Modeling in Tamarin

Cas Cremers

Email: cremers@cispa.de

All materials under CC-BY license

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

June 2023

About me

- **Formal methods/symbolic analysis**
  - Co-developer of Scyther & Tamarin
- **Applied cryptography**
  - Security models & proof techniques
- **Standardization and real-world applications**
  - TLS 1.3, IEEE 802.11, ISO, SPDM, …
  - Secure messaging (contributed to MLS)
  - DP3T / Corona Warn App
- Looking to hire phd & postdocs!

Demo

Demo

Tamarin

```
cas@Yoga:~/tamarin_ex3_from_slides$ ls
foo_eligibility.spthy  NAXOS_eCK_PFS.spthy  sources-nolemma-load.spthy
loop.spthy             NAXOS_eCK.spthy      sources.spthy
cas@Yoga:~/tamarin_ex3_from_slides$
```

Demo

```
Tamarin
cas@Yoga:~/tamarin_ex3_from_slides$ ls
foo_eligibility.spthy  NAXOS_eCK_PFS.spthy  sources-nolemma-load.spthy
loop.spthy             NAXOS_eCK.spthy      sources.spthy
cas@Yoga:~/tamarin_ex3_from_slides$ tamarin-prover interactive .
```

Demo

Tamarin

```
 stdout:  2.7

 stderr:
 checking installation: 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' ...
Finished loading theories ... server ready at

    http://127.0.0.1:3001

08/Dec/2020:19:56:06 +0100 [Info#yesod-core] Application launched @(yesod-co
re-1.6.18-Ab7hNtiUzJgGsCLpKcpJyh:Yesod.Core.Dispatch src/Yesod/Core/Dispatch
.hs:163:11)
```

Tamarin: high-level

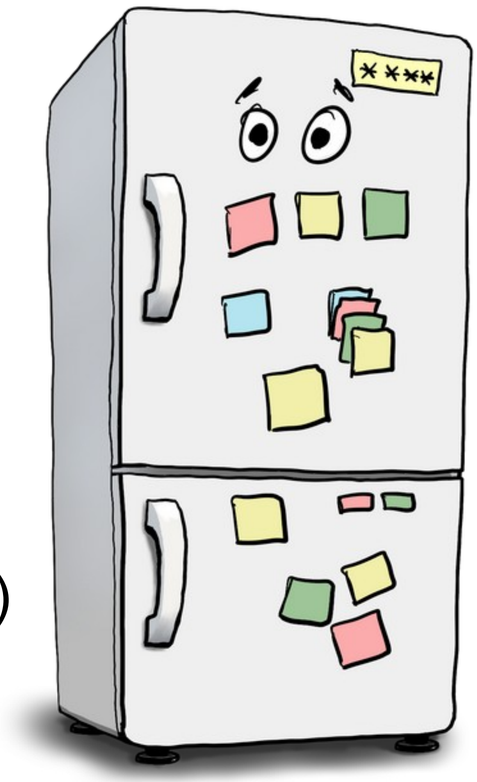
- **Modeling** protocol & adversary done using multiset rewriting
  - Specifies transition system; induces set of traces
- **Property** specification using fragment of first-order logic
  - Specifies “good” traces
- Tamarin tries to
  - provide proof that all system traces are good, or
  - construct a counterexample trace of the system (attack)

Modeling in Tamarin

Modeling in Tamarin

- **Multiset rewriting**; surprisingly similar to "oracles"
- 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, …

The model

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

The model

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

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**)]
      - With additional constraints on systems such that **x** unique

Semantics

- **Transition relation**

  $S \text{–[a]}\!\to_R ((S \setminus^\# l) \cup^\# r)$

  where

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

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**

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

Semantics

- **Transition relation**

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

  where

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

- **Executions**

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

- **Traces**

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

‘c’ constant

Semantics: example 1

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

‘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')]

‘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**
  - rule1: [ ] –[Init()]→ [!C('ok'), D('1')]
  - rule2: [!C(x), D(y)] –[Step(x,y)]→ [D(h(y))]

‘c’ constant

Semantics: example 2 (persistent facts)

