Security protocol analysis using the Tamarin Prover

Cas Cremers

CISPA
HELMHOLTZ CENTER FOR INFORMATION SECURITY

Overview & Structure

- **Overview**
  - Introduction
  - Tamarin's foundations
  - Modeling a protocol
  - Modeling security properties
  - Algorithm intuition
  - In practice
  - Symbolic vs computational
  - Where do I go from here?

Overview & Structure

- **Mode of operation**
  - No need to use tools during the talk, exercises are for afterwards
  - Short blocks (circa 20 minutes), then answering questions
  - Please write your questions in the Zulip chat channel:

    `”Vericrypt2020 Tamarin Prover”`

    You should have a link in your mail
  - We’ll have a break in the middle (10-15 minutes)

Security Protocols

Problem

- **How do we know if a protocol is secure?**
  - Traditionally: Smart people stare at it

Problem

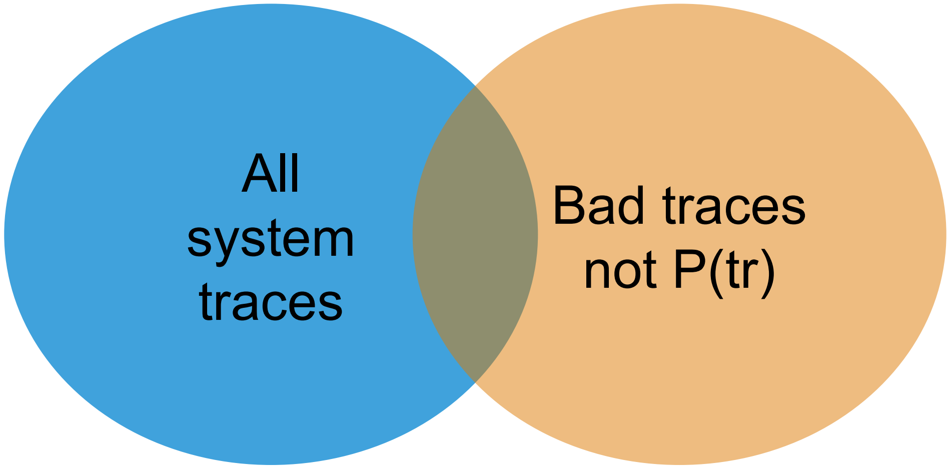
- **How do we know if a protocol is secure?**
  - Traditionally: Smart people stare at it
- **More structured approach:**
  - Specify threat model & intended property
  - Stare at the protocol, try to find attack
  - Write the proof

Problem

- **How do we know if a protocol is secure?**
  - Traditionally: 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 the 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

- Some highlights:
  - TLS 1.3
  - 5G-AKA
  - EMV (Chip and pin)

What can Tamarin do for you?

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

Selected case studies

- Key exchange protocols
  - Naxos, Signed DH, KEA+, UM, Tsx
- Group protocols
  - GDH, TAK, (Sig)Joux, STR
- Identity-based KE
  - RYY, Scott, Chen-Kudla
- Loops
  - TESLA1 & 2
- Non-monotonic global state
  - Keyserver, Envelope, Exclusive secrets, Contract signing, Security device
- PKI and friends
  - ARPKI, DECIM
- Detailed cryptographic primitives
  - WS-Security, X509, Scuttlebut, Let's Encrypt ACME, Bluetooth handshake, Tendermint

- More complex analyses:
  - TLS 1.3
  - EMV (Chip and pin)
  - 5G-AKA
  - 802.11 WPA2 (Wifi)
  - TPM 2.0 direct anonymous attestation
  - DNP3 SAv5 (power grid)
  - Noise protocols
  - YubiKey/YubiHSM

