Security protocol analysis using the Tamarin Prover

David Basin, **Cas Cremers**, Jannik Dreier, Ralf Sasse

Morning overview

- Model checking and verification
- For security protocols:
  The Tamarin Prover
  - Modeling
  - Attacks and proofs
  - Algorithm intuition
  - In practice
- Hands-on session

Afternoon overview

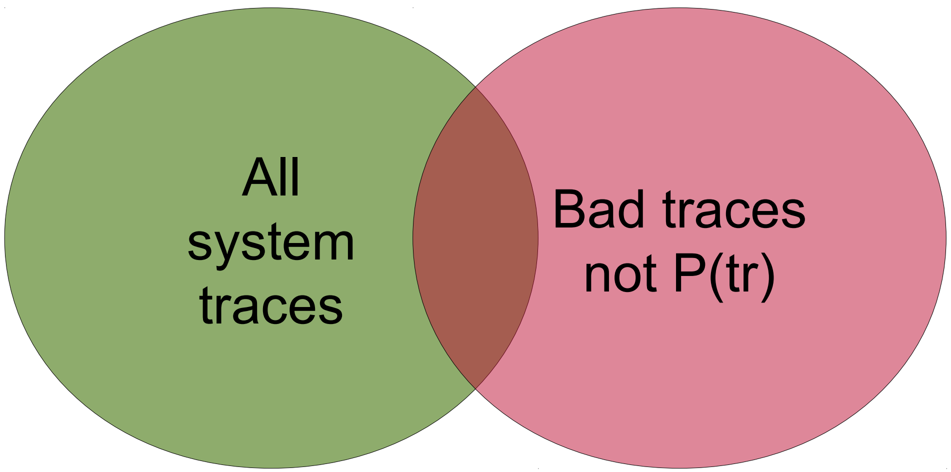
- Recap
- Induction
- State space reduction by sources lemmas
- Equational theories and adversary rules
- Observational equivalence

Problem

- How do we know if a protocol is secure?
  - Traditional: Smart people stare at it
- **More structured approach**:
  Specify threat model & intended property
  - Stare at the protocol, try to find attack.
  - Write the proof
- Can **formal methods** help?
  - Model checking, verification

Trace properties

- For now: trace properties (but more later!):
  - $\forall\, tr \in traces(System)\,.\, P(tr)$



Intersection empty?

Symbolic security analysis

- Idea: make transition system
  - with protocol participants
  - with adversary controlling network
- Encode property
  - **Authentication:**
    In all traces, if an initiator completes, there exists a responder with…
  - **Secrecy:**
    There is no trace in which Adversary learns k
- And check!
- Unfortunately, this turns out to be undecidable

The Tamarin Prover

- Symbolic analysis tool for systems in presence of a Dolev-Yao style network adversary

- Recent highlights:
  - Group key exchange protocols
  - ARPKI
  - TLS 1.3 (See talk tomorrow at TLS:DIV !)

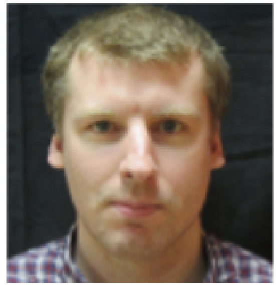
Simon Meier

Benedikt Schmidt

Cas Cremers

David Basin

Simon Meier



Benedikt Schmidt

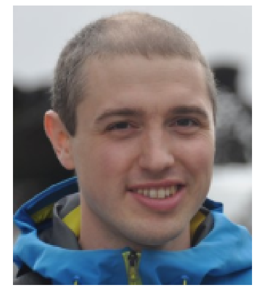
Cas Cremers

David Basin

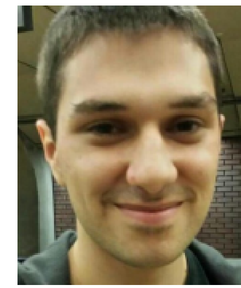
Robert Kunneman

Steve Kremer

Ralf Sasse



Jannik Dreier

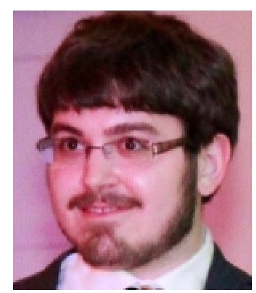


Cedric Staub

Sasa Radomirovic

Lara Schmid

Charles Dumenil



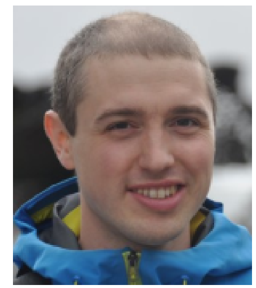
Kevin Milner

and more soon!

