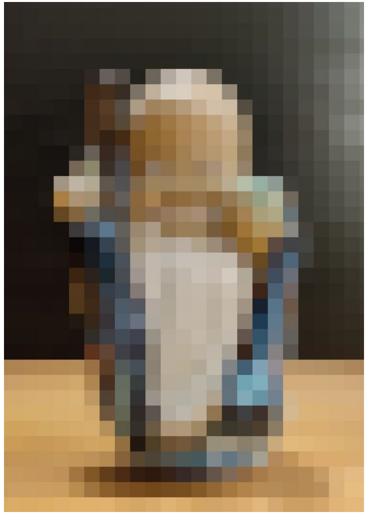
Heuristics?

- If Tamarin terminates, one of two options:
  - **Proof**, or
  - **counterexample** (in this context: attack)
- At each stage in proof, multiple constraint solving rules might be applicable
  - Similar to “how shall I try to prove this?”
  - Choice influences speed & termination, but not the outcome after termination
- Complex **heuristics choose rule**
  - user can give hints or override

Symbolic vs Computational?

Modeling real-world objects

Reality

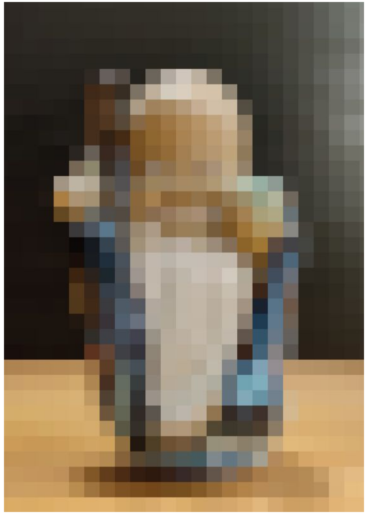


Symbolic

Modeling real-world objects

Reality

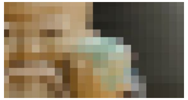
Computational



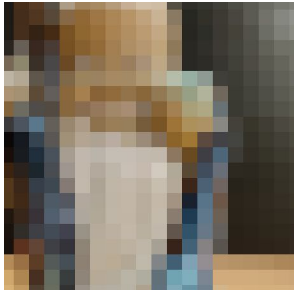
Symbolic

Modeling real-world objects

Reality



Computational



Symbolic

References

- Tamarin on github (https://tamarin-prover.github.io/)
  - Notably links to: all sources, example files, mailing list/google group, manual, tutorial data, (incomplete) list of papers
- More accurate modeling of cryptography
  - *Seems Legit: Automated Analysis of Subtle Attacks on Protocols that Use Signatures*
    Jackson, Cremers, Cohn-Gordon, Sasse – ia.cr/2019/779
  - *Prime, Order Please! Revisiting Small Subgroup and Invalid Curve Attacks on Protocols using Diffie-Hellman*
    Cremers, Jackson – ia.cr/2019/526
- Improving automation
  - *Automatic Generation of Sources Lemmas in Tamarin: Towards Automatic Proofs of Security Protocols*
    Cortier, Delaune, Dreier – Springer/HAL report
- EMV Chip and pin → attack to circumvent PIN requirement for VISA contactless
  - *The EMV Standard: Break, Fix, Verify*
    Basin, Sasse, Toro – emvrace.github.io