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)
```

ETH zürich

Simon
Meier

Benedikt
Schmidt

Cas
Cremers

David
Basin

ETH zürich

UNIVERSITY OF OXFORD

Inria

CISPA
HELMHOLTZ CENTER FOR INFORMATION SECURITY

Simon Meier

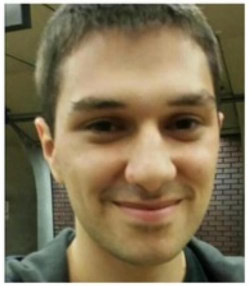
Benedikt Schmidt

Cas Cremers

David Basin

Robert Kunneman

Steve Kremer



Cedric Staub

Jannik Dreier

Ralf Sasse

Sasa Radomirovic

Lara Schmid

Charles Dumenil

Kevin Milner

Lucca Hirschi

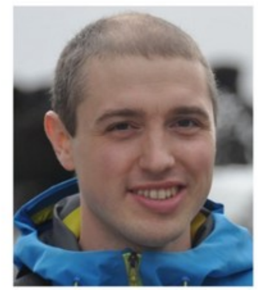
ETH zürich

Loria Inria

CISPA HELMHOLTZ CENTER FOR INFORMATION SECURITY

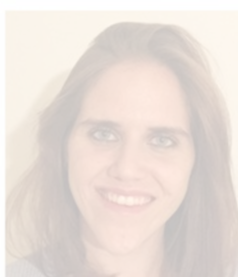
Cas Cremers

David Basin



Jannik Dreier

Ralf Sasse



Resources & documentation

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