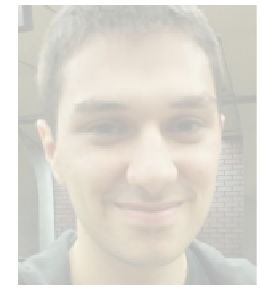
Cas Cremers

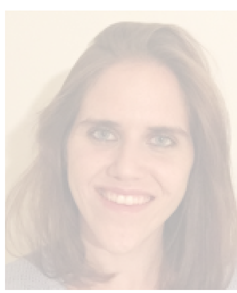
David Basin

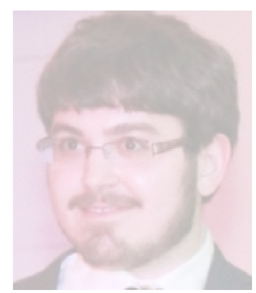
Ralf Sasse



Jannik Dreier







What can Tamarin do for you?

- Rapid prototyping
- Finding attacks before you start a proof effort
- Provide a symbolic proof
- Explore alternative designs/threat models quickly

Selected case studies

- AKE
  - Naxos
  - Signed DH
  - KEA+
  - UM
  - Tsx
- Group protocols
  - GDH
  - TAK
  - (Sig)Joux
  - STR
- ID-based AKE
  - RYY
  - Scott
  - Chen-Kudla
- Loops
  - TESLA1 & 2
- Non-monotonic global state
  - Keyserver
  - Envelope
  - Exclusive secrets
  - Contract signing
  - Security device
  - YubiKey
  - YubiHSM
- PKI with strong guarantees
  - ARPKI (also global state)
- Transparency
  - DECIM (also global state)
- TLS 1.3
  - Rev 10, 10+, and current

Resources & documentation

- Sources on github
- 100+ page manual
- Plenty of examples/case studies
- Algorithm details in theses, papers

Demo

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

- **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(_,_), $\_\hat{}\_$, $\_^{-1}$, _*_, 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**

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

  where l –[a]→ r is a ground instance of a rule and $l \subseteq^{\#} S$

- **Executions**

  $\text{Exec}(\,R) = \;\{\, [\,] \text{ –}[a_1]\!\rightarrow \ldots \text{–}[a_n]\!\rightarrow S_n$
  $\mid \forall n\,.\, \text{Fr}(n)$ appears only once on rhs $\}$

