Verifying Security Protocols in Tamarin

Ralf Sasse

Institute of Information Security
ETH Zurich

Tamarin Day 4, v.1

Jan 28, 2016

Roadmap

1. Observational Equivalence: High-Level Overview
2. Observational Equivalence Details
3. Case Studies

Outline

Motivation

- Security protocol design is **critical** and **error-prone**
- **Symbolic analysis** methods make a difference

Two types of properties:

- **Trace properties**:
  - ★ (Weak) secrecy as reachability
  - ★ Authentication as correspondence

- **Observational equivalence** $\approx$

Why observational equivalence?

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

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

- Intruder can only decrypt if he knows the secret key

Why observational equivalence?

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

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

- Intruder can only decrypt if he knows the secret key

Now 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

Why observational equivalence?

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

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

- Intruder can only decrypt if he knows the secret key

Now 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**, intruder does not know server's secret key

Why observational equivalence?

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

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

- Intruder can only decrypt if he knows the secret key

Now 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**, intruder 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: intruder cannot distinguish
  - ★ a system where the voter votes “Yes” from
  - ★ a system where the voter votes “No”

Observational Equivalence vs Reachability

- **Reachability**-based (weak) secrecy is insufficient
- **Stronger** notion: intruder cannot distinguish
  - ★ a system where the voter votes “Yes” from
  - ★ a system where the voter votes “No”
- **Observational equivalence** between two systems

Observational Equivalence vs Reachability

- **Reachability**-based (weak) secrecy is insufficient
- **Stronger** notion: intruder 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
  - ★ Game-based notions, e.g., ciphertext indistinguishability

Running Example

- **Auction** system
- Property: **strong secrecy** of bids

Running Example

- **Auction** system
- Property: **strong secrecy** of bids
- Property violated: **Shout-out auction**
  - ★ 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

Running Example

- **Auction** system
- Property: **strong secrecy** of bids
- Property violated: **Shout-out auction**
  - ★ 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 is observing

Symbolic Model for Observational Equivalence

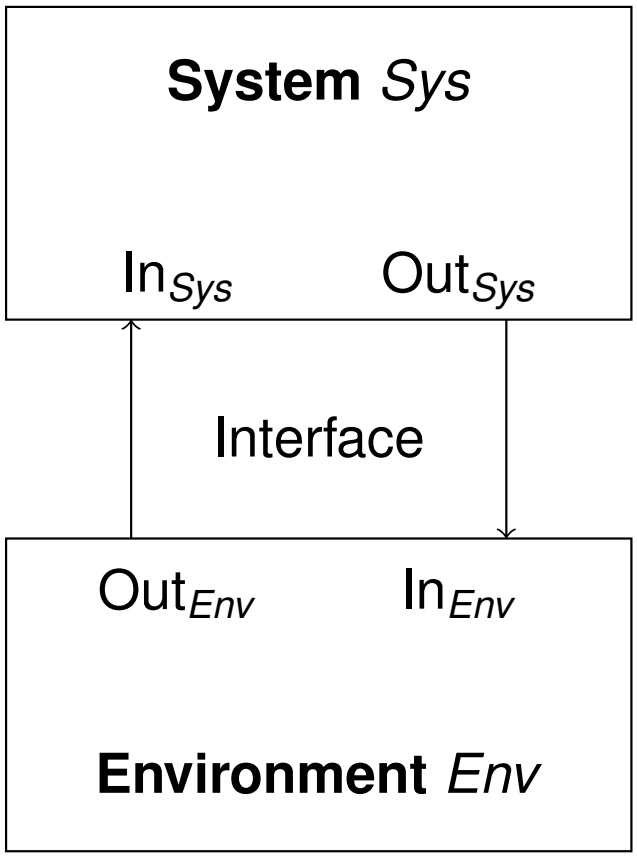
- **Symbolic abstractions** of cryptographic operators
- Enables **high** degree of **automation** for observational equivalence, existing tools:
  - ★ APTE, AKISS, ProVerif, SPEC, Maude-NPA
- Limitations: **No tool** can prove observational equivalence for protocols with **combination** of
  - ★ Mutable state
  - ★ Unbounded sessions
  - ★ Diffie-Hellman
- Examples of applications beyond **state-of-the-art**:
  - ★ Protocols using hardware security modules
  - ★ Key exchange protocols based on Diffie-Hellman

Contribution: Tamarin

- Novel definition of observational equivalence
  - ★ **General** definition within context of multiset rewriting setting
- **Sound algorithm**, first to support all of:
  - ★ Observational equivalence
  - ★ Mutable state
  - ★ Unbounded number of sessions
  - ★ Diffie-Hellman exponentiation
- Implementation, **practicality** demonstrated by
  - ★ non-trivial protocols
- **Largely automated** proofs
- Approach is **not complete**, but **succeeds broadly**

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):

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

  - ★ $K(X)$ represents that environment knows term X
  - ★ last rule models comparisons by the adversary

- Each specification yields a labeled transition system
- Observational equivalence is a kind of **bisimulation** accounting for the adversaries' **viewpoint** and **capabilities**
  - ★ Our definition can be instantiated for various adversaries

Diff terms

- General definition **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

Diff terms

- General definition **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

$$\frac{}{\mathrm{Out}_{Sys}(A)} \qquad \frac{}{\mathrm{Out}_{Sys}(B)}$$

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

$$\frac{}{\mathrm{Out}_{Sys}(\mathbf{diff}(A, B))}$$

Approximating observational equivalence using mirroring

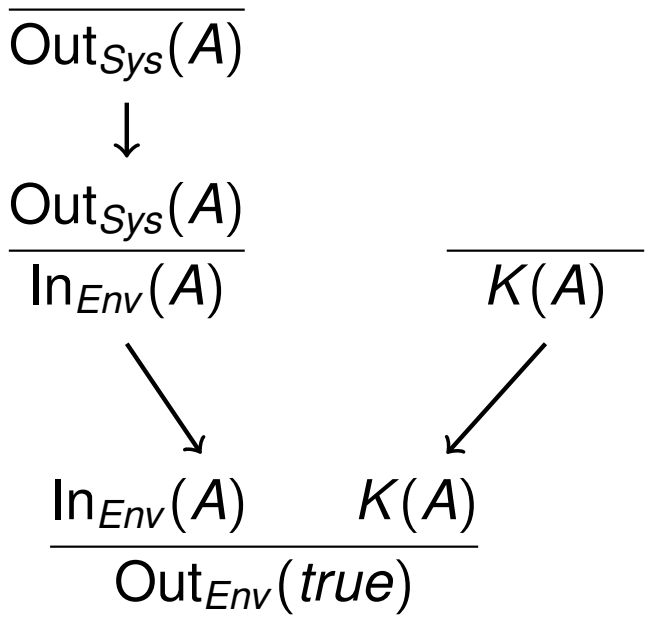
- Both systems contain the same rules modulo diff-terms
- Idea: assume that each rule simulates itself
- **Mirrors** each execution into the other system
- If the mirrors are **valid executions**, we have **observational equivalence** (sound approximation)
- We represent executions using dependency graphs
  - ★ Computed via backwards constraint solving

