# Advanced modeling, properties, and state space reduction

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

# Overview

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

# 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)**
- System state is a multiset of facts
  - **Initial state** is the empty multiset
  - **Rules** specify the transitions ("moves")
- Rules are of the form:
  - `l --> r` == `l --[ ]-> r`
  - `l --[ a ]-> r`

# 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 and $l \subseteq^{\#} S$

- **Executions**

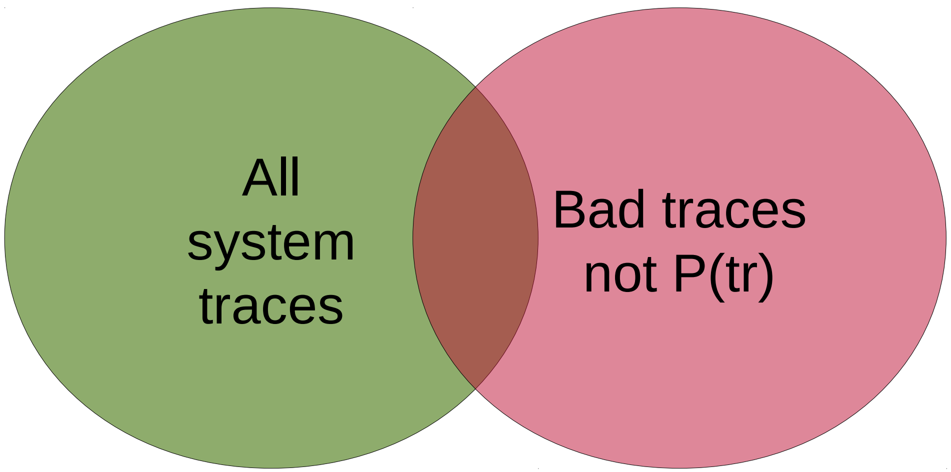
  $\mathrm{Exec}(R) = \{ \emptyset \text{ –}[a_1]\!\to \ldots \text{ –}[a_n]\!\to S_n$
  $\mid \forall n\,.\, \mathrm{Fr}(n)$ appears only once on rhs $\}$

- **Traces**

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

# Trace properties

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




Intersection empty?

# Induction

$$R_{loop} := \left\{ \frac{\mathsf{Fr}(x)}{\mathsf{A}(x)}[\mathsf{Start}(x)],\ \frac{\mathsf{A}(x)}{\mathsf{A}(x)}[\mathsf{Loop}(x)] \right\}$$

- Proof goal: $\forall x\ i.\mathsf{Loop}(x)@i \Rightarrow \exists j.\mathsf{Start}(x)@j$
  - $j < i$ ? Not needed in formula, but will hold
- Naive constraint solving does not work
- Such properties are needed:
  - "Reuse" lemmas
  - "Sources" lemmas

# Constraint solving failure

System $\Gamma_1$:

$\mathsf{Loop}(x)@i \qquad \overbrace{(\forall j.\ \mathsf{Start}(x)@j \Rightarrow \bot))}^{\psi_1}$

System $\Gamma_{1^2}$:

$\psi_1$

$i : \dfrac{\mathsf{A}(x)}{\mathsf{A}(x)}[\mathsf{Loop}(x)]$

System $\Gamma_{1^3}$:

$\psi_1$

$j_1 : \dfrac{\mathsf{A}(x)}{\mathsf{A}(x)}[\mathsf{Loop}(x)]$

$\downarrow$

$i : \dfrac{\mathsf{A}(x)}{\mathsf{A}(x)}[\mathsf{Loop}(x)]$

*ad infinitum* …

⋮

System $\Gamma_{1^2,2}$:

$\psi_1$

$j_1 : \dfrac{\mathsf{Fr}(x)}{\mathsf{A}(x)}[\mathsf{Start}(x)]$

$\downarrow$

$i : \dfrac{\mathsf{A}(x)}{\mathsf{A}(x)}[\mathsf{Loop}(x)]$

$\psi_1$ and $\mathsf{Start}(x)@j_1$
$\Rightarrow$ *contradiction*

System $\Gamma_{1^3,2}$:

$\psi_1$

$j_2 : \dfrac{\mathsf{Fr}(x)}{\mathsf{A}(x)}[\mathsf{Start}(x)]$

$\downarrow$

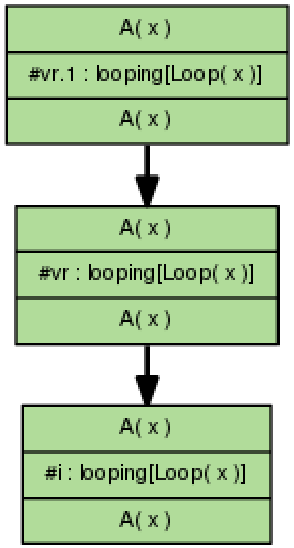
$j_1 : \dfrac{\mathsf{A}(x)}{\mathsf{A}(x)}[\mathsf{Loop}(x)]$