Tamarin prover

Constraint solver

Tamarin prover

Theorem Prover

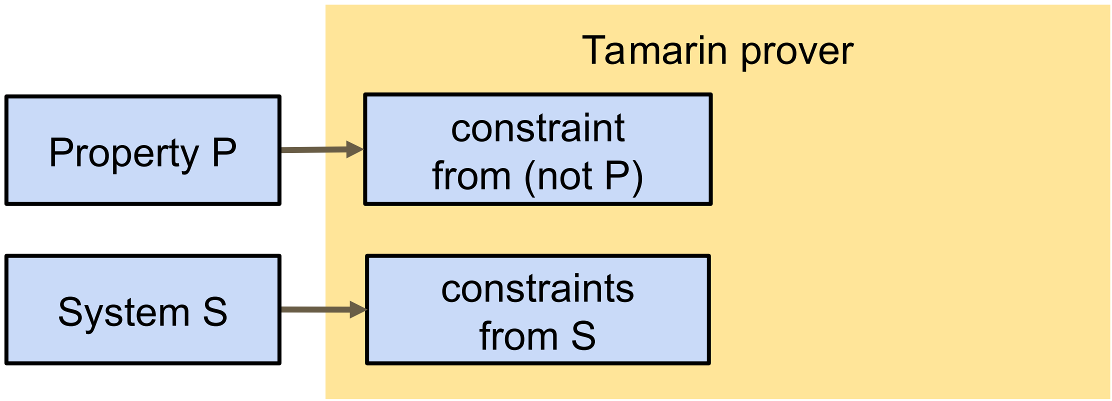
Constraint solver

Tamarin workflow

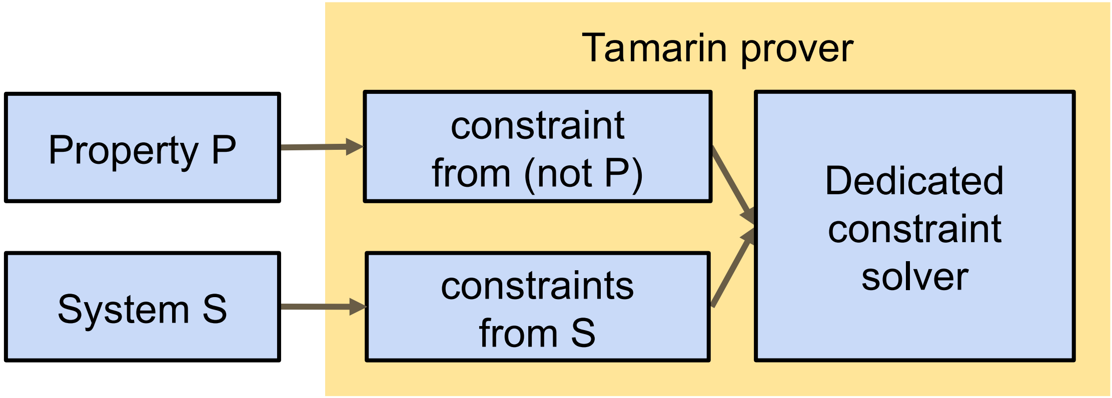
Property P

System S

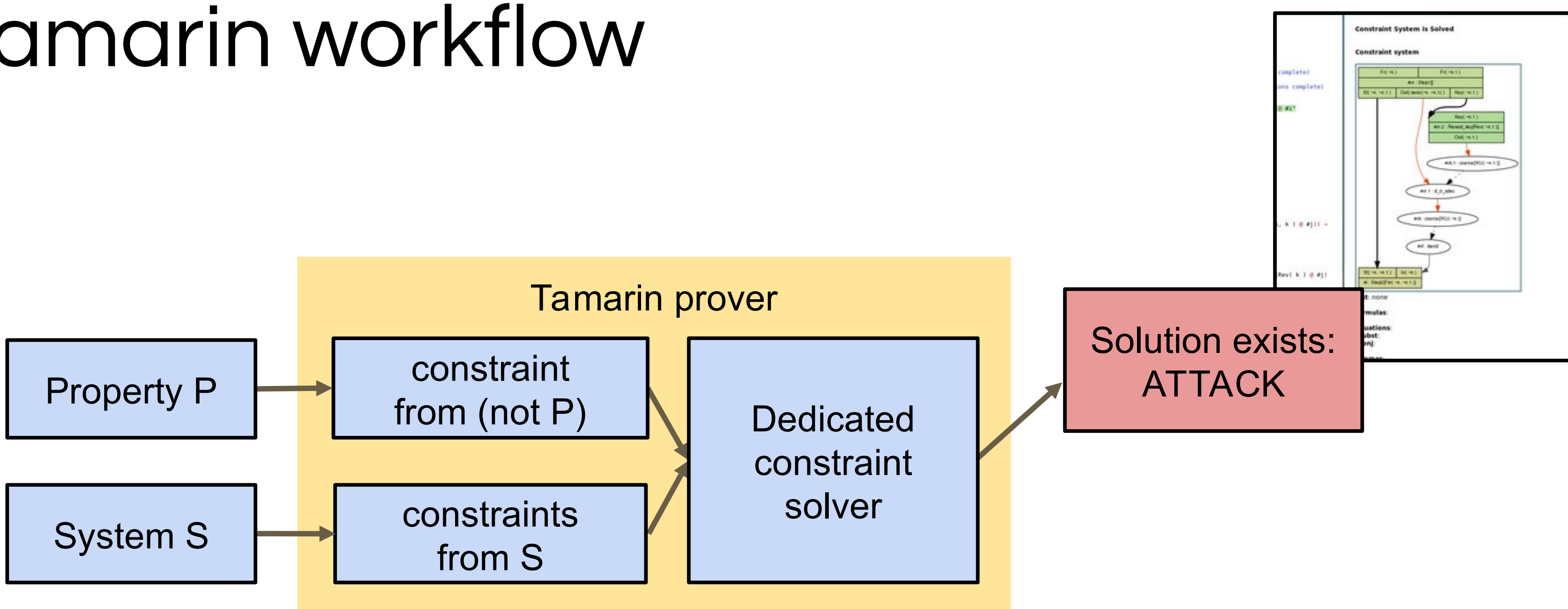
Tamarin workflow



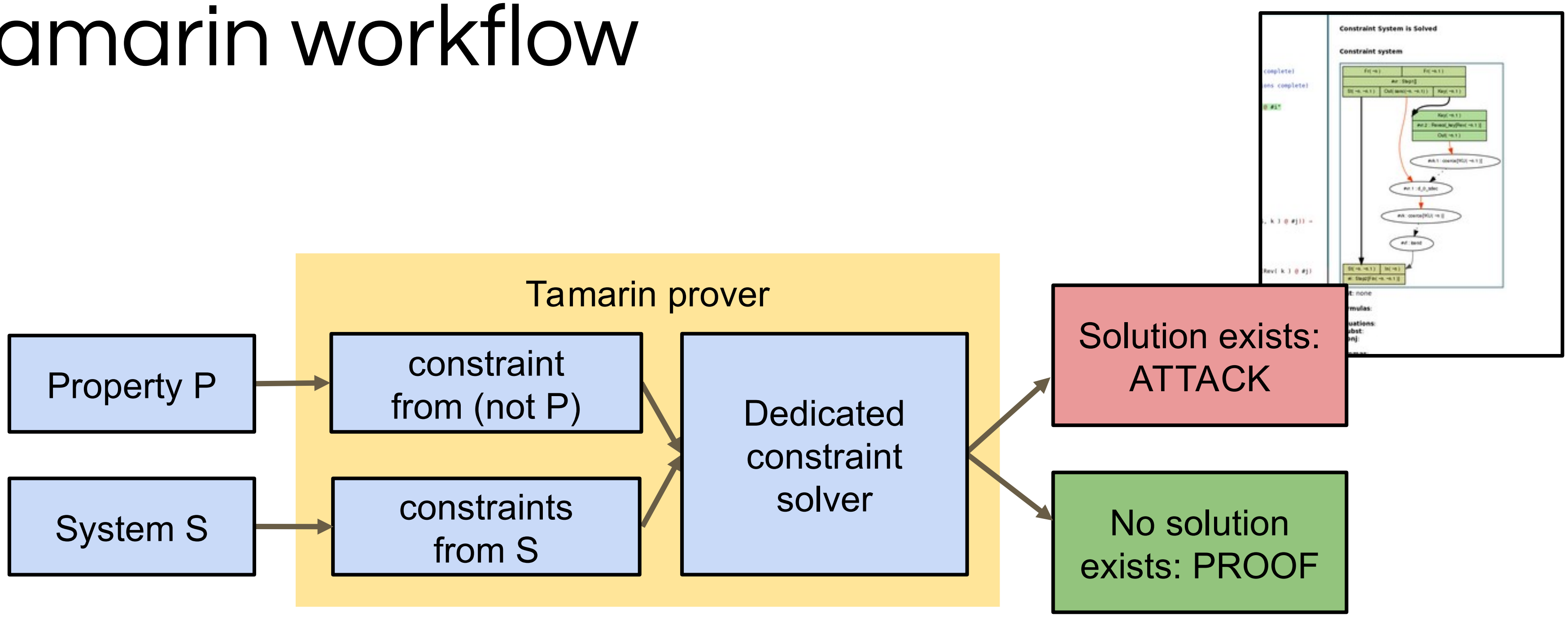
Tamarin workflow



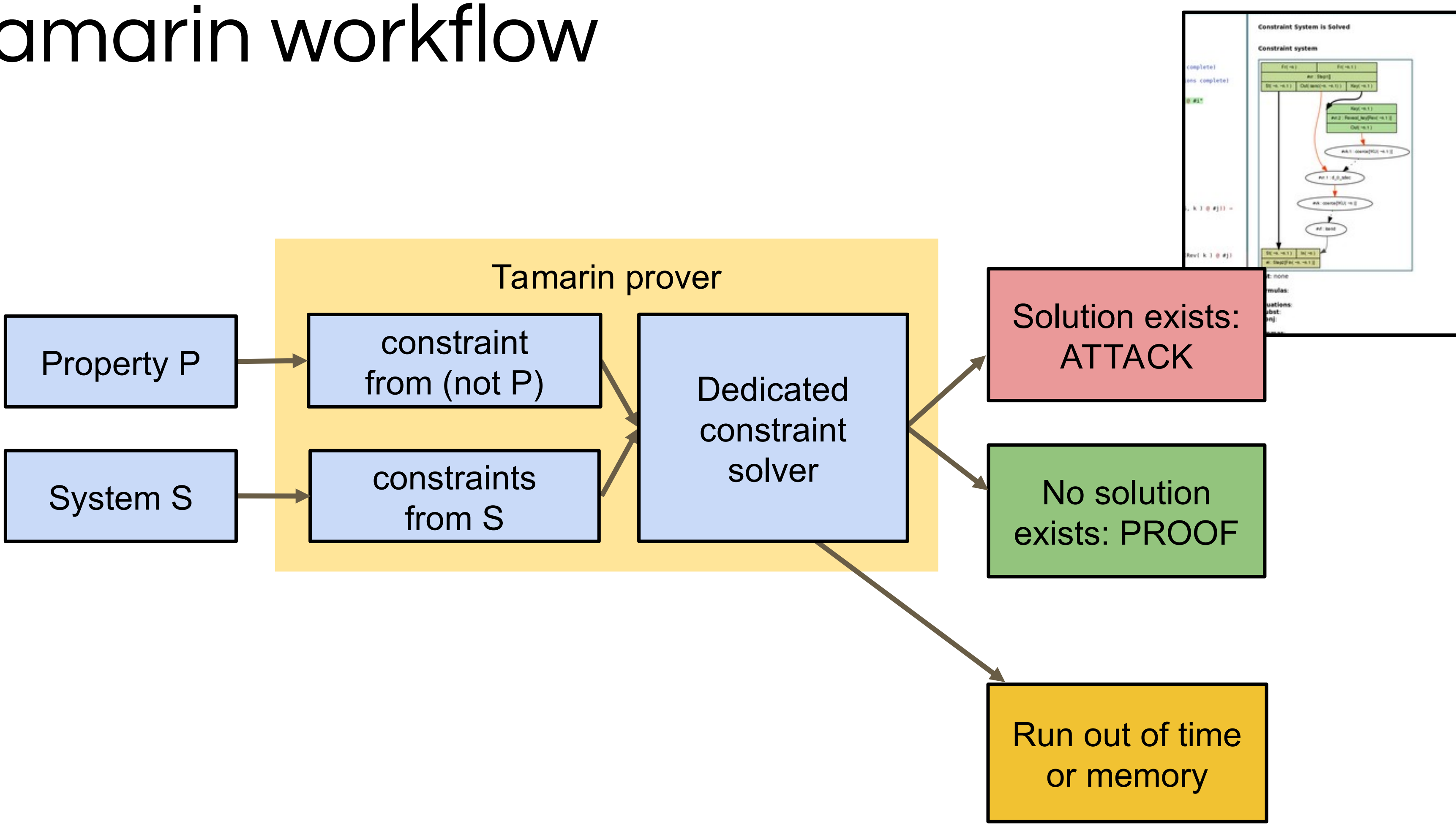
Tamarin workflow



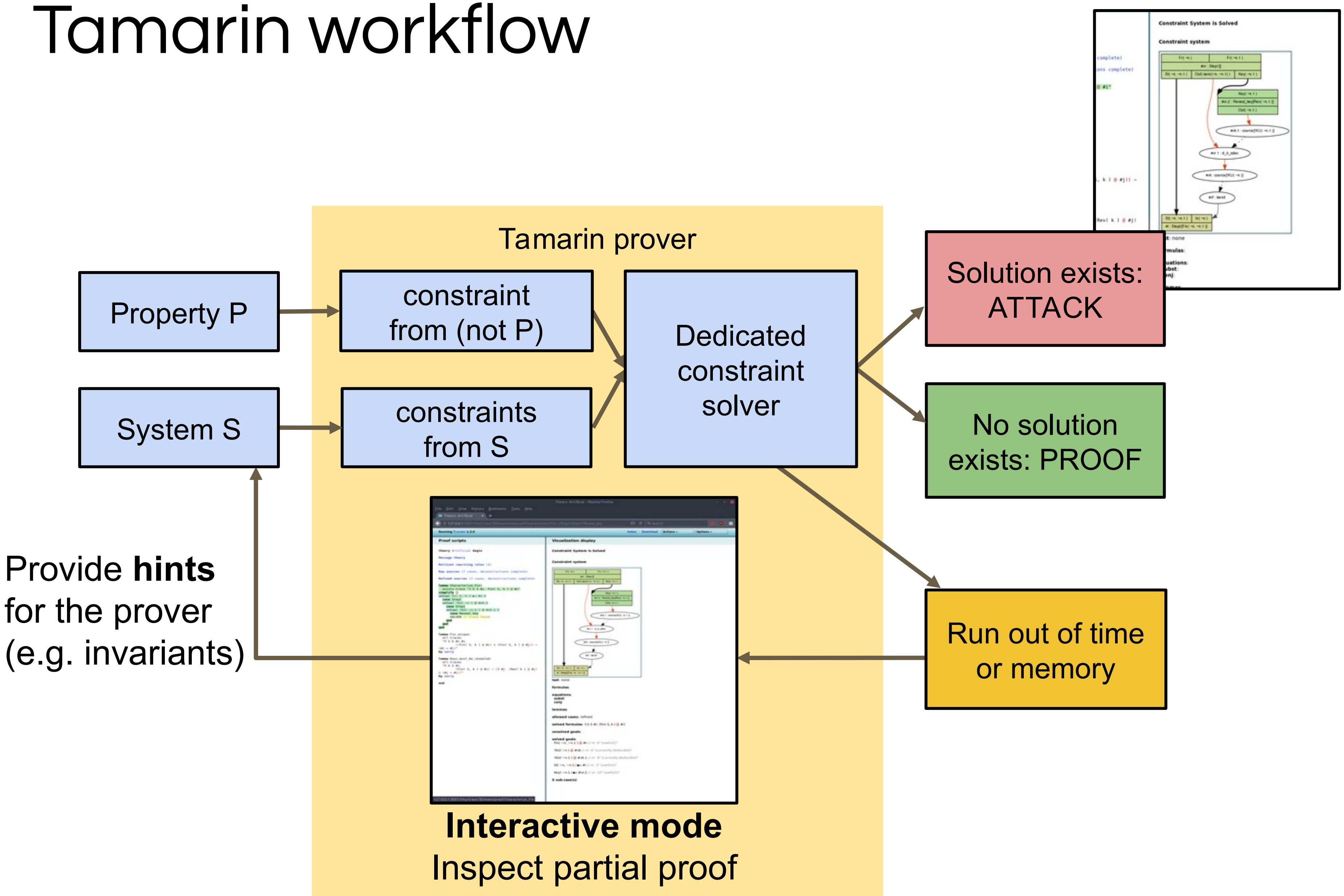
Tamarin workflow



Tamarin workflow



Tamarin workflow



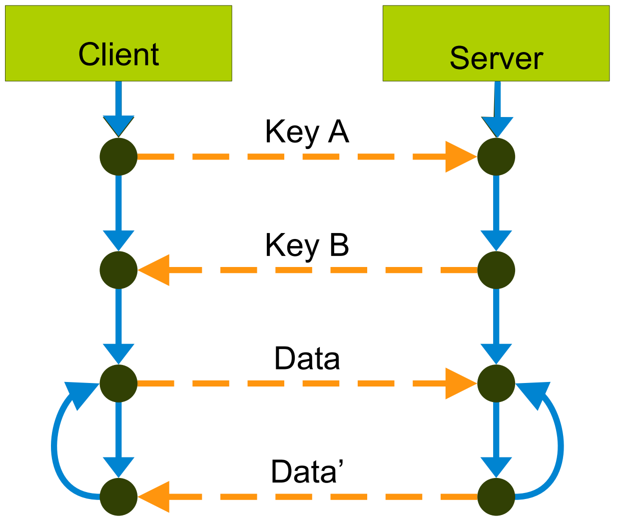
Tamarin: high-level

- **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)

- What we saw:
  - We all use security protocols on a daily basis
  - What Tamarin was built for
  - Who is behind it
  - The main workflow of Tamarin and how users interact with it

Next up: Tamarin’s foundations

Modeling in Tamarin



- **Multiset rewriting**
- 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^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**

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

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

- **Executions**

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

- **Traces**

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

‘c’ constant

Example 1: basic

- **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')]

Example 2: fresh & public

‘c’	constant
~t	t has type fresh
$t	t has type public

- **Rules**
  - rule 1: [Fr(~k)] –[GenKey($A)]→ [Key($A, ~k)]

- **Execution example**
  - [ ]
  - –[GenKey(‘alex’)]→ [Key(‘alex’, k.1)]
  - –[GenKey(‘alex’)]→ [Key(‘alex’, k.2)]
  - –[GenKey(‘blake’)]→ [Key(‘blake’, k.3)]

- **Corresponding trace**
  - [GenKey(‘alex’), GenKey(‘alex’), GenKey(‘blake’)]

‘c’	constant
!F	F is persistent

Example 3: 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'))]

- What we saw:
  - What the underlying model of Tamarin is: multiset rewriting
  - Basic elements: *fresh*, *public*, *constant*
  - How rules define a transition system
  - How the transitions define action traces

Next up: modeling NAXOS

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

$\boxed{\mathrm{I}}$

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

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

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