Dependency graphs and mirrors

Bidder picks A, observer compares to public value A

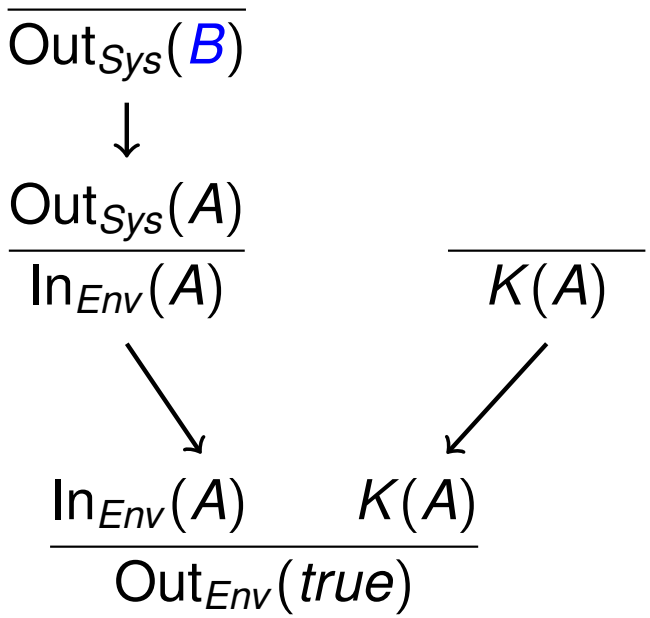
Dependency graphs and mirrors

Bidder picks A, observer compares to public value A



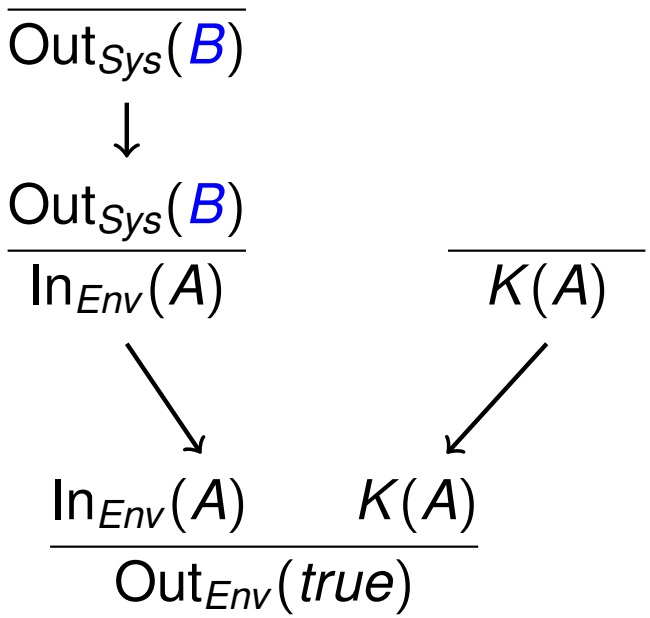
Dependency graphs and mirrors

Bidder picks B, observer compares to public value A



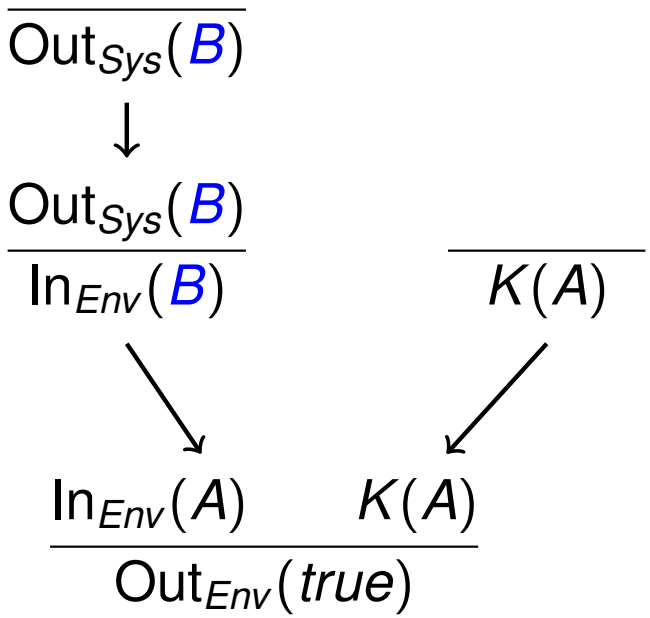
Dependency graphs and mirrors

Bidder picks B, observer compares to public value A



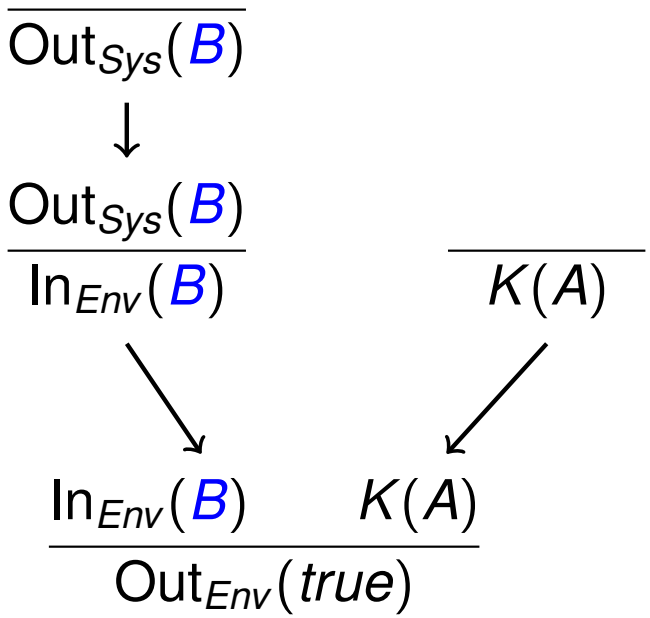
Dependency graphs and mirrors