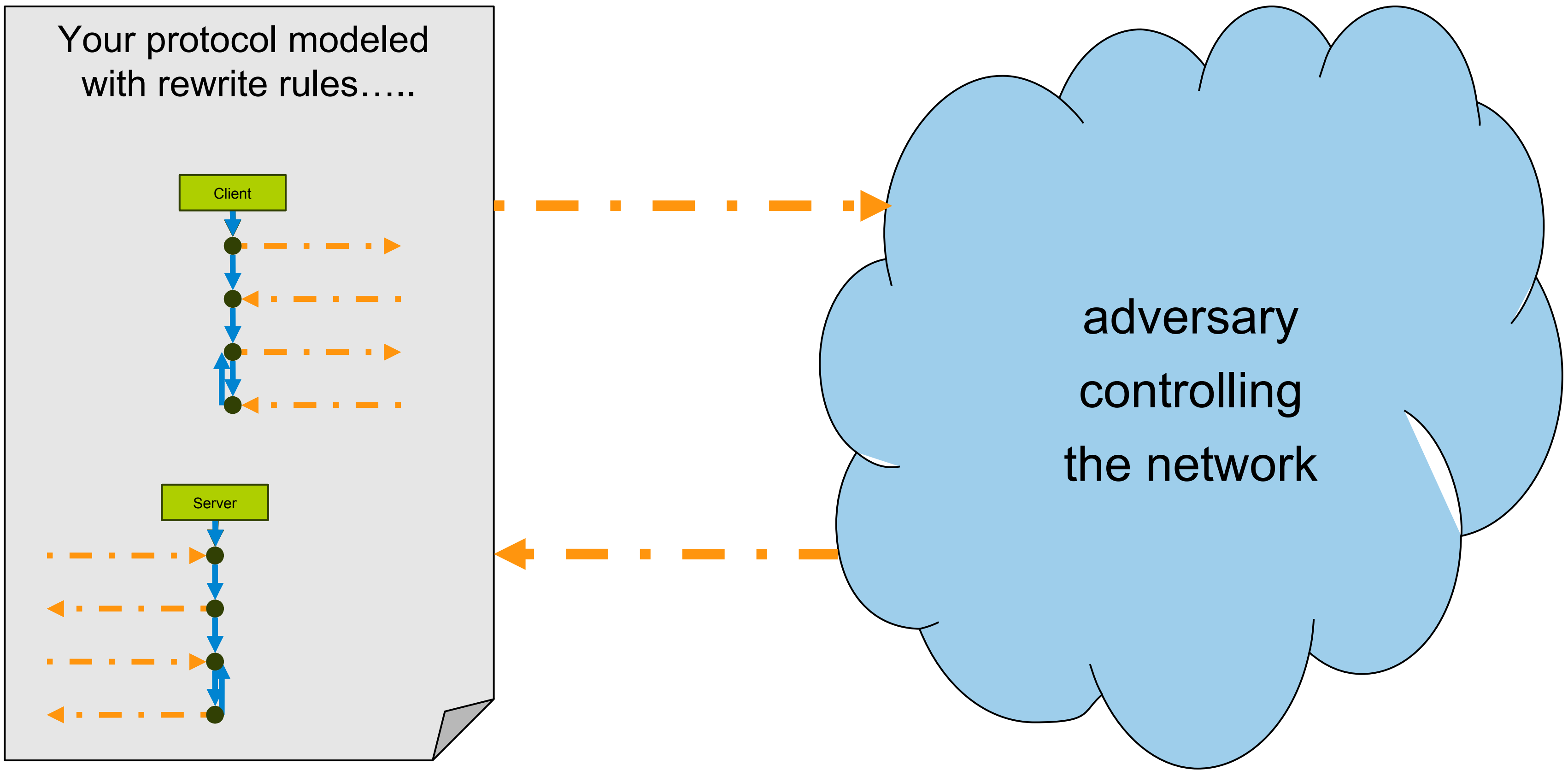
- **Rules**
  - rule1: [ ] –[Init()]→ [!C('ok'), D('1')]
  - rule2: [!C(x), D(y)] –[Step(x,y)]→ [D(h(y))]
- **Execution example**
  - [ ]
  - –[Init()]→ [!C('ok'), D('1')]
  - –[Step('ok','1')]→ [!C('ok'), D(h('1'))]
  - –[Step('ok',h('1'))]→ [!C('ok'), D(h(h('1')))]