- **Traces**

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

‘c’ constant

Semantics: example 1

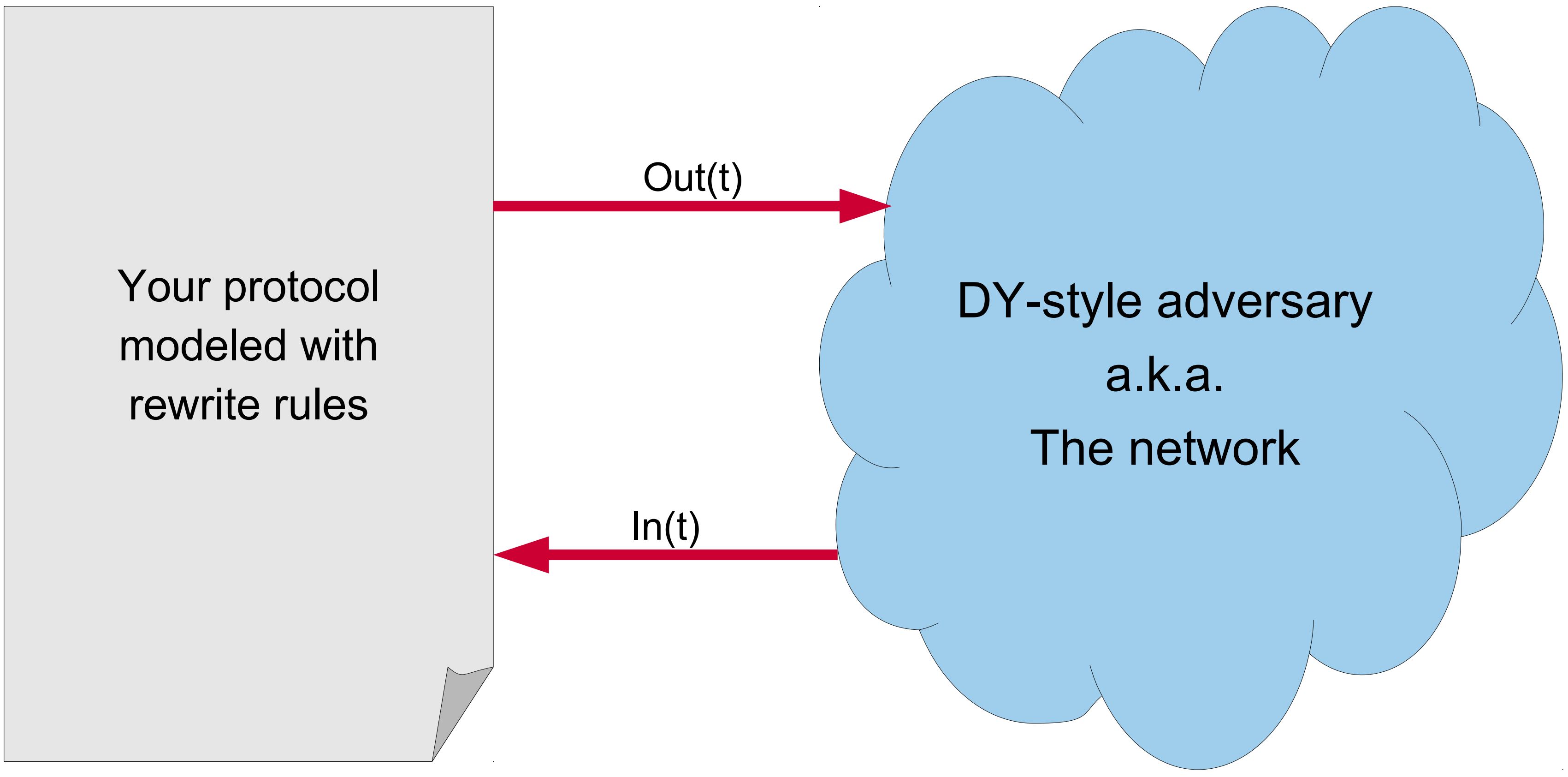
- **Rules**
  - rule 1: [ ] –[Init()]→ [A('5')]
  - rule 2: [A(x)] –[Step(x)]→ [B(x)]
- **Execution example**
  - [ ]
  - –[Init()]→ [A('5')]
  - –[Init()]→ [A('5'), A('5')]
  - –[Step('5')]→ [A('5'), B('5')]
- **Corresponding trace**
  - [Init(), Init(), Step('5')]

‘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))]
- **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



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

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

Modeling Naxos

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

$\boxed{\mathrm{I}}$

$$\text{Fresh } esk_I$$

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

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

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

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

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

```
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

$\boxed{\mathrm{I}}$

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

$\boxed{\mathrm{I}}$

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) ]


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

- 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

```
rule Init_2:
  let exI = h1(< ~eskI, ~lkI >),
      kI  = 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, kI) ]-->
   []


Lemma trivial_key_secrecy:
  ''(All #i Test A B k. Accept(Test,A,B,k)@i => Not (Ex #j. K(k)@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 k.
    (
      Accept(Test, A, B, k) @ i1
      &
      not ( (Ex #ia . LtkRev  ( A ) @ ia )
          | (Ex #ib . LtkRev  ( B ) @ ib )
          )
    )
    ==> not (Ex #i2. K( k ) @ 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...