Bidder picks B, observer compares to public value A



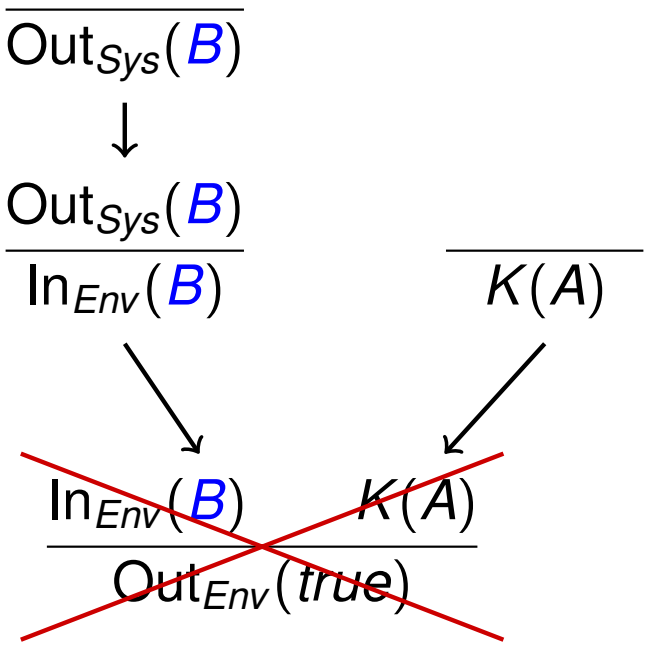
Dependency graphs and mirrors

Bidder picks B, observer compares to public value A



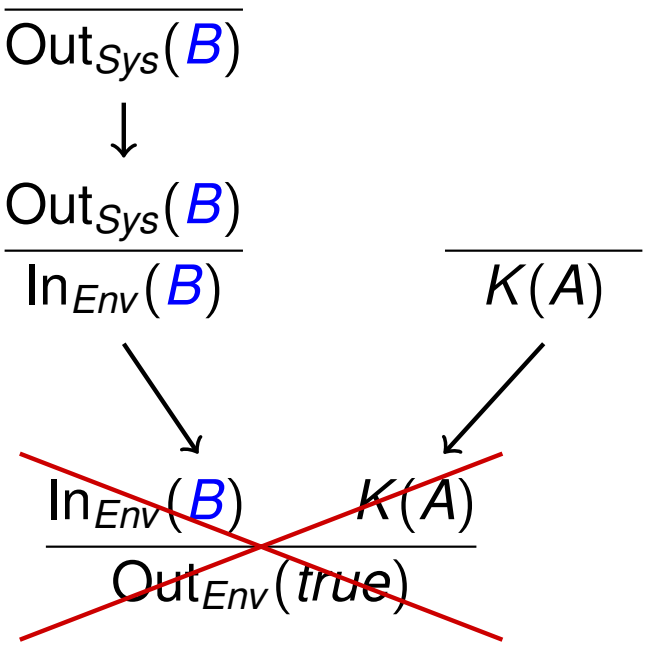
Dependency graphs and mirrors

Bidder picks B, observer compares to public value A



Dependency graphs and mirrors

Bidder picks B, observer compares to public value A



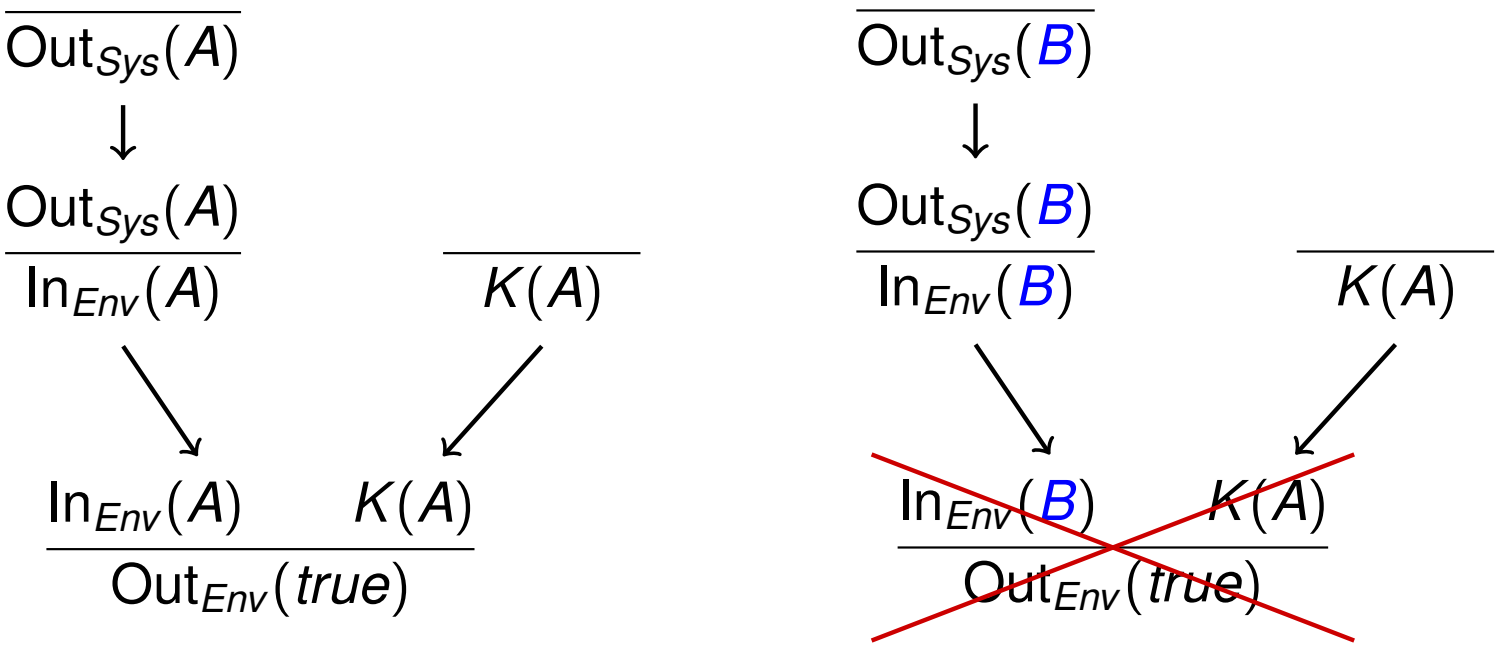
- Dependency graph mirror for bidder choice B is **invalid**
  - ★ Adversary choices stay fixed, comparison is with A