$\downarrow$

$i : \dfrac{\mathsf{A}(x)}{\mathsf{A}(x)}[\mathsf{Loop}(x)]$

$\psi_1$ and $\mathsf{Start}(x)@j_2$
$\Rightarrow$ *contradiction*

# Demo

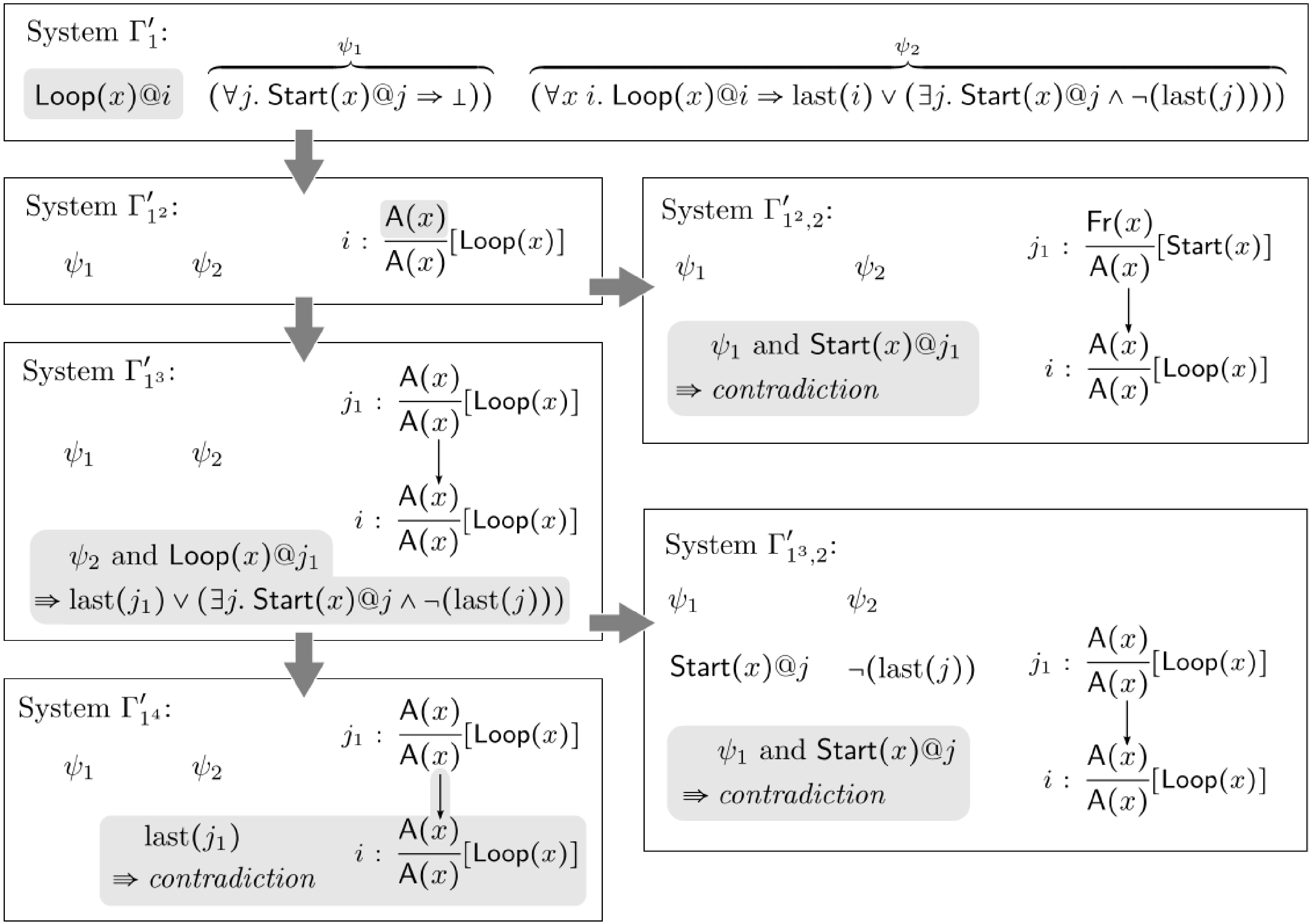

A( x )
#vr.1 : looping[Loop( x )]
A( x )
A( x )
#vr : looping[Loop( x )]
A( x )
A( x )
#i : looping[Loop( x )]
A( x )

# Induction – on time points

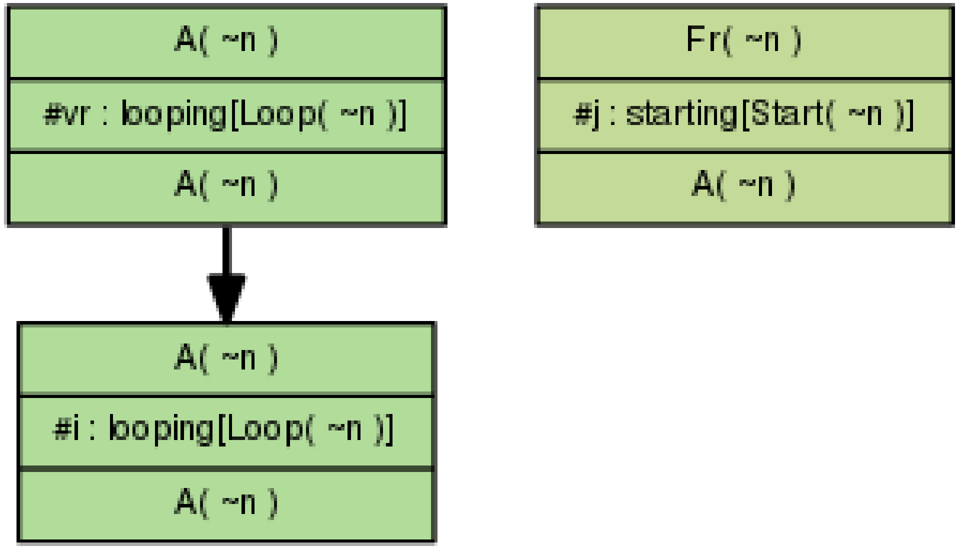
- Informally, induction works on previous slide
- Formally, for IH $\phi$
  1) Check if $\phi$ holds for empty trace
  2) Consider special last rule index on trace
     - Assume $\phi$ holds at all non-last indices, and prove for last

- Added constraint reduction rules for last atoms
- Allows proof of previous example

# Example – solved by induction



# Demo – using induction



# Induction in general

- Required for all “sources” lemmas
- Often required for “reuse” lemmas
- Helps for all looping constructs, used in e.g.:
  - YubiKey
  - TPM
  - PKCS11
  - Group protocols
  - Counters

# State space reduction

Pre-computation

Partial deconstructions

Sources lemmas

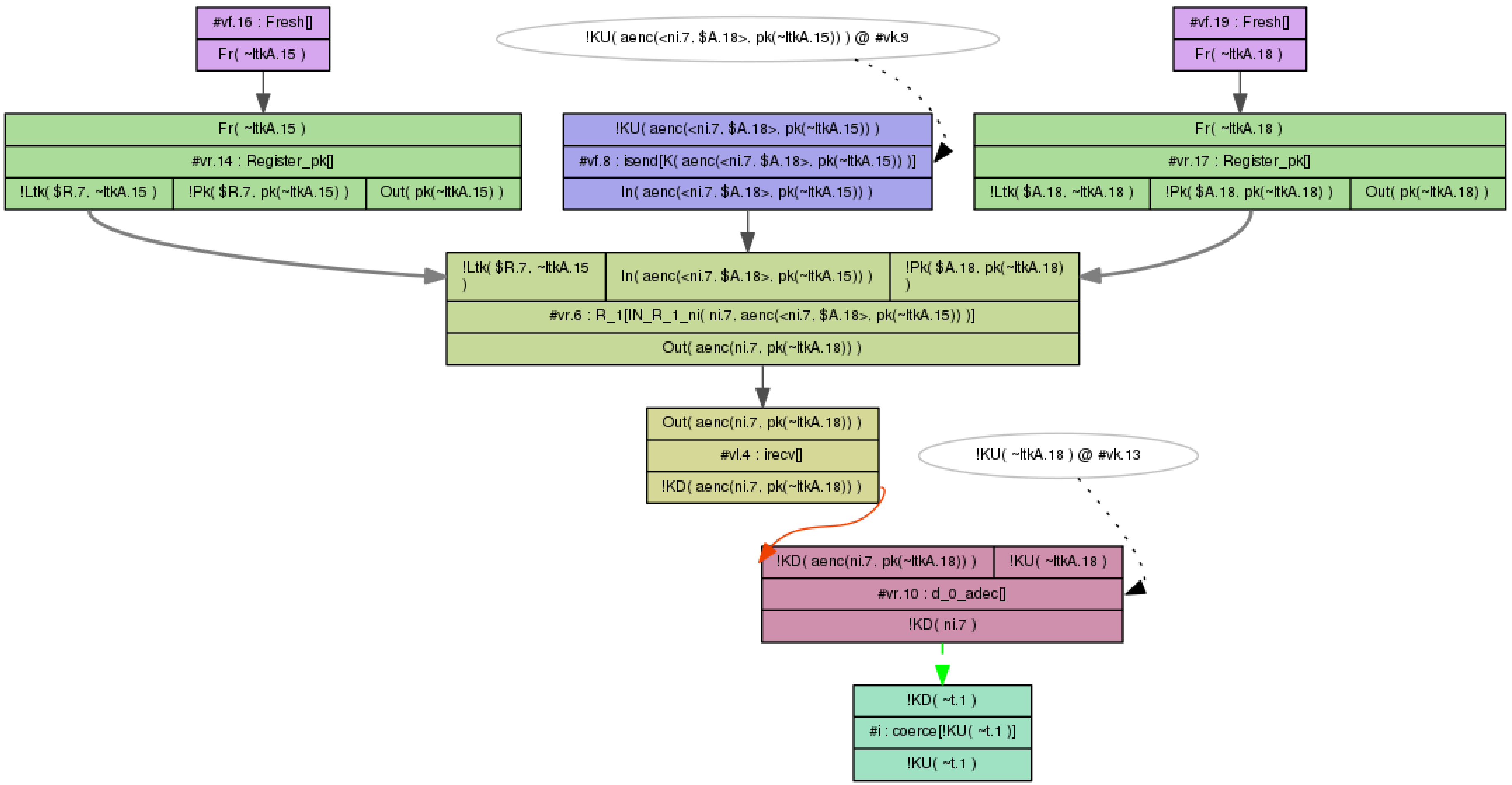
# Precomputation

- Idea: for all facts in rule premises compute their possible sources
- sources are (combinations of) rules yielding such a fact as (part of the) result
- Initial precomputations are called raw sources
- Sometimes these precomputations are incomplete, and give partial deconstructions
- GUI shows both raw and refined sources