‘c’ constant

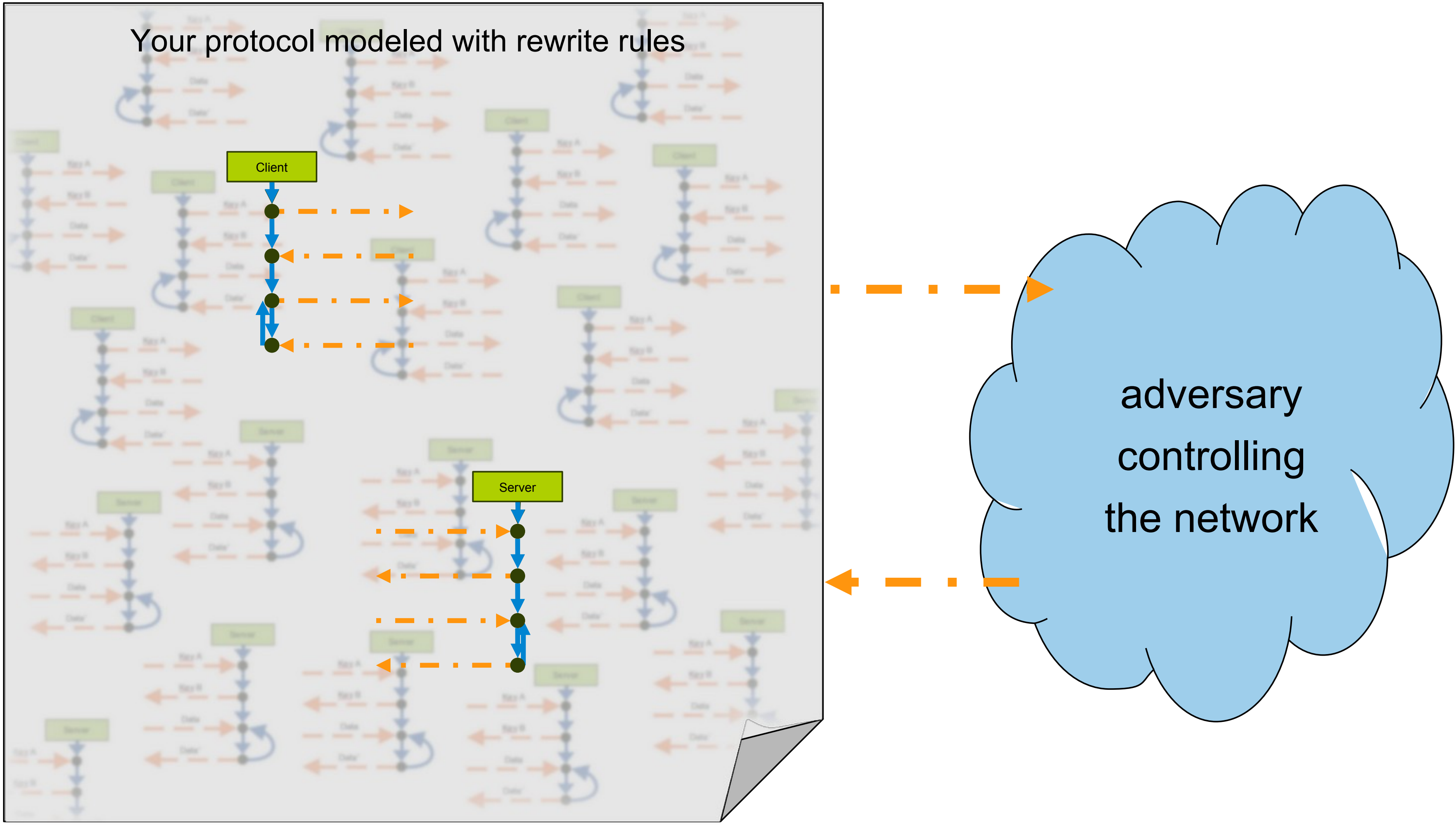
Semantics: example 2 (persistent facts)

- **Rules**
  - rule1: [ ] –[Init()]→ [!C('ok'), D('1')]
  - rule2: [!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

I		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

I		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	$\xleftarrow{hk_R}$	$hk_R = g^{ex_R}$

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

Modeling Naxos

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

$\boxed{\mathrm{I}}$

Fresh esk_I

$ex_I = h1(esk_I, lk_I)$

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

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

Modeling Naxos

I

Fresh esk_I

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

$$hk_I = g^{ex_I}$$

$$\xrightarrow{hk_I}$$

```
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

I

Fresh esk_I

$ex_I = h1(esk_I, lk_I)$

$hk_I = g^{ex_I}$

$\xrightarrow{hk_I}$

```
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

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

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

- first order logic interpreted over a trace

– False	False
– Equality	$t_1 =_E t_2$
– Timepoint ordering	#i < #j
– Timepoint equality	#i = #j
– Action at timepoint #i	A@#i

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

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) ]-->
   []
```

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()