Invalid mirrors and attacks

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

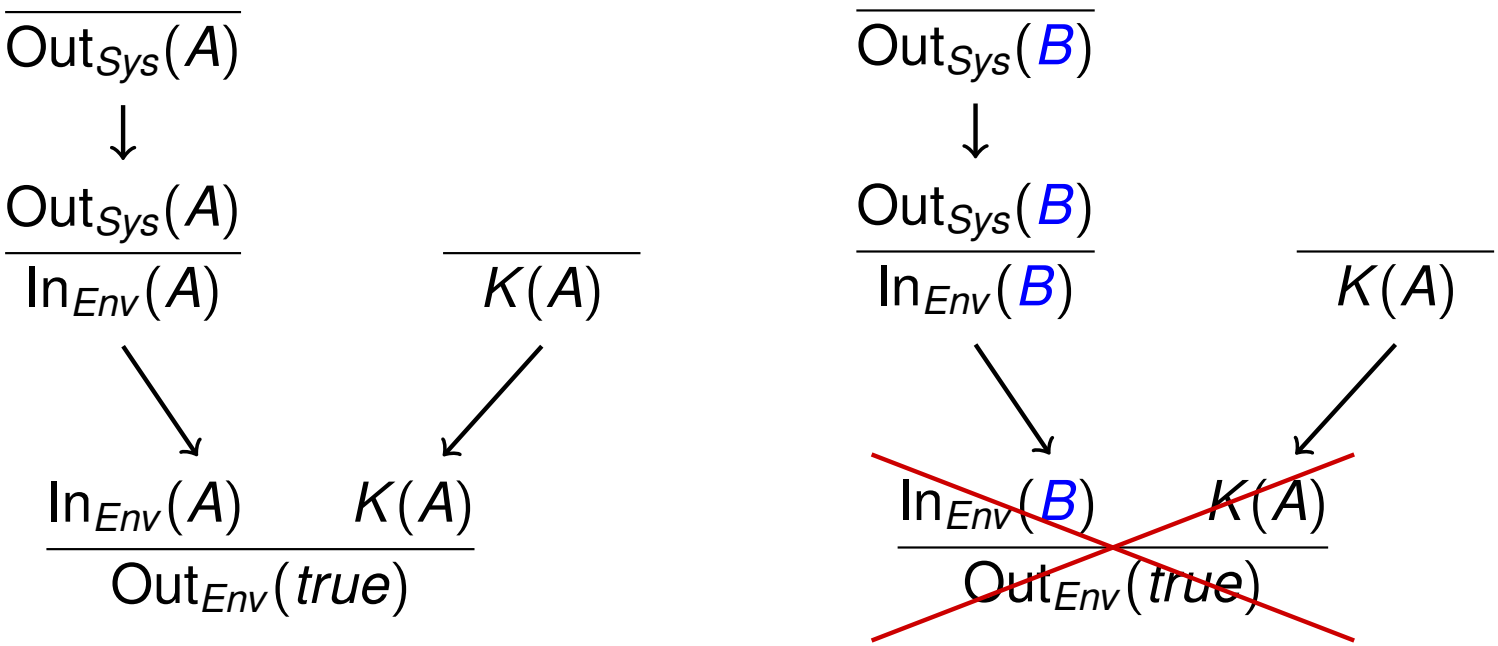
Invalid mirrors and attacks

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



Invalid mirrors and attacks

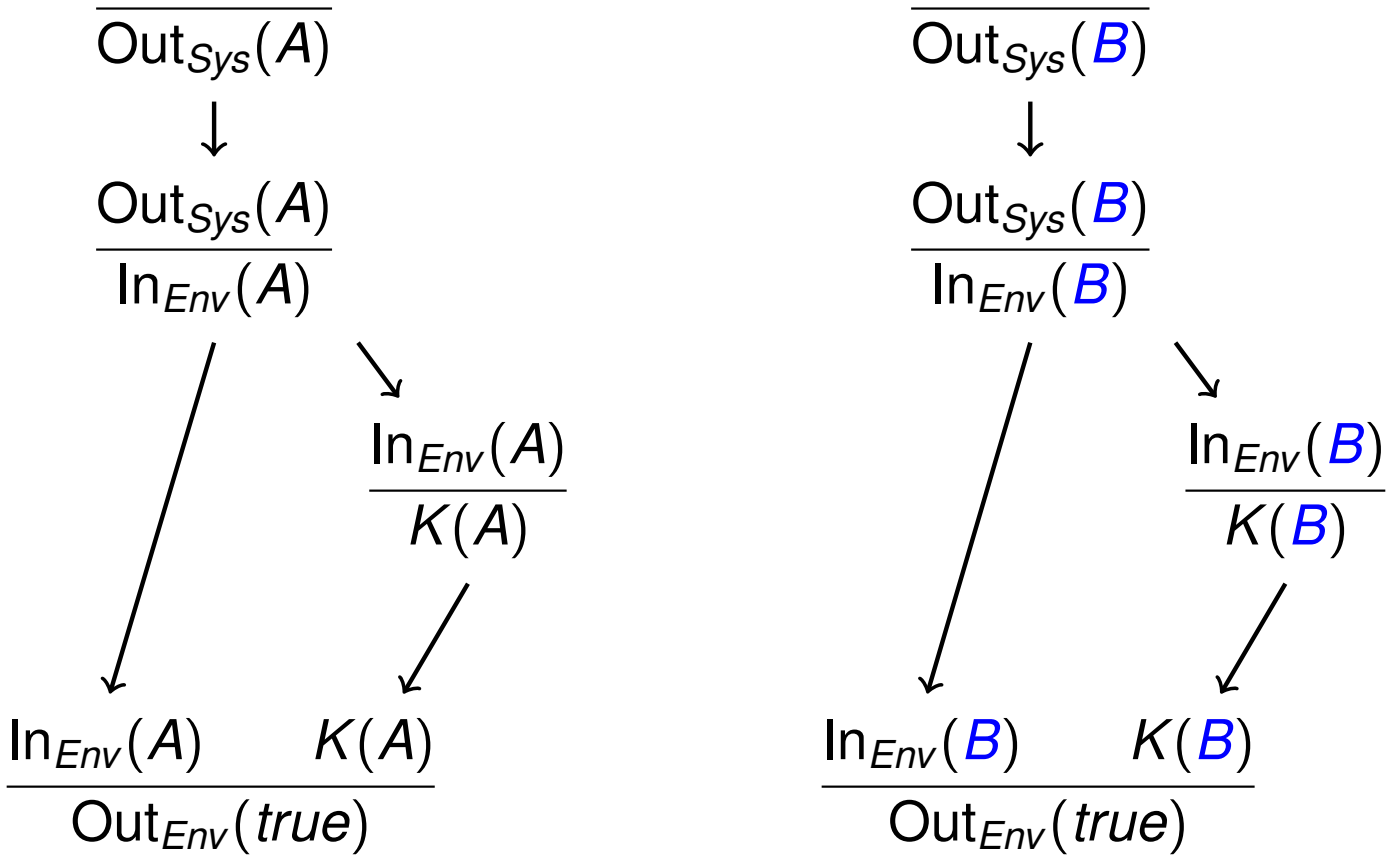
Bidder picks A/B, observer compares to public value A