```
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

$ex_I = h1(esk_I, lk_I)$

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

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

$ex_I = h1(esk_I, lk_I)$

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

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 ) ] --> []
```

- What we saw:
  - The NAXOS protocol
  - How to model it using Tamarin’s rules

Next up: security properties

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

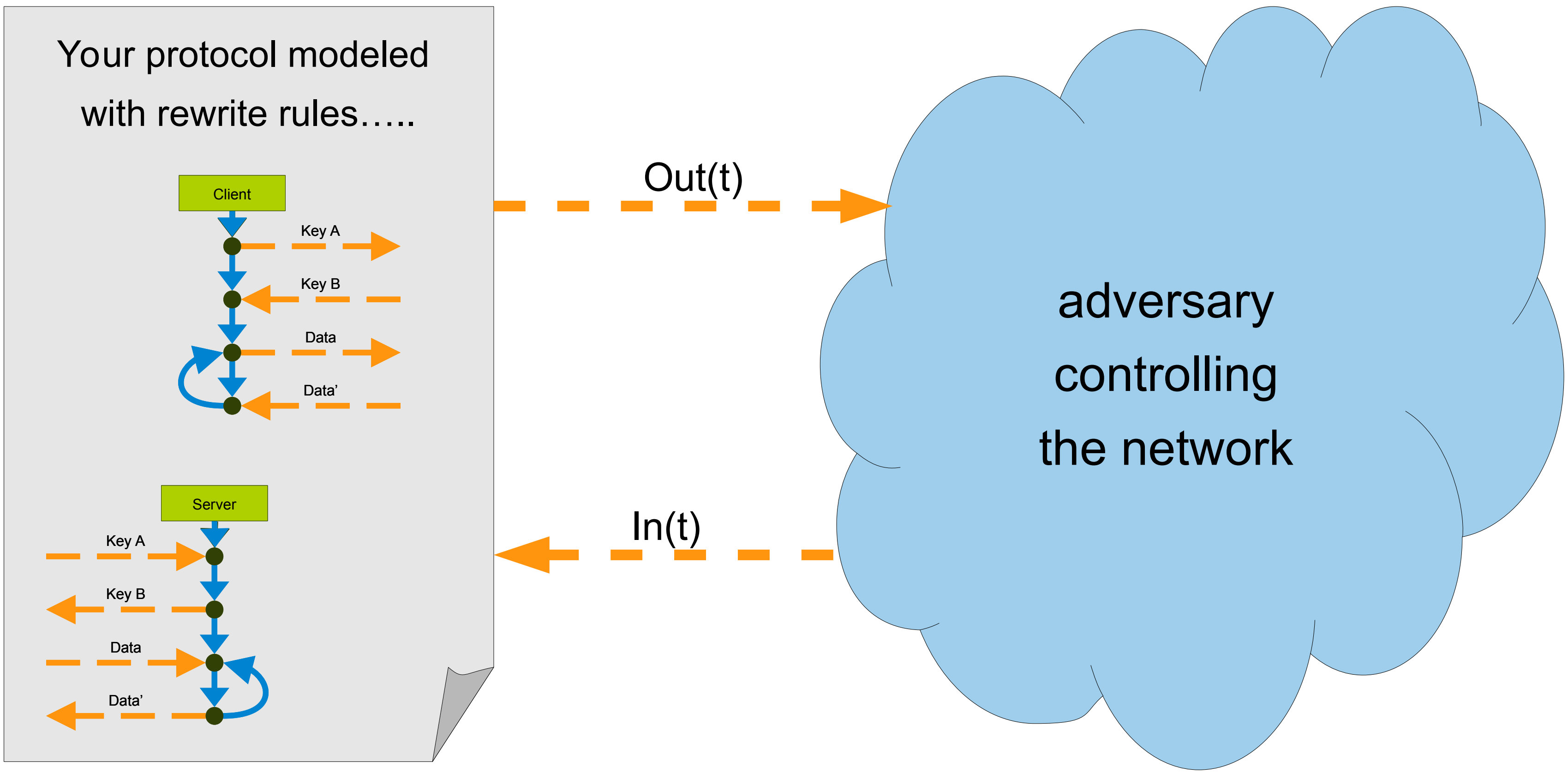
- Adversary can
  - learn **long-term keys**,
  - learn the **randomness** generated in sessions,
  - learn **session keys**
- But only as long as the Test session is *clean*:
  - **No reveal of session key of** Test session or its **matching session**, and
  - No reveal of randomness of Test session 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

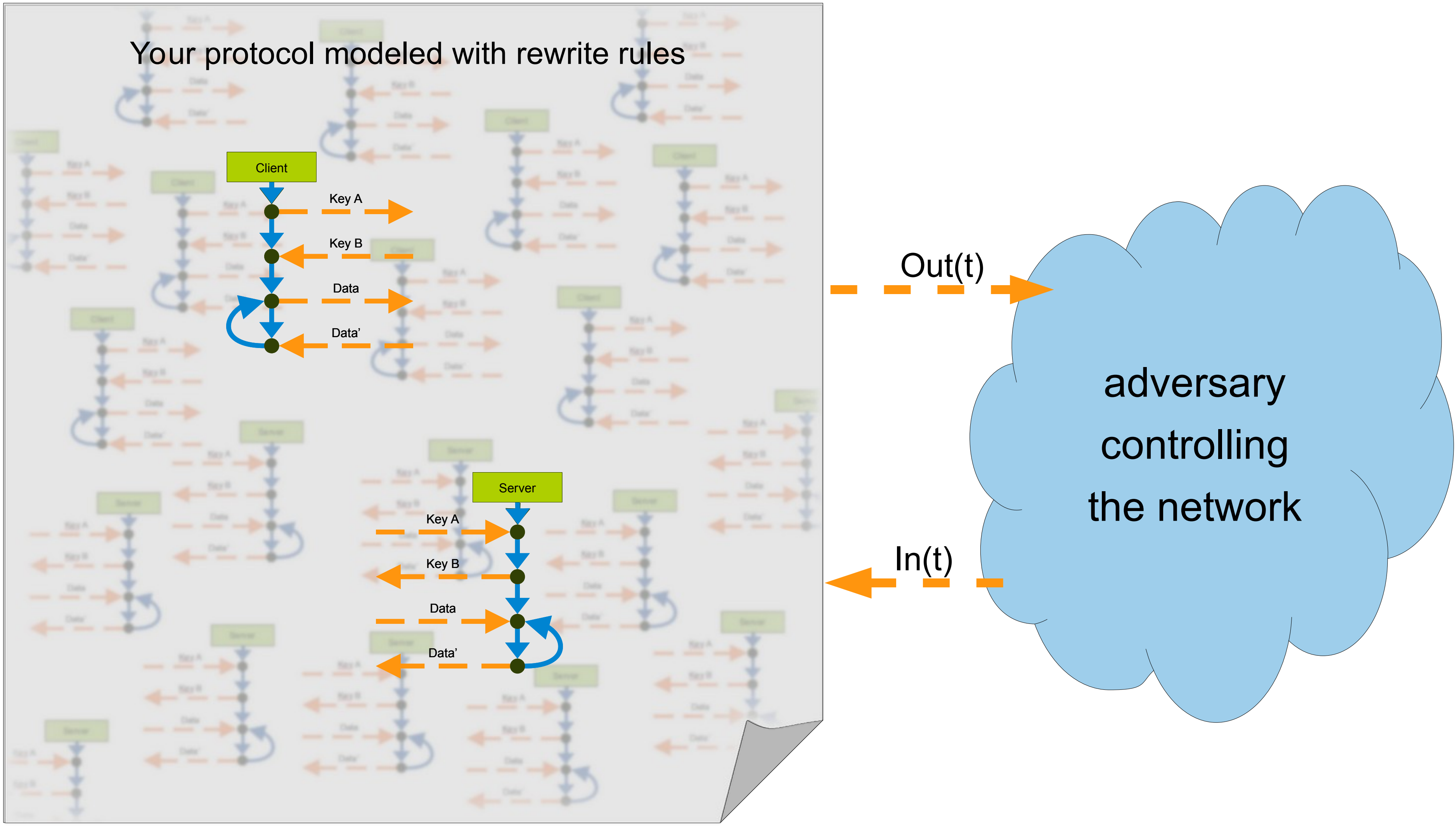
```
Lemma eCK_key_secrecy:
  "(All #i1 #i2 Test A B k. Accept(Test, A, B, k) @ i1
                          & K( k ) @ i2 ==>
  (
      (Ex #i3. SesskRev( Test ) @ i3 )
    | (Ex MatchingSession #i3 #i4 ms.
           ( Sid ( MatchingSession, ms ) @ i3
           & Match( Test, ms ) @ i4)
          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