```
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

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**

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

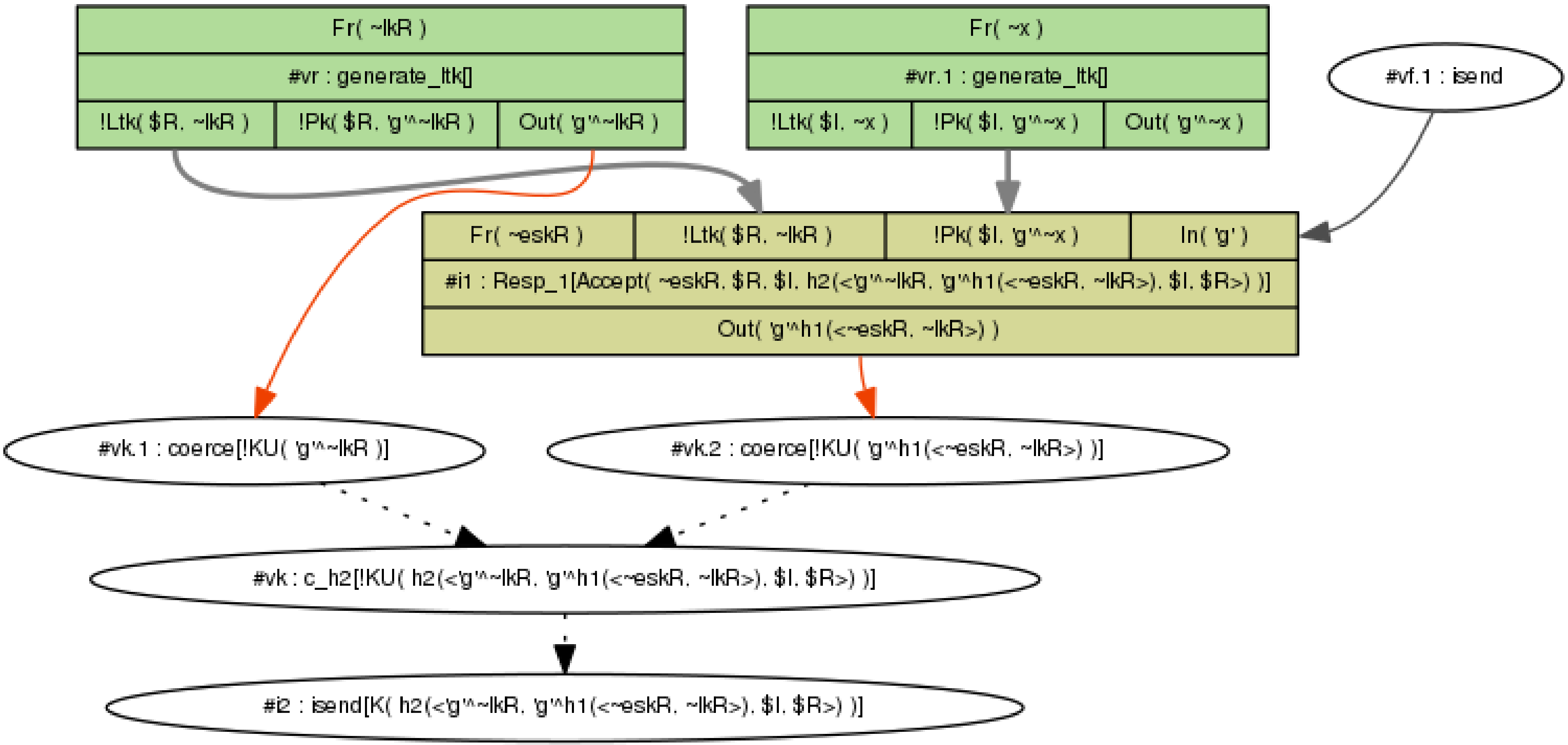
```
Lemma eCK_key_secrecy:
  "(All #i1 #i2 Test A B key. Accept(Test, A, B, key) @ i1
                              & K( key ) @ i2 ==>
  (
      (Ex #i3. SesskRev( Test ) @ i3 )
    | (Ex MatchingSession #i3 #i4 ms.
           ( Sid ( MatchingSession, ms ) @ i3
           & Match( Test, ms ) @ i4)
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

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

Demo

Tamarin’s algorithm

Reading Tamarin’s graphs



Algorithm intuition

- **Constraint solving algorithm**
- Main ingredients:
  - Dependency graphs
  - Deconstruction (decryption) chains
  - Finite variant property

Algorithm intuition

- **Constraint solving algorithm**
- Main ingredients:
  - Dependency graphs
  - Deconstruction (decryption) chains
  - Finite variant property
- **Invariant**: if adversary knows M then either
  - M was sent in plain
  - Adversary can construct M by knowing subterms
  - Adversary can deconstruct M …. from message sent by protocol rule

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
- Try to establish:
  - no solutions exist for constraint system, or
  - there exists a „realizable“ execution (trace)
- If multiple rules can be applied: use heuristics

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

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

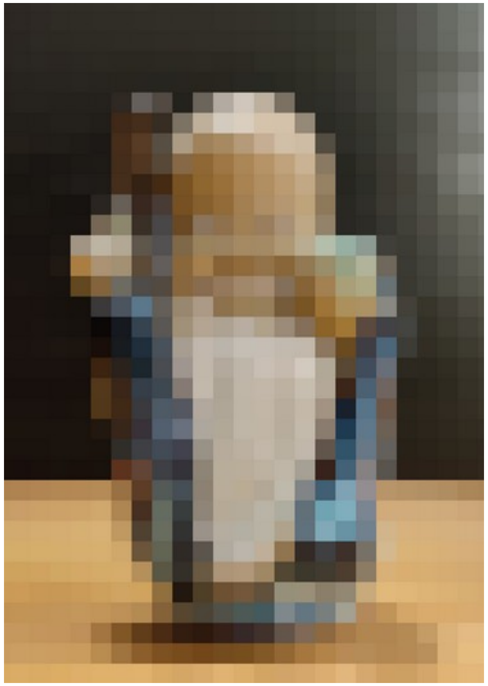
How do I know my model is correct?

- **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!

Symbolic vs Computational?

Modeling real-world objects

Reality

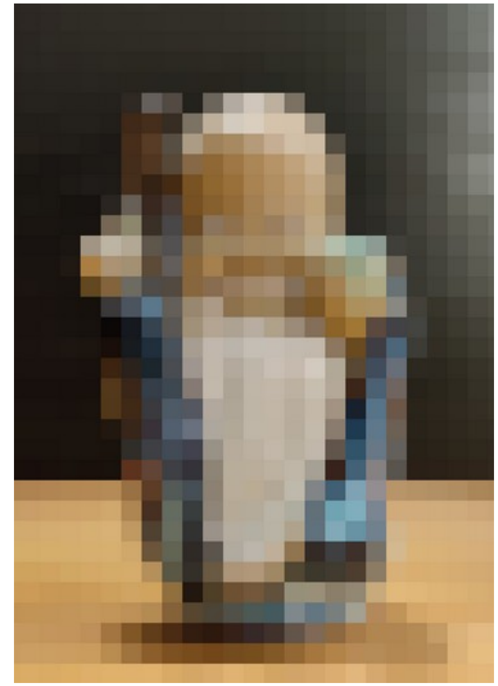