- **Counter example** to observational equivalence of the given systems

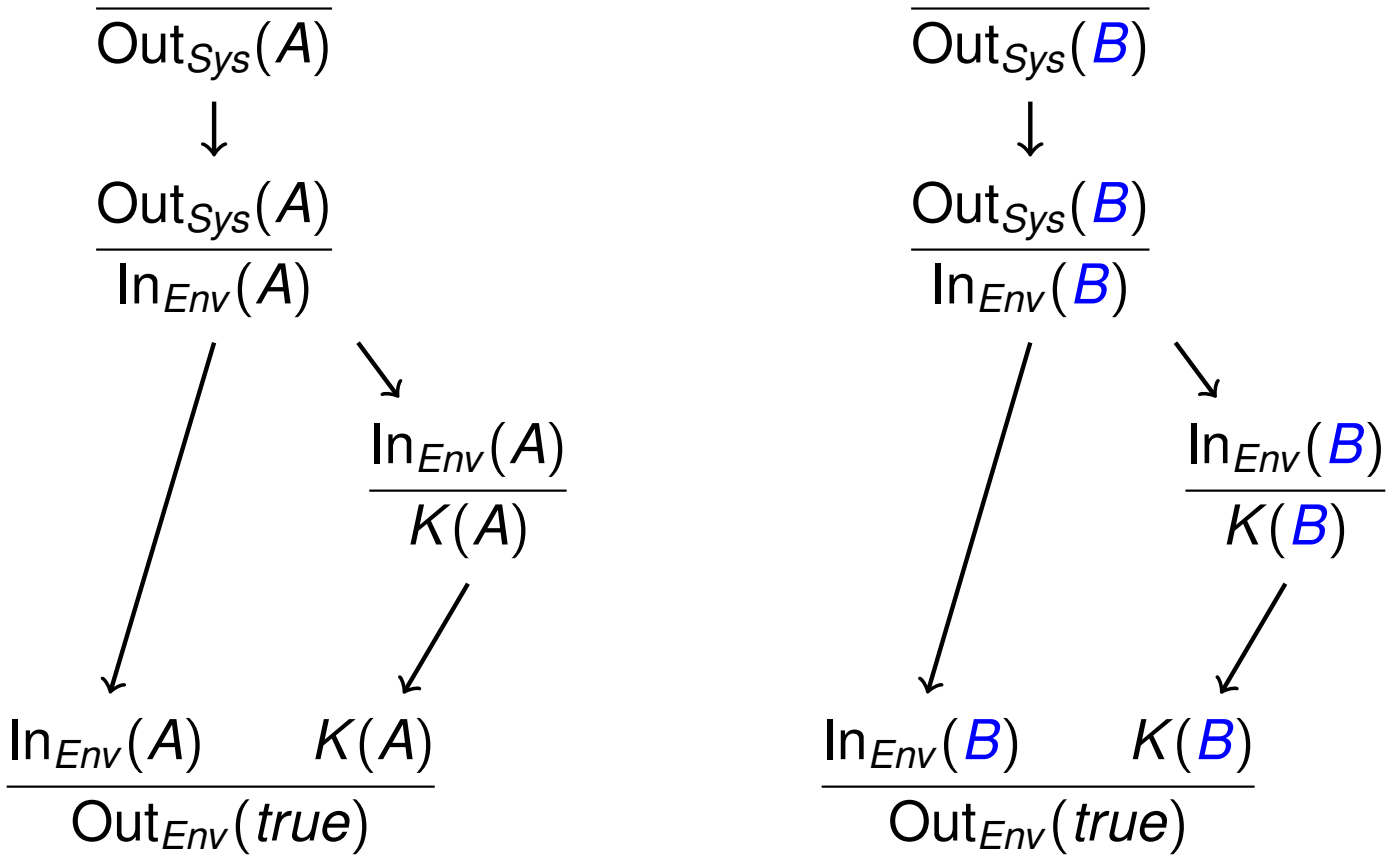
Valid mirror

Observer compares system output to itself



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

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

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

Tamarin advantages

Approach implemented in the Tamarin tool:

- Tamarin supports **verification** with:
  - ★ equational theories (DH), induction, loops, mutable state
- Security protocol model is based on rewriting
- Restricted FOL for security properties
- Equational theories modeling algebraic properties
- Constraint-solving algorithm for analysis of unbounded number of sessions
- Performance good despite undecidability

Tamarin extension for observational equivalence

Implemented algorithm:

- Extended constraint solving
- (**Normal**) dependency graphs
  - ★ Important for state space reduction and termination
- Equivalence of dependency graphs by mirroring

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

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
  - ★ We recommend to use probabilistic encryption

Future Work

- Implement **more precise approximation**
  - ★ Currently rules must match 1–1 due to mirroring
  - ★ General definition does allow matching using different, even multiple, rules
- Protocols with loops: need induction
- Further case studies
  - ★ Signed Diffie-Hellman with Perfect Forward Secrecy
  - ★ NAXOS, authenticated key exchange with PFS

Related Work

	APTE	AKISS	ProVerif	ProVerifDH	SPEC	Maude-NPA	Tamarin	Extension
Unbounded sessions			x	x		x	x	x
Mutable state	x	x			x	?	x	x
Diffie-Hellman	x	x		x	x	x	x	x
Definable crypto	x	x	x	x		x	x	x
Verification	x	x	x	x	x	x	x	x
Obs. equiv.	x	x	x		x	/		x