Tamarin tackles complex interaction with adversary

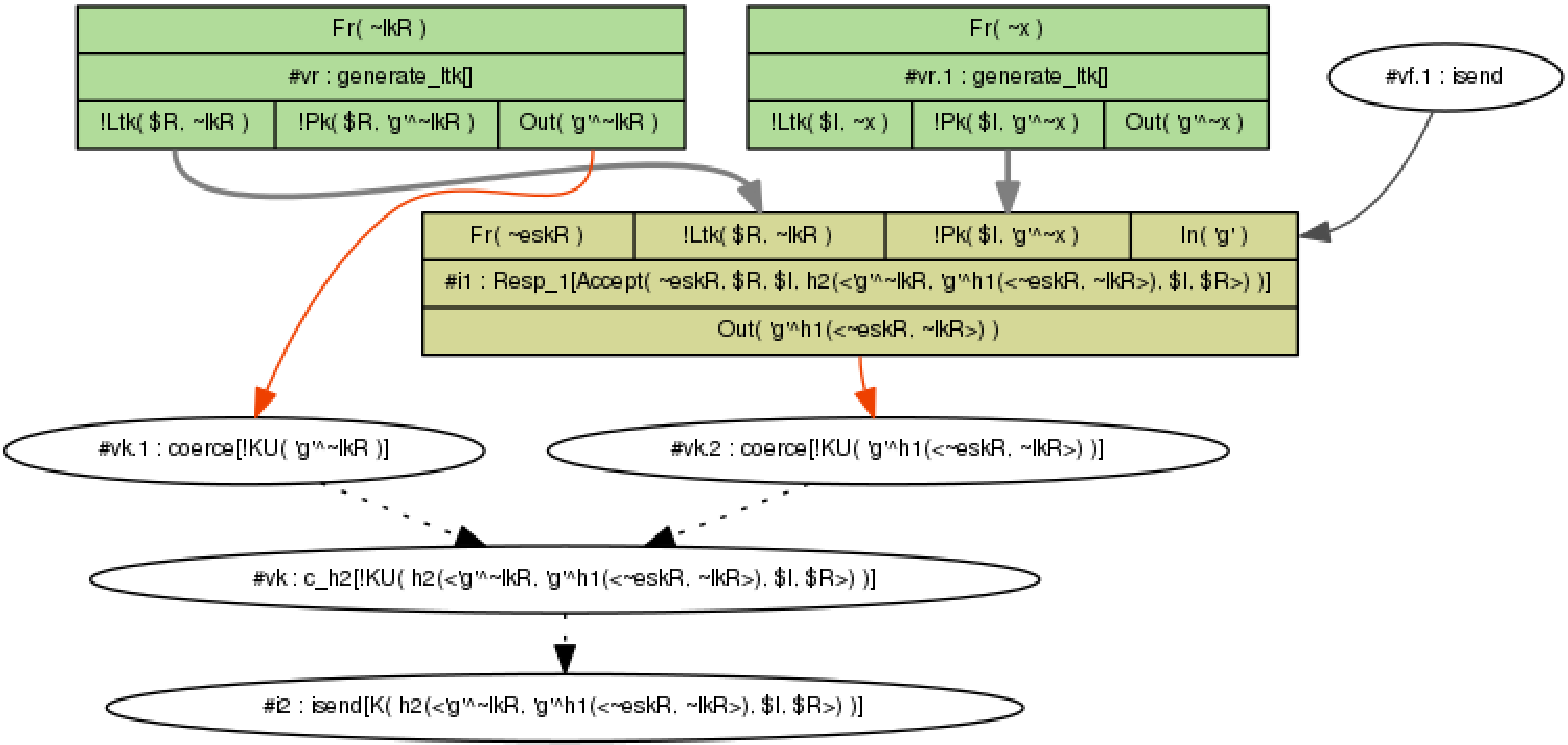


Tamarin tackles complex interaction with adversary



Demo

Reading Tamarin's graphs



- What we saw:
  - How we can write complex security properties in Tamarin
  - How Tamarin deals with counterexamples

Next up: Algorithm intuition

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** (20+)
- 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

- What we saw:
  - Some intuition for Tamarin's algorithm
  - How it resembles constraint solving
  - What this looks like in the graphical user interface

Next up: in practice

How do I know my model is correct?