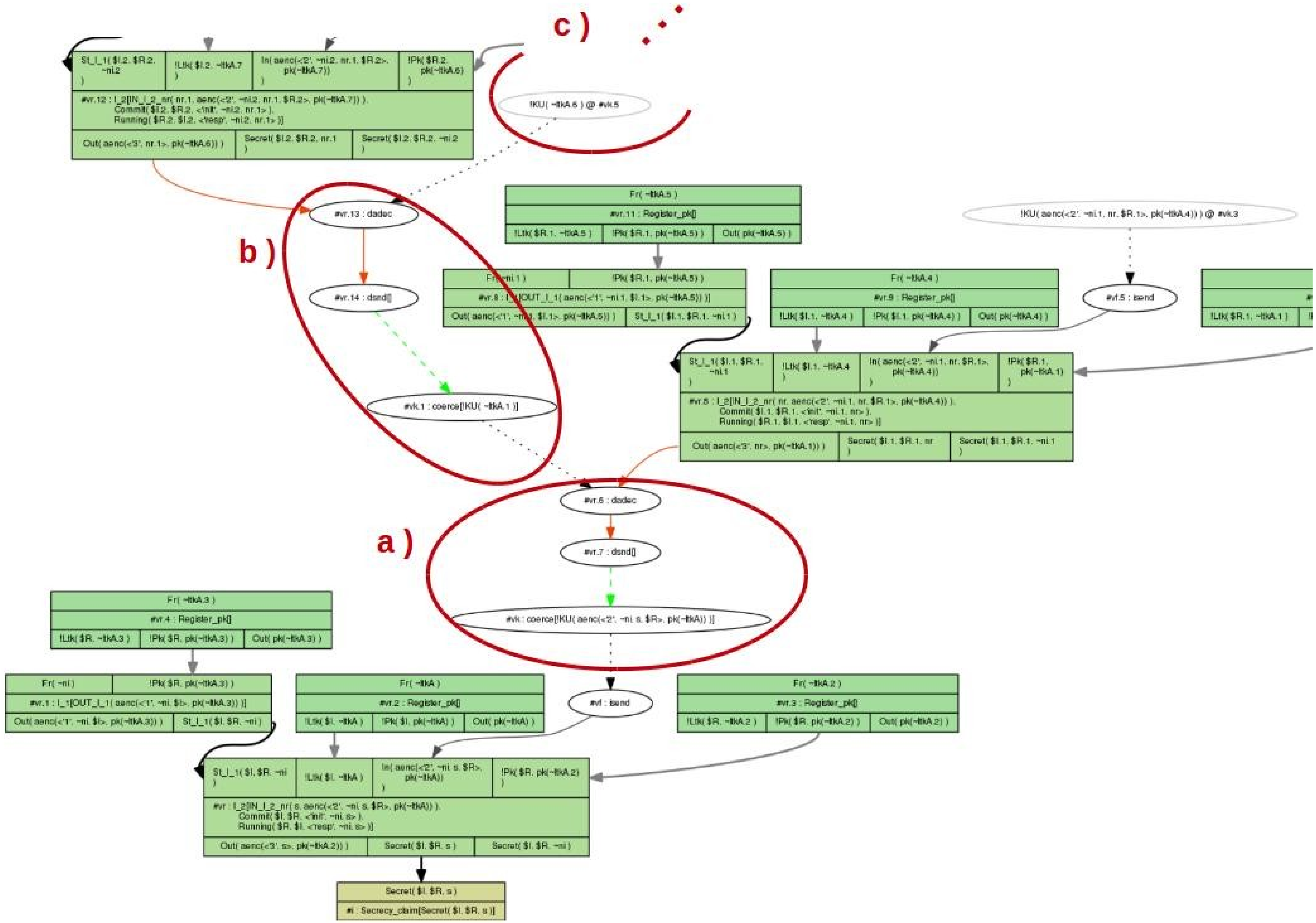
# Demo

```
theory sources begin

Message theory

Multiset rewriting rules (5)

Raw sources (8 cases, 6 partial deconstructions left)

Refined sources (8 cases, deconstructions complete)
```

# Partial deconstruction – derive any value



# See demo for detail



# Partial deconstructions – issues

- Proofs much more complicated
  - Possibly non-termination due to partial deconstructions
- Need to resolve such partial deconstructions
- Claim (and then prove) such deconstructions are not possible, by sources lemma

# Example protocol

```
1. I -> R: {ni,I}pk(R)

2. I <- R: {ni}pk(I)
```

```
rule I_1:
  let m1 = aenc{~ni, $I}pkR  in
    [ Fr(~ni) , !Pk($R, pkR) ]
  --[ OUT_I_1(m1) ]->
    [ Out( m1 ) ]

rule R_1:
  let m1 = aenc{ni, I}pk(ltkR)
      m2 = aenc{ni}pkI    in
    [ !Ltk($R, ltkR) , In( m1 ), !Pk(I, pkI) ]
  --[ IN_R_1_ni( ni, m1 ) ]->
    [ Out( m2 ) ]
```

This looks like a decryption oracle for values ni

# Really? Extract everything?

- Realization: only values actually sent by legitimate party (whose private key must be compromised) or adversary-generated terms
  - which are known to the adversary previously

```
lemma types [sources]:
  " (All ni m1 #i.
      IN_R_1_ni( ni, m1) @ i
      ==>
      ( (Ex #j. K(ni) @ j & j < i)
      | (Ex #j. OUT_I_1( m1 ) @ j) ) ) "
```

# Demo

Problems with partial deconstructions

Sources lemma removes partial deconstructions for refined sources

Automatic proof of sources lemma

# Sources lemmas

- Explain where terms can come from or what their form must be
- Tamarin actions in order:

1) Determine possible sources (raw)
2) Apply sources lemma to raw sources to get refined sources
3) Prove sources lemma WRT raw sources
4) Prove other lemmas WRT refined sources

# Sources lemmas

- Important for termination
- Reduces state space to manageable size
- See more in following hands-on session

# Equational theories

- **Equational theories** are used in symbolic protocol verification to model the **algebraic properties** of the **cryptographic primitives**.
- Example (asymmetric encryption):

```
adec(aenc(m,pk(k),k) = m
```

- Subterm convergent
  - Right-hand side is subterm of left hand side (or constant)
- Built-in: Diffie-Hellman, bilinear pairing, multiset
- Recent extension: any FVP theory
  - Example: blind signatures

# Finite Variant Property

- FVP is a property of an equational theory
  - Theory must be confluent and terminating
  - For all terms there is a bound so that all instantiations of the term reach their normal form in at most bound number of steps.
  - Allows computation of complete set of variants
  - Set of variants represents the term modulo theory
- Any subterm-convergent theory has FVP
- Built-ins are proven to have FVP as well

# Protocol modulo equational theory

- Inefficient to compute protocol execution modulo equational theory
- Compute variants of rules modulo equational theory
- Adversary: Construction and deconstruction rules
  - Construction rules allow adversary to apply any operator
  - Deconstruction rules represent applying those operators where the result changes modulo the equational theory
  - e.g.: applying decryption operator to encrypted term with correct key
- Adversary rules are automatically derived

# Message deduction

The adversary is specified using multiset rewrite rules

$$\mathrm{MD} = \left\{ \begin{array}{llll} \dfrac{\mathsf{Out}(x)}{\mathsf{K}(x)} & \dfrac{\mathsf{K}(x)}{\mathsf{In}(x)}[\mathsf{K}(x)] & \dfrac{\mathsf{Fr}(x:fr)}{\mathsf{K}(x:fr)} & \dfrac{}{\mathsf{K}(x:pub)} \\ \\ \multicolumn{4}{l}{\dfrac{\mathsf{K}(x_1)\ldots\mathsf{K}(x_k)}{\mathsf{K}(f(x_1,\ldots,x_k))} \text{ for all } f \in \Sigma} \end{array} \right\}$$