- APTE, AKISS
  - ★ Limited to bounded number of sessions
- ProVerif
  - ★ No mutable state support
  - ★ DH support only without observational equivalence
- SPEC
  - ★ Fixed crypto primitives, bounded number of sessions
- StatVerif, SAPIC
  - ★ Support mutable state, but no observational equivalence
- Maude-NPA
  - ★ Creates synchronous product of two similar protocols
  - ★ Suffers from termination issues - only finds attacks

Summary

- Extended multiset rewriting approach to observational equivalence
- Result is well-suited for cryptographic protocol analysis
- Shown algorithm's **effectiveness and scope** on case studies
- **High** degree of **automation**

Outline

Modeling

Define functions $one(\cdot)$ and $two(\cdot)$, and a fact symbol M.

$$\begin{aligned} Env = \{ \quad & E_{null} : \multimap[]\rightarrow \mathrm{M}(null), \\ & E_{one} : \mathrm{M}(x) \multimap[]\rightarrow \mathrm{M}(one(x)), \\ & E_{two} : \mathrm{M}(x) \multimap[]\rightarrow \mathrm{M}(two(x)), \\ & E_{check} : \mathrm{M}(x), \mathrm{In}_{Env}(x) \multimap[]\rightarrow \mathrm{Out}_{Env}(true) \quad \} \end{aligned}$$

Using the final rule, E_{check}, the system can compare a constructed term with the value stored in the $\mathrm{In}_{Env}(\cdot)$ fact.

Modeling Example

Each fact is associated with a recipe of its derivation

- Order the premises and conclusions of a rule: $id : l \multimap[\ a\ ]\mapsto r$ as $seq^{\leq}(l)$ and $seq^{\leq}(r)$
- For fact $F \in r$, where k is the index of F in $seq^{\leq}(r)$ we get

$$recipe(F) = id_k(newvars(F), [recipe(l_1), \ldots, recipe(l_n)])$$

- Abusing notation, for a rule with name id we get:

$$\begin{aligned} recipe(id) = \quad & id([newvars(r_1), \ldots, newvars(r_m)], \\ & [recipe(l_1), \ldots, recipe(l_n)]), \end{aligned}$$

Semantics

The semantics of a set of multiset rewrite rules P are given by a labeled transition relation $\rightarrow_P \subseteq \mathcal{G}^\sharp \times (\mathcal{G}^\sharp \times \rho) \times \mathcal{G}^\sharp$, defined by the transition rule:

$$\frac{ri = id : l -[\, a \,]\!\!\rightarrow r \in_E ginsts(P) \qquad lfacts(l) \subseteq^\sharp S \qquad pfacts(l) \subseteq S}{S \xrightarrow[recipe(id)]{set(a)}_P ((S \setminus^\sharp lfacts(l)) \cup^\sharp mset(r))}$$

Example: Pairs

Two systems, where the first system outputs a pair of identical values

$$S_A = \{ \quad A : \multimap[]\to \mathrm{Out}_{Sys}((x, x)) \quad \}$$

and the second system may output two different values

$$S_B = \{ \quad B : \multimap[]\to \mathrm{Out}_{Sys}((x, y)) \quad \}.$$

Example: Pairs

Two systems, where the first system outputs a pair of identical values

$$S_A = \{ \quad A : \multimap[] \to \mathsf{Out}_{Sys}((x, x)) \quad \}$$

and the second system may output two different values

$$S_B = \{ \quad B : \multimap[] \to \mathsf{Out}_{Sys}((x, y)) \quad \}.$$

In S_A, we have that

$$\emptyset \quad \xrightarrow[A([\{m/x\}],[])]{} \{\mathsf{Out}_{Sys}((m, m))\}.$$

Example: Pairs

Two systems, where the first system outputs a pair of identical values

$$S_A = \{ \quad A : \multimap[] \to \mathsf{Out}_{Sys}((x, x)) \quad \}$$

and the second system may output two different values

$$S_B = \{ \quad B : \multimap[] \to \mathsf{Out}_{Sys}((x, y)) \quad \}.$$

In S_A, we have that

$$\emptyset \quad \xrightarrow[A([\{m/x\}],[])]{} \{\mathsf{Out}_{Sys}((m, m))\}.$$

In S_B, we can either take a similar transition

$$\emptyset \quad \xrightarrow[B([\{m/x, m/y\}],[])]{} \{\mathsf{Out}_{Sys}((m, m))\}$$