- **Many ways to model incorrectly**
- Executability
- Break the protocol on purpose
- Look at the chains...
  - (requires an understanding of the algorithm)
- 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

Complexity and termination

- Basic examples
  - Key exchange protocols
  - Signature-based protocols

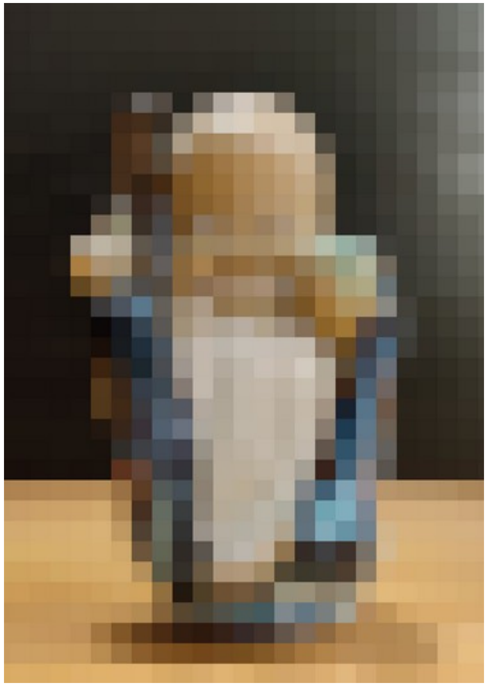
- More complex often needs hints (lemmas)
  - XOR
  - Protocols with complex loops/state machines
  - Diff-equivalence

- What we saw:
  - Challenges in practice include
    - For any formal approach: how and what to model?
    - How do I sanity-check my model?
    - Termination: heuristics & lemmas

Next up: symbolic analysis for cryptographers

Modeling real-world objects

Reality

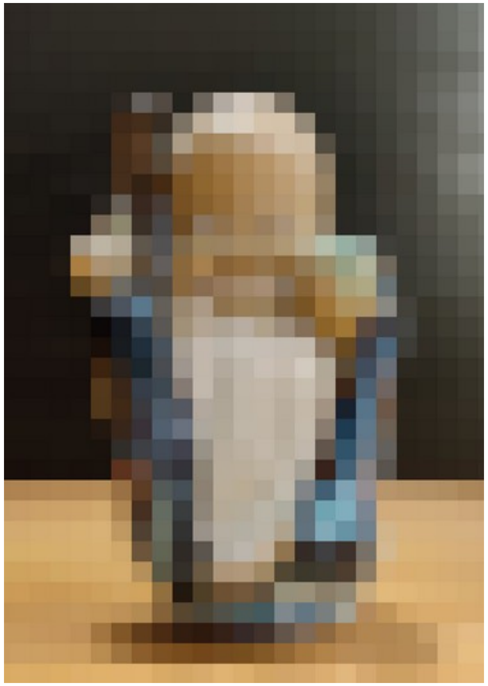


Symbolic

Modeling real-world objects

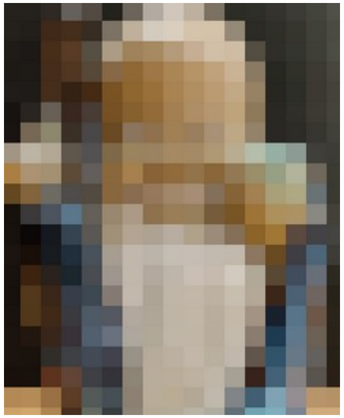
Reality

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** (some of it recent)
  - Negotiation, weak crypto
  - Non-prime order curves
  - DSKS attacks

- What we saw:
  - Some aspects of the relation between symbolic and computational approaches

Finally: where do I go from here?

Other Tamarin features….

- Advanced equational theory support
  - Diffie-Hellman, XOR, multisets, subterm-convergent and more...
- Construct your own proof interactively, and export it so others can verify
- Program your own heuristic
- Diff-equivalence (observational equivalence)
- Restrictions & conditional rules
- Applied-Pi input (through SAPIC integration)

Some recent results

FRESH

- 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

Starter exercise

- Start from files in `tamarin_ex1_part1.zip`
- 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?)

Advanced exercise

- Make sure you have the manual at hand
  - https://tamarin-prover.github.io/manual/index.html
- Try the toy protocol example exercise by Benjamin Kiesl
  - The last questions are challenging and will develop a deeper understanding of Tamarin
  - https://github.com/benjaminkiesl/tamarin_toy_protocol

Which tool should I be using?

- If you are starting out in the domain:
  - Try to find **existing protocol models** that are **close**(ish) to your problem for each tool
    - Pick the tool with the closest existing model, start by adapting that model
- More advanced:
  - Choice can be driven by the **security property** and **threat model** that you are interested in
  - Most approaches give incomparable guarantees; as a consequence they cover different attacks

Tamarin: conclusions

- **Tamarin** offers **many unique features**
  - State machine modeling, flexible properties, equational theories, global state, …
  - Enables automated analysis in areas previously unexplored
  - Many case studies available, from small protocols to large real-world protocols
  - Tamarin *found many new attacks*, impacting several real-world deployments
- Tool and sources are **free**; development on Github
  tamarin-prover.github.io
  - A real team effort!
- **Cas Cremers**


```
email:    cremers@cispa.de              twitter: @CasCremers
website:  https://cispa.saarland/group/cremers/index.html
```