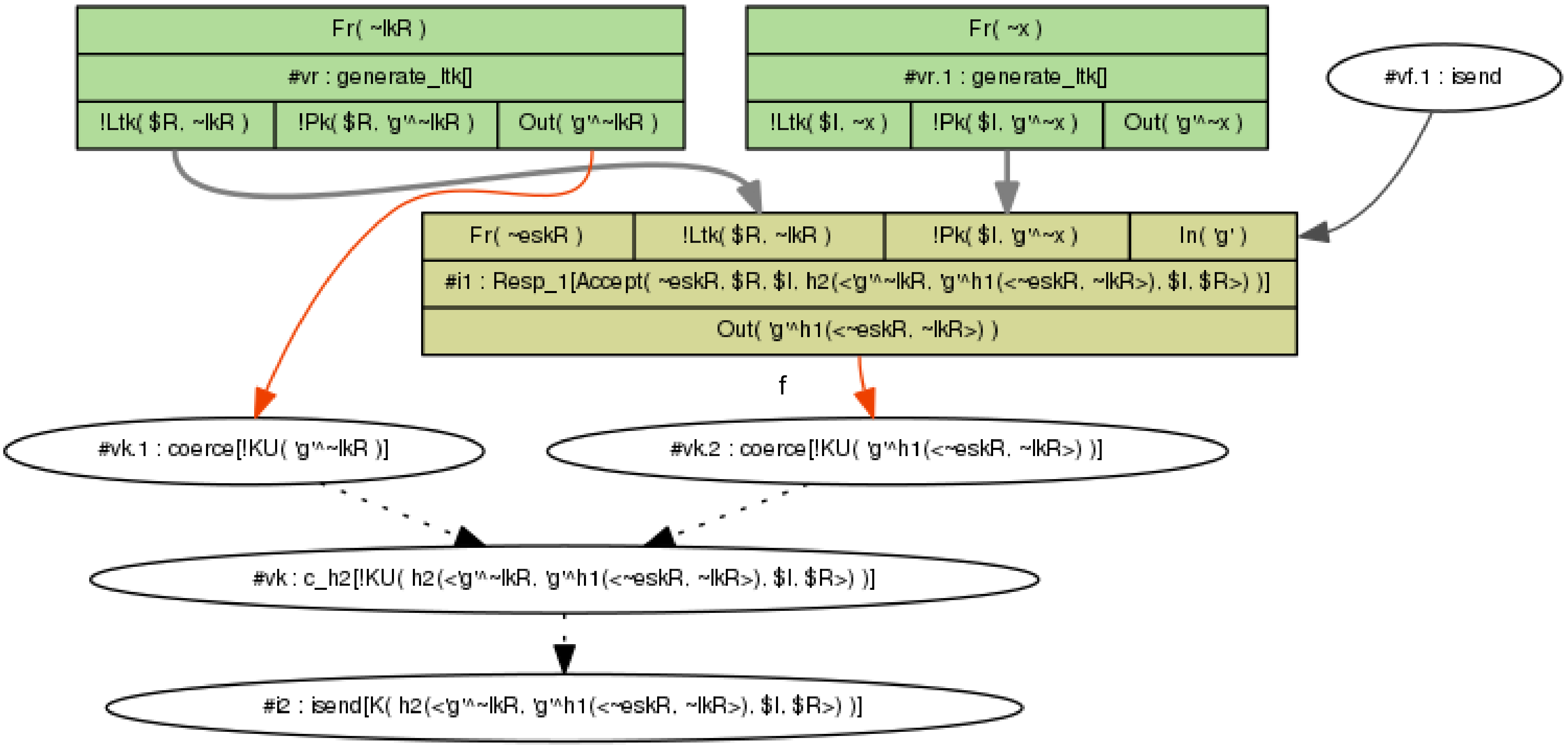
Specifying eCK

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

Reading Tamarin's graphs



Algorithm intuition

- **Constraint solving algorithm**
- Main ingredients:
  - Dependency graphs
  - Deconstruction (decryption) chains
  - Finite variant property (more this afternoon)
- **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** (27!)
- 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

Demo

How do I know my model is correct?

- **Lots of ways to cause errors**
- Look at the chains...
  - (requires an understanding of the algorithm)
- Executability
- Break the protocol on purpose
- Much easier to check these things than in manual proofs!

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 proof obligation, only help finding a proof
  - Specify lemma that can be used to prune proof trees at multiple points
  - ... more this afternoon and at TLS:DIV

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**
  - Limited equational theories, e.g., MQV style exponentiation tricky: we miss Kaliski's UKS attack on MQV.
- **What we *could* do** (but often don't; ongoing work)
  - Negotiation, weak crypto
  - Small subgroup attacks
  - DSKS attacks
  - Length extension attacks

Hands-on session!

- Take Naxos-simplified (not full eCK)
- Remove
  - First argument to KDF, check what happens
  - Second etc.
- Load more complex threat model version
  - Do the same for the properties that held before
- Load eCK model version

Discussion & solutions

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 (some this afternoon!)
  - Induction, restrictions, reusable lemmas, heuristics tuning, …
  - Tomorrow at TLS:DIV – TLS 1.3 analysis
  - Many new features planned!
- Tool and sources are **free**; development on Github

Cas.cremers@cs.ox.ac.uk