or alternatively

Example: Pairs

Two systems, where the first system outputs a pair of identical values

$$S_A = \{ \quad A : \multimap[] \to \mathrm{Out}_{Sys}((x, x)) \quad \}$$

and the second system may output two different values

$$S_B = \{ \quad B : \multimap[] \to \mathrm{Out}_{Sys}((x, y)) \quad \}.$$

In S_A, we have that

$$\emptyset \xrightarrow[A([\{m/x\}],[])]{} \{\mathrm{Out}_{Sys}((m, m))\}.$$

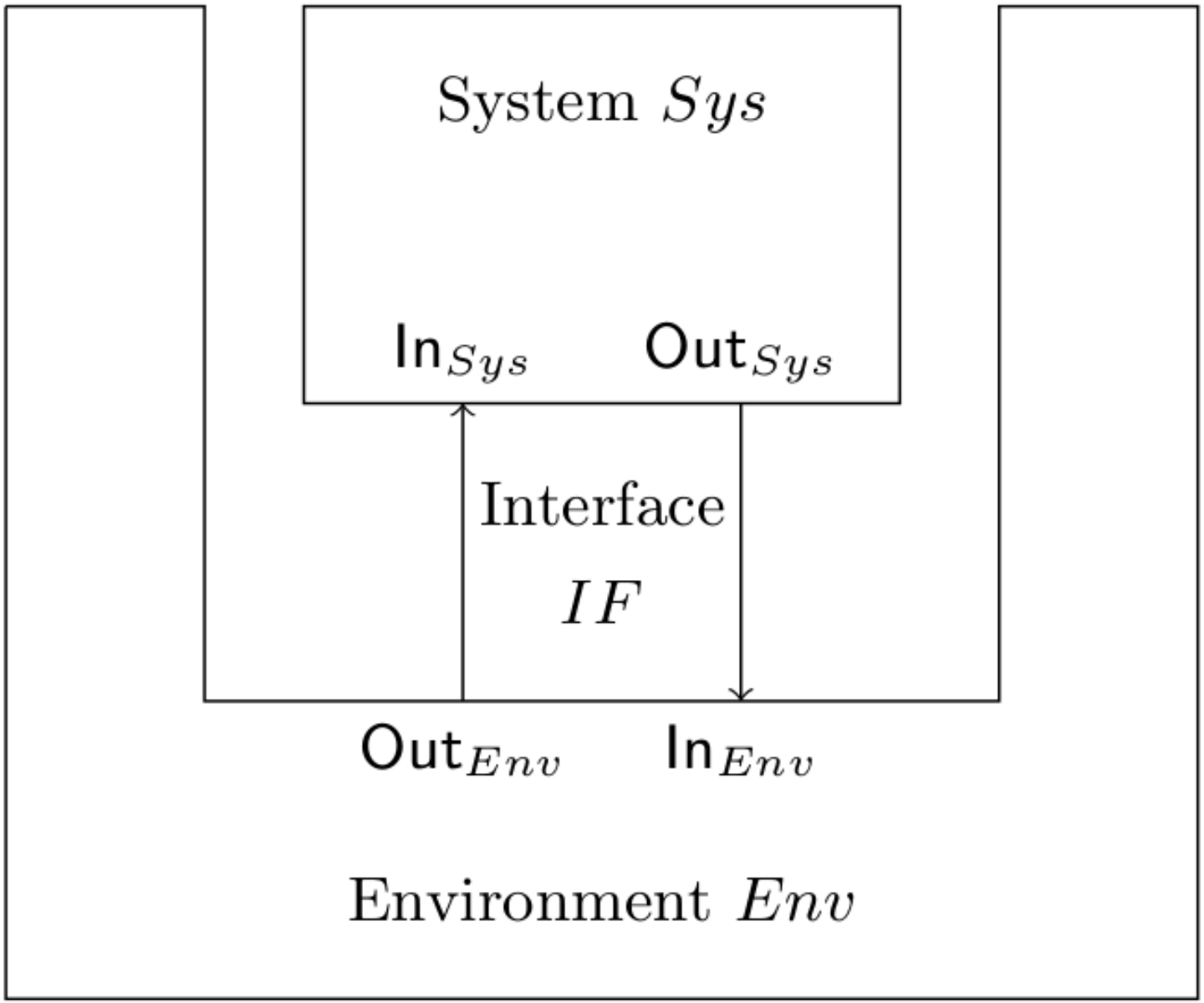
In S_B, we can either take a similar transition

$$\emptyset \xrightarrow[B([\{m/x, m/y\}],[])]{} \{\mathrm{Out}_{Sys}((m, m))\}$$

or alternatively

$$\emptyset \xrightarrow[B([\{m/x, n/y\}],[])]{} \{\mathrm{Out}_{Sys}((m, n))\}.$$

System Model



Interface rules

$$
\begin{aligned}
OUT &= \{OUT : \mathrm{Out}_{Sys}(M) \multimap [\ \mathrm{O}\ ] \mapsto \mathrm{In}_{Env}(M)\} \\
IN &= \{IN : \mathrm{Out}_{Env}(M) \multimap [\ \mathrm{I}\ ] \mapsto \mathrm{In}_{Sys}(M)\} \\
IF &= OUT \cup IN
\end{aligned}
$$

Special recipe for In_{Env}

- Environment is not supposed to see the internal system choices.
- Reflected by defining recipe of $\mathsf{In}_{Env}(M)$ fact differently:

$$recipe(\mathsf{In}_{Env}(M)) = OUT_1([], x)$$

  where x is a new variable.
- The recipe associated to the rule is: $recipe(OUT) = OUT([], x)$.
- Replaces the $\mathsf{Out}_{Sys}(M)$ fact's recipe with a variable, as the recipe is considered internal to the system.

Pairs and recipes

$$Env = \{ \quad C : \mathsf{In}_{Env}(x, x) \multimap[] \to \mathsf{Out}_{Env}(true) \quad \}.$$

Then in $S_A \cup IF \cup Env$ we have

$$\begin{aligned} \emptyset \quad & \xrightarrow[A([\{m/x\}],[])]{} \{\mathsf{Out}_{Sys}((m, m))\} \\ & \xrightarrow[OUT([],z)]{O} \{\mathsf{In}_{Env}((m, m))\} \\ & \xrightarrow[C([],[OUT_1([],z)])]{} \{\mathsf{Out}_{Env}(true)\}. \end{aligned}$$

In $S_B \cup IF \cup Env$ similarly:

$$\begin{aligned} \emptyset \quad & \xrightarrow[B([\{m/x,m/y\}],[])]{} \{\mathsf{Out}_{Sys}((m, m))\} \\ & \xrightarrow[OUT([],z)]{O} \{\mathsf{In}_{Env}((m, m))\} \\ & \xrightarrow[C([],[OUT_1([],z)])]{} \{\mathsf{Out}_{Env}(true)\}. \end{aligned}$$

To adversary these look the same.

Pairs and recipes

However, in $S_B \cup IF \cup Env$ we also have the following transitions:

$$\emptyset \xrightarrow[B([\{m/x, n/y\}],[])]{} \{\mathrm{Out}_{Sys}((m,n))\}$$
$$\xrightarrow[OUT([],z)]{O} \{\mathrm{In}_{Env}((m,n))\}\,.$$

Note that the first transition has a different recipe as we instantiate y differently, but again the output replaces this recipe with a new variable z, which is the same on both sides.
The two system will turn out to be not observational equivalent shortly.

High-level definition

S_A and S_B are observational equivalent wrt interface and environment, given empty initial states of the relation:

- Interface and environment rules must make the same choices on both sides, with same recipe.
- Application of system rules S_A (resp. S_B) must be matchable by some number of system rule applications of S_B (resp. S_A), with some recipes, and end in states in the relation.

Observational equivalence - definition

Two sets of multiset rewrite rules S_A and S_B are observational equivalent with respect to an environment *Env* (and interface *IF*) if there is a relation between the initial states in $S_A \cup IF \cup Env$ (left system) and $S_B \cup IF \cup Env$ (right system), and for all pairs of states in that relation:

- If the left system can make a move with an environment or interface rule, the right system can match it precisely
  - ★ Resulting states are in the relation
- If the left system can make a move with an S_A rule, the right system can match it, possibly using multiple steps
  - ★ resulting states are in the relation

The same holds in the other direction.