Example:

$$\frac{\mathsf{Out}(\ \mathtt{aenc}(m, \mathtt{pk}(k))\ )}{\mathsf{K}(\ \mathtt{aenc}(m, \mathtt{pk}(k))\ )} \qquad \frac{\mathsf{Out}(k)}{\mathsf{K}(k)}$$

$$\frac{\mathsf{K}(\ \mathtt{aenc}(\overset{\downarrow}{m}, \mathtt{pk}(k))\ ) \qquad \mathsf{K}(\overset{\downarrow}{k})}{\mathsf{K}(\ \mathtt{dec}(\mathtt{aenc}(m, \mathtt{pk}(k)), k)\ )}$$

$$\frac{\mathsf{K}(\ \mathtt{dec}(\mathtt{aenc}(\overset{\downarrow}{m}, \mathtt{pk}(k)), k)\ )}{\mathsf{In}(\ \mathtt{dec}(\mathtt{aenc}(m, \mathtt{pk}(k)), k)\ )}[\mathsf{K}(\ \mathtt{dec}(\mathtt{aenc}(m, \mathtt{pk}(k)), k)\ )]$$

# Message deduction

The adversary is specified using multiset rewrite rules

$$\mathrm{MD} = \left\{ \begin{array}{llll} \dfrac{\mathsf{Out}(x)}{\mathsf{K}(x)} & \dfrac{\mathsf{K}(x)}{(x)}[\mathsf{K}(x)] & \dfrac{\mathsf{Fr}(x:fr)}{\mathsf{K}(x:fr)} & \dfrac{}{\mathsf{K}(x:pub)} \\ \\ \multicolumn{4}{l}{\dfrac{\mathsf{K}(x_1)\ldots\mathsf{K}(x_k)}{\mathsf{K}(f(x_1,\ldots,x_k))} \text{ for all } f \in \Sigma} \end{array} \right\}$$

Example:

$$\dfrac{\dfrac{\mathsf{Out}(\ \mathtt{aenc}(m, \mathtt{pk}(k))\ )}{\mathsf{K}(\ \mathtt{aenc}(m, \mathtt{pk}(k))\ )} \qquad \dfrac{\mathsf{Out}(k)}{\mathsf{K}(k)}}{\dfrac{\mathsf{K}(\ \mathtt{aenc}(m, \mathtt{pk}(k))\ ) \qquad \mathsf{K}(k)}{\mathsf{K}(m)}}$$

$$\frac{\mathsf{K}(m)}{\mathsf{In}(m)}[\mathsf{K}(m)] \qquad \text{as } \mathtt{dec}(\mathtt{aenc}(m, \mathtt{pk}(k)), k) = m$$

# Prevent loops and redundant derivation

## Split adversary knowledge into $\mathrm{K}^{\Uparrow}$ and $\mathrm{K}^{\Downarrow}$

$$\frac{\mathsf{K}(\langle a, b\rangle)}{\mathsf{K}(a)}$$

$$\downarrow$$

$$\frac{\mathsf{K}(a) \quad \mathsf{K}(c)}{K(\langle a, c\rangle)}$$

$$\downarrow$$

$$\frac{\mathsf{K}(\langle a, c\rangle)}{\mathsf{K}(a)}$$

$$\downarrow$$

$$\frac{\mathsf{K}(a) \quad \mathsf{K}(d)}{\mathsf{K}(\langle a, d\rangle)}$$

$$\frac{\mathsf{K}^{\Uparrow}(a) \quad \mathsf{K}^{\Uparrow}(c)}{\mathsf{K}^{\Uparrow}(\langle a, c\rangle)}$$

$$\cancel{\downarrow}$$

$$\frac{\mathsf{K}^{\Downarrow}(\langle a, c\rangle)}{\mathsf{K}^{\Downarrow}(a)}$$

$$\frac{\mathsf{K}^{\Downarrow}(\langle a, b\rangle)}{\mathsf{K}^{\Downarrow}(a)}$$

$$\downarrow$$

$$\frac{\mathsf{K}^{\Downarrow}(a)}{\mathsf{K}^{\Uparrow}(a)}$$

$$\downarrow$$

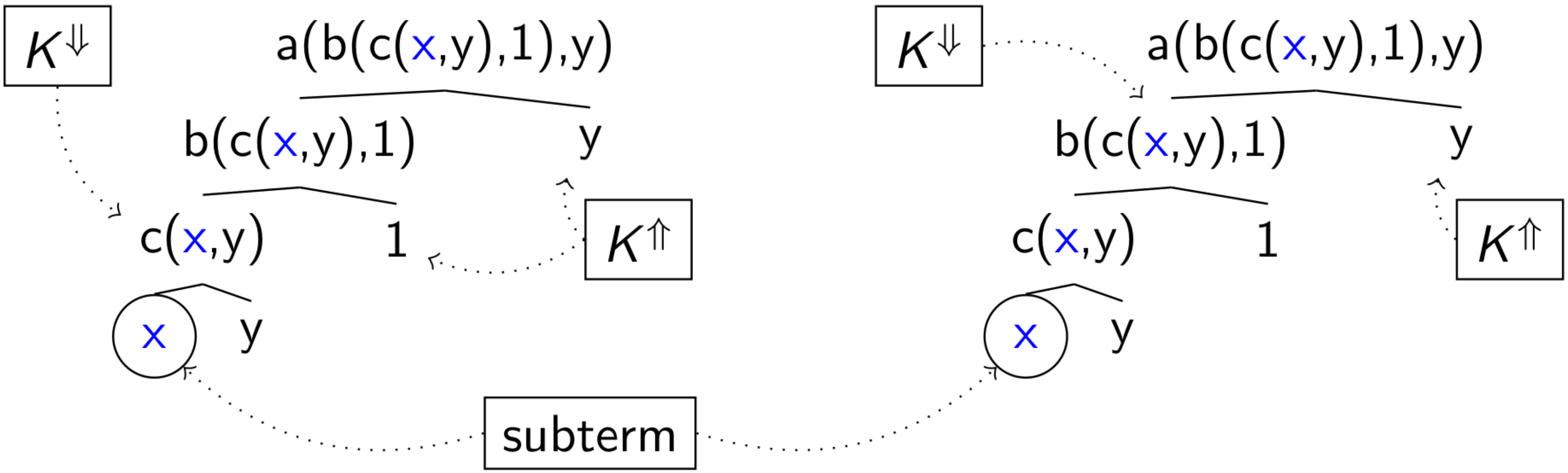
$$\frac{\mathsf{K}^{\Uparrow}(a) \quad \mathsf{K}^{\Uparrow}(d)}{\mathsf{K}^{\Uparrow}(\langle a, d\rangle)}$$