Symbolic

Modeling real-world objects

Reality

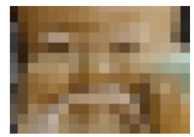
Computational



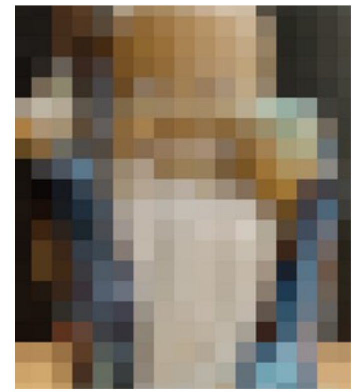
Symbolic

Modeling real-world objects

Reality



Computational



Symbolic

Symbolic analysis for cryptographers

- **Fundamental differences**
  - Dolev-Yao attacker strong abstraction of Probabilistic Polynomial Time Turing Machine
  - Terms are an abstract view of bitstrings
  - No quantitative information (e.g. bounds)
- Current **algorithm limitations**
  - Restrictions on equational theories, e.g., MQV style exponentiation tricky: we miss Kaliski's UKS attack on MQV.
- **What we *can* do** (recent developments)
  - Negotiation, weak crypto
  - Non-prime order curves
  - DSKS attacks
  - Length extension attacks

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 tuning, ...
  - Many new features planned!
- Tool and sources are **free**; development on Github

cremers@cispa.de

Morning exercise

- Start from files in https://github.com/tamarin-prover/teaching/tree/master/tutorial-models/1_morning
- Consider `NAXOS_01_simple.spthy`
  - Remove specific elements:
    - Remove the first argument to the `h2` function used to compute the session key, and check with Tamarin what happens if you analyse the properties
      - Note that you need to make the change both at the initiator and the responder
    - Remove the second argument instead, etc. etc.
- Repeat for `NAXOS_08_eCK.spthy`
  - Compare the results to before. Why do they differ?
- Compare `NAXOS_08_eCK.spthy` and `NAXOS_15_eCK_FPS.spthy`
  - Explain the difference (attacks?)

Afternoon exercise

- Try Benjamin Kiesl’s Toy Protocol tutorial:

  https://github.com/benjaminkiesl/tamarin_toy_protocol

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