# Normal deduction – equivalent to MD

$$ND = \left\{ \begin{array}{c}
\dfrac{\mathsf{Out}(x)}{\mathsf{K}^{\Downarrow}(x)} \\[2em]
\dfrac{\mathsf{K}^{\Downarrow}(\langle x, y\rangle)}{\mathsf{K}^{\Downarrow}(x)} \quad \dfrac{\mathsf{K}^{\Downarrow}(\langle x, y\rangle)}{\mathsf{K}^{\Downarrow}(y)} \quad \dfrac{\mathsf{K}^{\Downarrow}(aenc(m, pk(k))) \qquad \mathsf{K}^{\Uparrow}(k)}{\mathsf{K}^{\Downarrow}(m)} \\[2em]
\dfrac{\mathsf{K}^{\Downarrow}(x)}{\mathsf{K}^{\Uparrow}(x)} \qquad \dfrac{\mathsf{Fr}(x : fr)}{\mathsf{K}^{\Uparrow}(x : fr)} \qquad \dfrac{}{\mathsf{K}^{\Uparrow}(x : pub)} \\[2em]
\dfrac{\mathsf{K}^{\Uparrow}(x) \qquad \mathsf{K}^{\Uparrow}(y)}{\mathsf{K}^{\Uparrow}(\langle x, y\rangle)} \qquad \dfrac{\mathsf{K}^{\Uparrow}(p)}{\mathsf{K}^{\Uparrow}(fst(p))} \qquad \dfrac{\mathsf{K}^{\Uparrow}(p)}{\mathsf{K}^{\Uparrow}(snd(p))} \\[2em]
\dfrac{\mathsf{K}^{\Uparrow}(m) \qquad \mathsf{K}^{\Uparrow}(k)}{\mathsf{K}^{\Uparrow}(aenc(m, k))} \qquad \dfrac{\mathsf{K}^{\Uparrow}(c) \qquad \mathsf{K}^{\Uparrow}(k)}{\mathsf{K}^{\Uparrow}(adec(c, k))} \qquad \dfrac{\mathsf{K}^{\Uparrow}(k)}{\mathsf{K}^{\Uparrow}(pk(k))} \\[2em]
\dfrac{\mathsf{K}^{\Uparrow}(x)}{\mathsf{In}(x)}[\mathsf{K}(x)]
\end{array} \right\}$$

# Deconstruction rules for subterm-convergent equations

$$\boxed{a(b(c(x, y), 1), y) \to x}$$

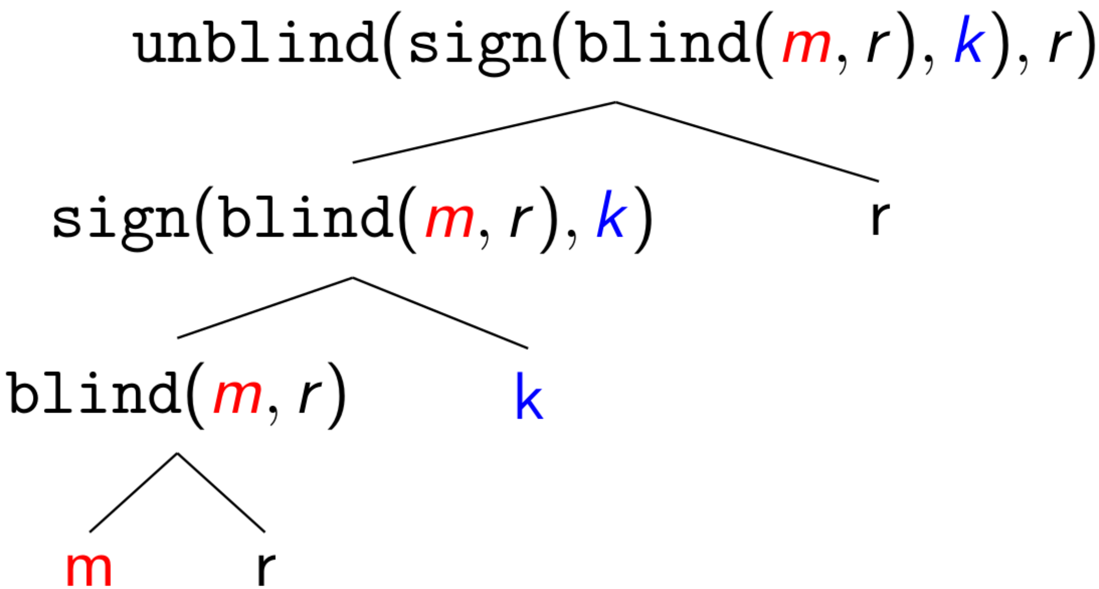



Yields two deconstruction rules:

$$\frac{\mathsf{K}^{\Downarrow}(c(x,y)) \quad \mathsf{K}^{\Uparrow}(1) \quad \mathsf{K}^{\Uparrow}(y)}{\mathsf{K}^{\Downarrow}(x)} \qquad \frac{\mathsf{K}^{\Downarrow}(b(c(x,y),1)) \quad \mathsf{K}^{\Uparrow}(y)}{\mathsf{K}^{\Downarrow}(x)}$$

# General deconstruction rule example

$$\boxed{\mathtt{unblind}(\mathtt{sign}(\mathtt{blind}(m, r), k), r) \rightarrow \mathtt{sign}(m, k)}$$




Yields two deconstruction rules (2nd twice):

$$\frac{\mathsf{K}^{\Downarrow}(blind(m,r)) \quad \mathsf{K}^{\Uparrow}(k) \quad \mathsf{K}^{\Uparrow}(r)}{\mathsf{K}^{\Downarrow}(sign(m,k))} \qquad \frac{\mathsf{K}^{\Downarrow}(sign(blind(m,r),k)) \quad \mathsf{K}^{\Uparrow}(r)}{\mathsf{K}^{\Downarrow}(sign(m,k))}$$

# Protocol by Fujioka, Okamoto, and Ohta [FOO92]

Protocol combines

- Blind signatures

  `unblind(sign(blind(x,r),k),r) = sign(x,k)`

- **Commitments:** `open(commit(v,r),r) = v`

to **ensure**:

- vote privacy (equivalence property)
- eligibility (trace property)

```
"All vote #j. VotePublished (vote) @ j ==>
 (Ex r b #i. Registered(blind(commit(vote,r),b)) @ i
            & #i < #j )"
```

# FOO92

Runs in three **phases**:

- Eligibility check
- Voting
- Counting

**Authorities**:

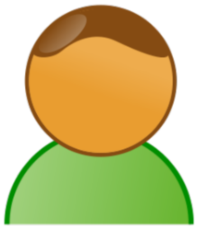
- Administrator
- Collector

**Assumption**:

- Anonymous channel to the collector

# Eligibility check

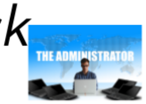
Bob



Administrator


sign(blind(commit(  $, r_1^B), r_2^B), sk$  ), 

Verify signature & eligibility

sign(blind(commit(  $, r_1^B), r_2^B), sk$  )

Verify signature and unblind:

sign(commit(  $, r_1^B), sk$  )

# Demo

Message theory

Multiset rewriting rules

Heuristics vs user-guided

# Restrictions

- Restrictions exclude undesired traces
  - Take care not to exclude attacks!
- Safe to use for certain checks:
  - Equality
  - Inequality
  - LessThan
  - GreaterThan
  - OnlyOnce
- Use same format as lemmas
- Essentially: Conditional Rewriting

# Restriction Example

- **`restriction once:`**

  **`“All #i #j. OnlyOnce()@#i & OnlyOnce()@#j ==> #i=#j”`**

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

- **Execution removed by restriction**
    - [ ]
    - –[ OnlyOnce() ]→ [ A('5')]
    - –[ OnlyOnce() ]→ [ A('5'), A('5') ]
    - –[ Step('5')]→ [ A('5'), B('5') ]

- **Execution still allowed**
    - [ ]
    - –[ Init()       ]→ [ A('5'), A('5') ]
    - –[ Step('5')]→ [ A('5'), B('5') ]

# Restriction Example 2

- **`restriction InEq:`**

  **`“All x #i. Neq(x,x)@#i ==> F”`**
- **Rules**
  - rule 1: [ ]  –[ A1() ]→ [ A('1') ]
  - rule 2: [ ]  –[ A2() ]→ [ A('2') ]
  - rule 3: [ A(x), A(y) ] –[ Neq(x,y) ]→ [ B(x,y) ]
- **Execution removed by restriction – valid without restriction**
  - [ ]
  - –[ A1() ]→ [ A('1')]
  - –[ A1() ]→ [ A('1'), A('1') ]
  - –[ Neq('1','1') ]→ [ B('1','1') ]
- **Execution allowed**
  - [ ]
  - –[ A1() ]→ [ A('1')]
  - –[ A2() ]→ [ A('1'), A('2') ]
  - –[ Neq('1','2') ]→ [ B('1','1') ]

# Observational equivalence

Two types of properties:

- Trace properties
  - (Weak) secrecy as reachability
  - Authentication as correspondence

- Observational equivalence $\approx$

# Why observational equivalence?

- Consider classic **Dolev-Yao** adversary for deterministic public-key encryption:

$$\frac{enc(x, pk(k)) \quad k}{x}$$

- Adversary can only decrypt if he knows the secret key

Consider a simple voting system:

- Voter chooses `v="Yes"` or `v="No"`
- Encrypt v using server's public key `pk(k)`**:**

  `c = enc(v,pk(k))`
- Send `c` to server

Is the vote secret?

- Dolev-Yao: **Yes**, adversary does not know server's secret key
- Reality: **No**, encryption is deterministic and there are only two choices
  - **Attack**: encrypt `"Yes"`, and compare to `c`

# Observational equivalence vs reachability

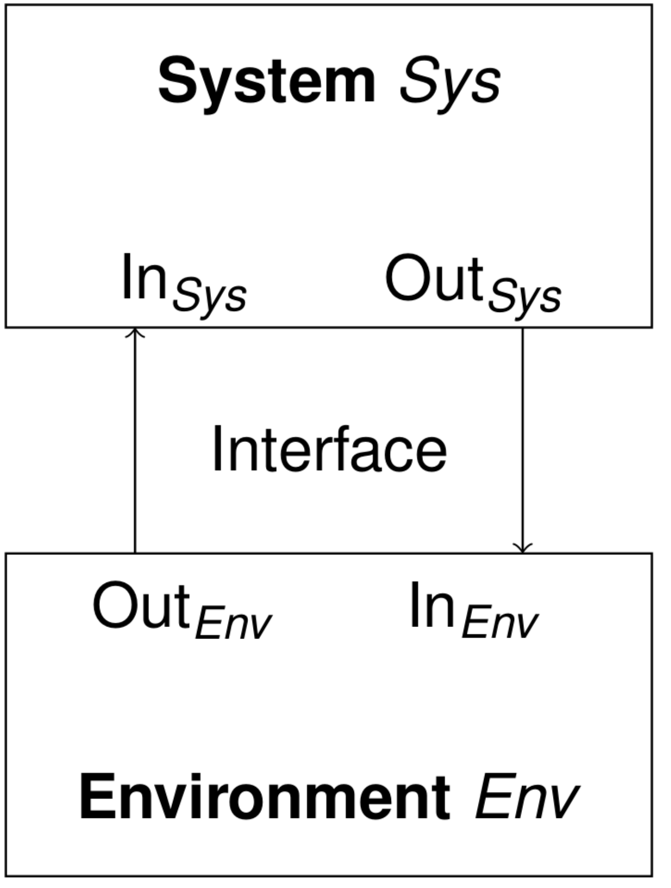
- **Reachability**-based (weak) secrecy is insufficient
- **Stronger** notion: adversary cannot distinguish
  - a system where the voter votes "`Yes`" from
  - a system where the voter votes "`No`"
- **Observational equivalence** between two systems
- Can be used to express
  - Strong secrecy
  - Privacy notions

# Running example

- **Auction** system for a **shout-out auction**
- Property: **strong secrecy** of bids
- Property violated:
  - Broadcast bid (e.g., `A` or `B`)
  - Send **"**`A`**"** in first system
  - Send **"**`B`**"** in second system
  - Observer knows if he is observing first or second system
- Property holds using **shared symmetric key**:
  - Shared symmetric key k between bidder and auctioneer
  - Send **"**$\{A\}_k$**"** in first system
  - Send **"**$\{B\}_k$**"** in second system
  - Observer has no access to k, does not know which system he observes

# System and environment

- We separate **environment** and **system**
  - System: agents running according to protocol
  - Environment: adversary acting according to its capabilities
- Environment can observe:
  - Output of the system
  - If system reacts at all



# Defining observational equivalence

- **Two** system **specifications** given as set of rules
  - One rule per role action (send/receive)
  - Running example shout-out auction:

System 1: $\dfrac{}{\mathrm{Out}_{Sys}(A)}$ System 2: $\dfrac{}{\mathrm{Out}_{Sys}(B)}$

- Interface and environment/adversary rule(s):

$$\frac{\mathrm{Out}_{Sys}(X)}{\mathrm{In}_{Env}(X)} \qquad \frac{\mathrm{Out}_{Env}(X)}{\mathrm{In}_{Sys}(X)} \qquad \frac{\mathrm{In}_{Env}(X) \quad K(X)}{\mathrm{Out}_{Env}(true)}$$

  - Last rule models comparison by the adversary
- Each specification yields a labeled transition system
- Observational equivalence is a kind of **bisimulation** accounting for the adversaries' **viewpoint** and **capabilities**

# Diff terms

- General definitions of observational equivalence **difficult** to verify: requires inventing simulation relation
- Idea: **specialize** for cryptographic protocols
  - Consider strong bid secrecy:
    - both systems differ in **secret bid** only, i.e.,
    - both specifications contain **same rule(s)**, which differ only in **some terms**
  - Exploit this similarity in description and proof
- Approach: two systems described by one specification – using **diff**-terms
  - Running example

$$\overline{Out_{Sys}(A)} \qquad\qquad \overline{Out_{Sys}(B)}$$

  - Is equivalent to one rule with a **diff**-term

$$\overline{Out_{Sys}(\mathbf{diff}(A, B))}$$

# Approximating observational equivalence using mirroring

- Both systems contain same rules modulo diff-terms
- Idea: assume that each rule simulates itself
- Compute **mirrors** of each execution into the other system
- If the mirrors are **valid executions**, we have **observational equivalence** (sound approximation)

# Invalid mirrors and attacks

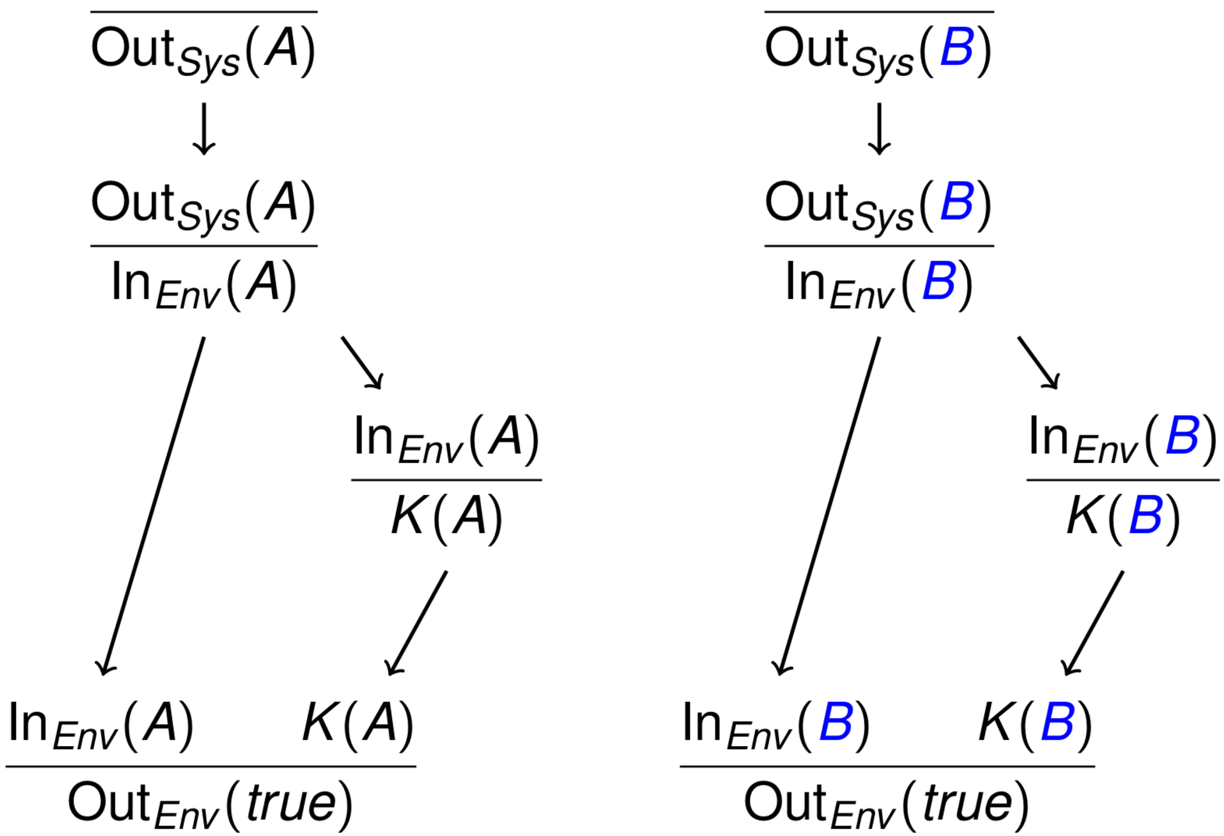
Bidder picks A/B, observer compares to public value A

$$\begin{array}{ccc} \dfrac{}{\mathrm{Out}_{Sys}(A)} & & \\ \downarrow & & \\ \dfrac{\mathrm{Out}_{Sys}(A)}{\mathrm{In}_{Env}(A)} & & \dfrac{}{K(A)} \\ \searrow & & \swarrow \\ & \dfrac{\mathrm{In}_{Env}(A) \qquad K(A)}{\mathrm{Out}_{Env}(\mathit{true})} & \end{array} \qquad \begin{array}{ccc} \dfrac{}{\mathrm{Out}_{Sys}({\color{blue}B})} & & \\ \downarrow & & \\ \dfrac{\mathrm{Out}_{Sys}({\color{blue}B})}{\mathrm{In}_{Env}({\color{blue}B})} & & \dfrac{}{K(A)} \\ \searrow & & \swarrow \\ & \xcancel{\dfrac{\mathrm{In}_{Env}({\color{blue}B}) \qquad K(A)}{\mathrm{Out}_{Env}(\mathit{true})}} & \end{array}$$

**Counter example** to observational equivalence

# Valid mirror

Observer compares system output to itself

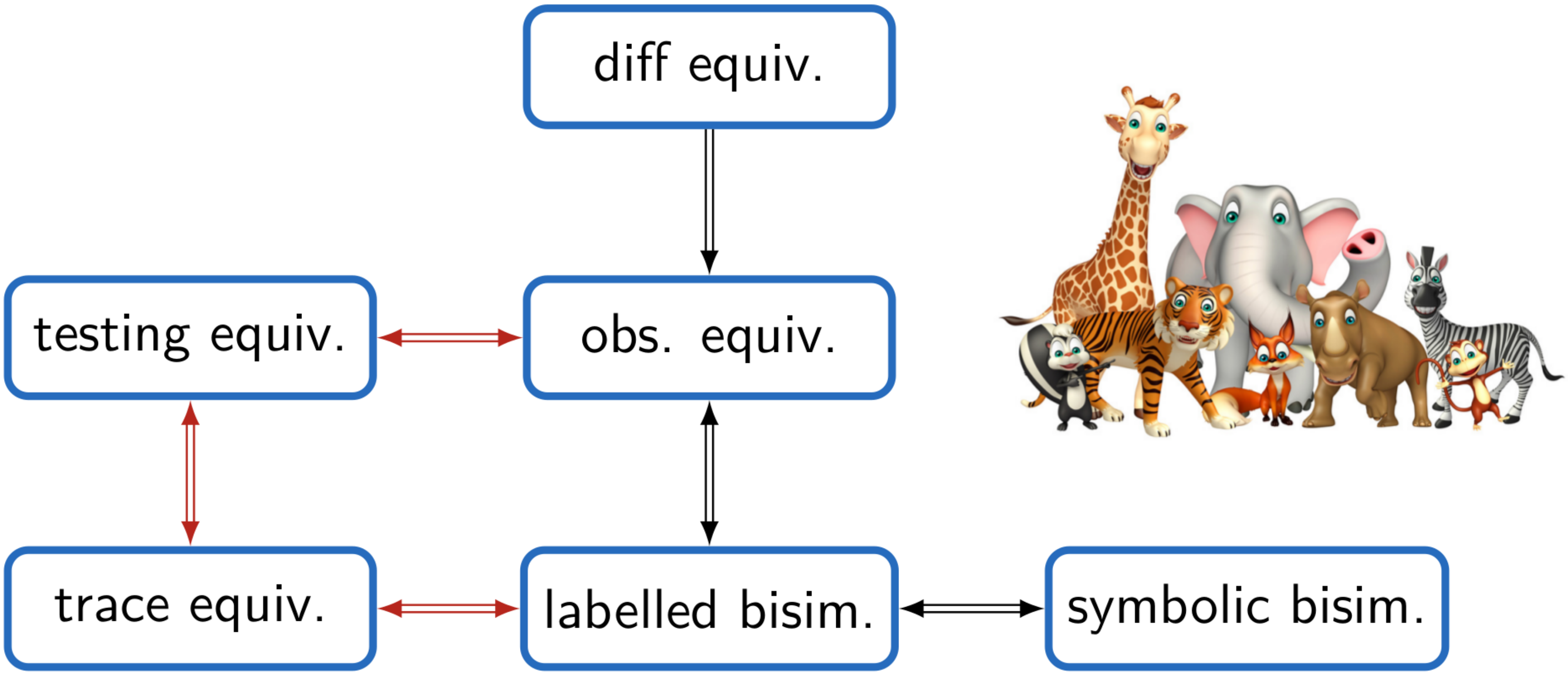



- **All** mirrors need to be valid for observational equivalence

# Dependency graph equivalence

- A **diff**-system is dependency graph equivalent if mirrors of all dependency graphs rooted in any rule on both sides are valid.
  - Sound but incomplete approximation
  - Efficient and sufficient in practice
- Input:
  - Protocol specification
  - Property: equivalence given two choices for some term(s)
    - Example: random value vs expected value
- Output:
  - Yes, observational equivalent
  - No, dependency graph with invalid mirror
  - Non-termination possible

# The equivalence zoo

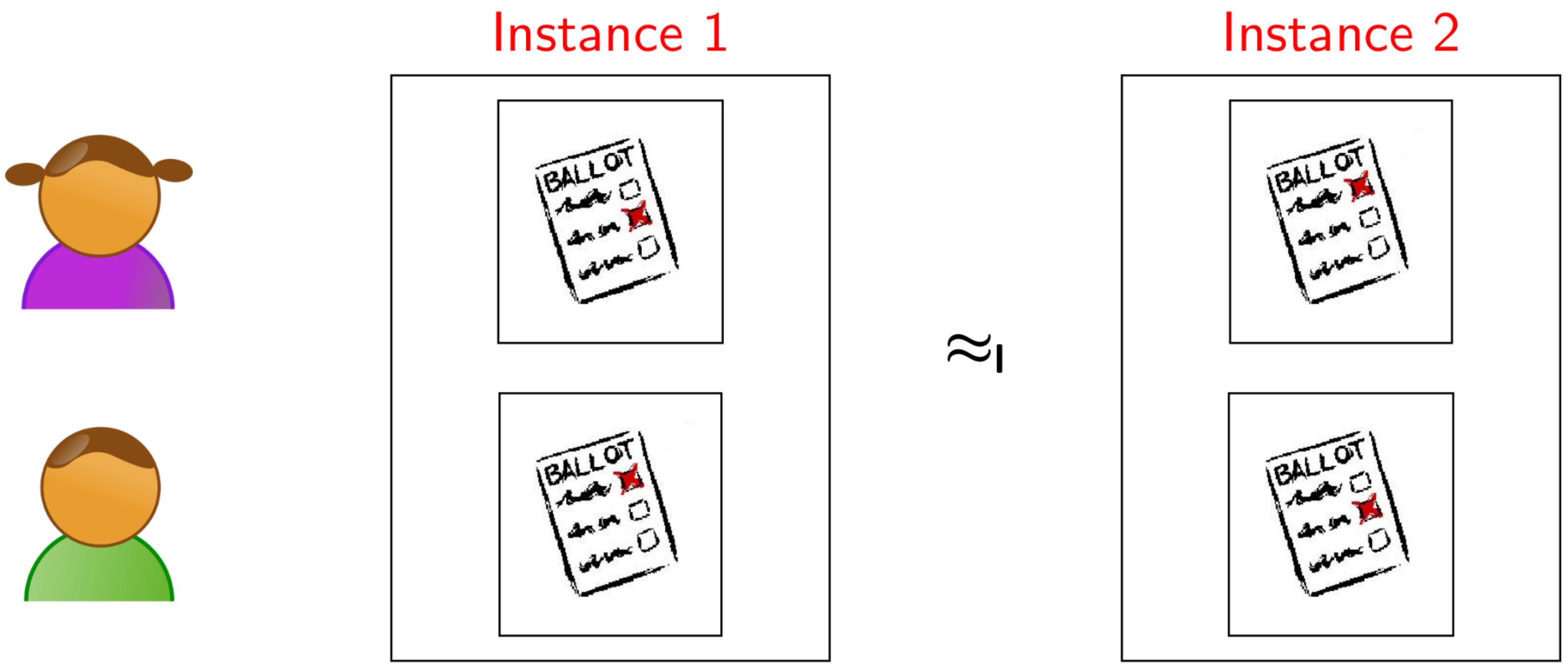



Red arrows require assumptions: determinate processes + bounded sessions (no replication)

# Case studies

- **Feldhofer's RFID** protocol
  - Adversary cannot determine which RFID tag is communicating with reader
  - Automatically verified in 1.6 seconds
- **Signed Diffie-Hellman** key exchange
  - Real-or-random secrecy of session key
  - Needs manual guidance in one subcase
  - Automatically completed proof in 2.5 minutes
- TPM_Envelope
  - Real-or-random secrecy
  - Finds attack for deterministic encryption
    - Despite previous proof wrt trace-based secrecy
  - Recommendation: use probabilistic encryption

# Vote privacy for FOO92




Eligibility verified previously

Vote privacy verified automatically as well

# Case studies (II)

| Protocol | Property | Result | Time |
|---|---|---|---|
| Chaum | Unforgeability | ✓ | 0.2s |
| Chaum | Anonymity | ✓ | 7.6s |
| Chaum | Unlinkability | ✓ | 1m13.7s |
| FOO | Eligibility | ✓ | 10.3s |
| FOO | Vote Privacy | ✓ | 4m11.1s |
| Okamoto | Eligibility | ✓ | 8.4s |
| Okamoto | Vote Privacy | ✓ | 1m20.3s |
| Okamoto | Receipt-Freeness | ✓ | 13m35.8s |
| Denning-Sacco | Session Matching | × | 0.3s |
| Needham-Schroeder | Key Secrecy | × | 24.0s |

# Large scale

- TLS 1.3 analyzed with Tamarin at:
  - v10
  - v10+ (fixes to v10)
  - current version
- See tomorrow at TLS:DIV (room 107, 16:00)
- Attack found: 18 messages, 3 modes
  - Finding it manually unlikely

# The future

- Increasing **scope**
  - **Properties**
    - Much ongoing work on trace equivalence properties
- Increasing **precision**
  - Expanding supported equational theories
  - Tamarin is already more precise than other tools, e.g., for Diffie-Hellman representation

# Tamarin: Conclusions

- **Tamarin** offers **many unique features**
  - Unbounded analysis, (guarded) FOL properties, equivalence properties, equational theories, global state, …
  - Enables automated analysis in areas previously out of scope
- It has **additional features** we did not touch on today
  - Reusable lemmas, heuristics tuning, …
- Tool and sources are **free**; development on Github
- Want to continue with Tamarin?
  - https://tamarin-prover.github.io for news and publications
  - https://github.com/tamarin-prover/tamarin-prover

**ralf.sasse@inf.ethz.ch**

# Backup slides

# Guarded Formulas

- All formulas (lemma, restriction) must be **guarded**
  - All variables quantified over **must** appear in terms

$$\forall \overline{x}.F(\overline{z})@i \Rightarrow \psi \qquad \exists \overline{x}.F(\overline{z})@i \land \psi$$

  - Where F is a fact and $\overline{x}$ and $\overline{z}$ are vectors of variables such that $\overline{x} \subseteq \overline{z} \cup i$
  - i.e., all bound variables appear in the fact formula $\mathrm{F}(\overline{z})@i$

# Executability Lemmas

- Executability lemmas are existential properties
- These show the existence of some protocol trace satisfying the formula…
- … instead of the usual case where all traces must satisfy the formula.
- Heuristics tuned for verification
  - Manual intervention needed more often for executability

- lemma exec: exists-trace “...(formula)...”

# Syntax Issues: Type Annotations

- Mark timepoint (index) variables with a hashmark (#) in quantification.
- We omit this in the slides, but it is required in the tool.
- Example:

$$\forall x \ \#i. F(x) @ i$$

# Syntax Issues: Type Annotations (ctd.)

- Mark fresh values with ~
- Mark public values with $
- Be consistent! If a rule contains ~x, $x, and x that is interpreted as three different variables!
  - You do get a warning about it, and should fix it.

# Warnings on Loading a theory

- Warnings give good information what is wrong:
  - Mismatch of type: use of $x and x in same rule
  - Using a fact name with different arities
  - Guardedness problems in formula
- Tamarin strict mode stops you from working with warnings, but is optional:
  - Add command-line parameter: --quit-on-warning

# Storing Proofs

- Complete (or partial) proofs can be stored
  - Click the “Download” button in top right
- These can be reloaded like normal theories
  - Proof is rechecked!

# Eligibility check - FOO92

```
rule V_1:
  let x = commit( $vote , ~r )
      e = blind( x, ~b )
      s = sign ( e , ~ltkV )
  in
    [ Fr( ~r ), Fr( ~b ), !Ltk( V, ~ltkV ) ]
  --[ Created_vote_V_1(x), Created_commit_V_1(e) ]->
    [ Out( <e,s> ), St_V_1( V, $vote, ~r, ~b ) ]

rule A_1:
  let d = sign( e, ~ltkA )
  in
    [ In( <e,sign(e,~ltkV)> ), !AdminLtk( A, ~ltkA ), !Ltk( V, ~ltkV ) ]
  --[ Registered(e), In_A_1(e) ]->
    [ Out(  <e,d>  ) ]

rule V_2:    // Check Admin_Signature & Check the commit
  let e = blind(commit(vote,~r),~b)
      d = sign(blind(commit(vote,~r),~b),~ltkA)
      y = sign(commit(vote,~r),~ltkA)
      x = commit(vote,~r)
  in
    [ In( <e,sign(e,~ltkA)> ), St_V_1( V, vote, ~r, ~b),  !AdminLtk(A, ~ltkA) ]
  --[ ]->
    [ Out( <x,y> ), St_V_2( V, A, vote, ~r ) ]
```

# Advanced modeling

- Channels
- Heuristics options

# Some simple examples

- Indistinguishability of **probabilistic encryption**
  - Adversary cannot distinguish random value from encryption
  - Automatically verified in 0.2 seconds

- Decisional Diffie-Hellman
  - Given algebraic properties of DH exponentiation as equational theory
  - Adversary cannot distinguish g^{ab} from random g^{c}
    - Given g^a and g^b
  - Automatically verified in 15.2 seconds