Algorithm

1: **function** VERIFY(S)
2: $RU \leftarrow L(S) \cup R(S) \cup IF \cup Env$

Algorithm

1: **function** VERIFY(S)
2: $RU \leftarrow L(S) \cup R(S) \cup IF \cup Env$
3: **while** $RU \neq \emptyset$ **do**
4: choose $r \in RU$, $RU \leftarrow (RU \setminus \{r\})$

Algorithm

1: **function** VERIFY(S)
2: $RU \leftarrow L(S) \cup R(S) \cup IF \cup Env$
3: **while** $RU \neq \emptyset$ **do**
4: choose $r \in RU$, $RU \leftarrow (RU \setminus \{r\})$
5: compute $DG \leftarrow dgraphs(r)$ by constraint solving

Algorithm

1: **function** VERIFY(S)
2: $RU \leftarrow L(S) \cup R(S) \cup IF \cup Env$
3: **while** $RU \neq \emptyset$ **do**
4: choose $r \in RU$, $RU \leftarrow (RU \setminus \{r\})$
5: compute $DG \leftarrow dgraphs(r)$ by constraint solving
6: **if** $\exists dg \in DG$ s.t. $mirrors(dg)$ lacks ground instances
7: **then return** “potential attack found: ”, dg

Algorithm

1: **function** VERIFY(S)
2: $RU \leftarrow L(S) \cup R(S) \cup IF \cup Env$
3: **while** $RU \neq \emptyset$ **do**
4: choose $r \in RU$, $RU \leftarrow (RU \setminus \{r\})$
5: compute $DG \leftarrow dgraphs(r)$ by constraint solving
6: **if** $\exists dg \in DG$ s.t. $mirrors(dg)$ lacks ground instances
7: **then return** “potential attack found: ”, dg
8: **return** “verification successful”

Example: Pairs

In $S_B \cup IF \cup Env$ we have

$$\emptyset \xrightarrow[B([\{m/x,n/y\}],[])]{} \{\mathrm{Out}_{Sys}((m,n))\} \xrightarrow[OUT([],z)]{O} \{\mathrm{In}_{Env}((m,n))\}\,.$$

The only way for $S_A \cup IF \cup Env$ to simulate this would be

$$\emptyset \xrightarrow[A([\{m/x\}],[])]{} \{\mathrm{Out}_{Sys}((m,m))\} \xrightarrow[OUT([],z)]{O} \{\mathrm{In}_{Env}((m,m))\}\,.$$

and further rule A transitions, which add more $\mathrm{Out}_{Sys}((m,m))$ to S. Output rule can only be used once, so just one $\mathrm{In}_{Env}((m,m))$ in S. Then

$$S \xrightarrow[C([],[OUT_1([],z)])]{} \{\mathrm{Out}_{Env}(true)\},$$

but in state $\{\mathrm{In}_{Env}((m,n))\}$ no such transition is possible, so $S_A \not\approx_{Env} S_B$.

Example result

The previous example showed that if a rule is applicable by the environment on one side, and not on the other, the two sides are distinguishable. Thus the two systems are not observationally equivalent.

Next we see an example where the recipe used by the environment is crucial for distinction.

Example: different Environment

$$S_A = \{ \quad A : \multimap[]\rightarrow \mathrm{Out}_{Sys}((x, x)) \quad \}$$

$$S_B = \{ \quad B : \multimap[]\rightarrow \mathrm{Out}_{Sys}((x, y)) \quad \}$$

Environment Env' with persistent $\mathrm{M}(\cdot)$:

$$\begin{aligned} Env' = \{ \quad & E_{fst} : \mathrm{In}_{Env}((x, y)) \multimap[]\rightarrow \mathrm{M}(x), \\ & E_{snd} : \mathrm{In}_{Env}((x, y)) \multimap[]\rightarrow \mathrm{M}(y), \\ & E_{cmp} : \mathrm{M}(x), \mathrm{M}(x) \multimap[]\rightarrow \mathrm{Out}_{Env}(true) \quad \}, \end{aligned}$$

We expect this environment to be able to distinguish S_A and S_B, but need to reason differently than before.

Example continued

Let us start with $S_B \cup IF \cup Env'$:

$$\begin{aligned}
\emptyset \quad &\xrightarrow[B([\{m/x,n/y\}],[])]{} \{\mathrm{Out}_{Sys}((m,n))\} \\
&\xrightarrow[OUT([],z)]{O} \{\mathrm{In}_{Env}((m,n))\} \\
&\xrightarrow[E_{fst}([],[OUT_1([],z)])]{} \{\mathrm{M}(m)\} \\
&\xrightarrow[E_{snd}([],[OUT_1([],z)])]{} \{\mathrm{M}(m),\mathrm{M}(n)\}\,.
\end{aligned}$$

In $S_A \cup IF \cup Env'$ this can be simulated as follows:

$$\begin{aligned}
\emptyset \quad &\xrightarrow[A([\{m/x\}],[])]{} \{\mathrm{Out}_{Sys}((m,m))\} \\
&\xrightarrow[OUT([],z)]{O} \{\mathrm{In}_{Env}((m,m))\} \\
&\xrightarrow[E_{fst}([],[OUT_1([],z)])]{} \{\mathrm{M}(m)\} \\
&\xrightarrow[E_{snd}([],[OUT_1([],z)])]{} \{\mathrm{M}(m),\mathrm{M}(m)\}\,.
\end{aligned}$$

Example continued

Moreover, we can compare the first and second value of the tuple with

$$\{\mathrm{M}(m), \mathrm{M}(m)\} \underset{r_1}{\longrightarrow} \{\mathrm{M}(m), \mathrm{M}(m), \mathrm{Out}_{Env}(true)\},$$

where

$$r_1 = E_{cmp}([], [E_{fst,1}([], [OUT_1([], z)]), E_{snd,1}([], [OUT_1([], z)])])$$

This transition cannot be matched by $S_B \cup IF \cup Env'$. Note however that the following transition is possible for $S_B \cup IF \cup Env'$:

$$\{\mathrm{M}(m), \mathrm{M}(n)\} \underset{r_2}{\longrightarrow} \{\mathrm{M}(m), \mathrm{M}(n), \mathrm{Out}_{Env}(true)\},$$

where

$$r_2 = E_{cmp}([], [E_{fst,1}([], [OUT_1([], z)]), E_{fst,1}([], [OUT_1([], z)])]).$$

The recipes are crucial, as otherwise we could compare the first value with itself and it would seem these are observationally equivalent.

Example: coin returning vending machine

Now we add equational reasoning to the mix. We use 1 € and 2 € coins. Represent these by functions *one* and *two*, with constant *null*. How to return 3 €?

Example: coin returning vending machine

Now we add equational reasoning to the mix. We use 1 € and 2 € coins. Represent these by functions *one* and *two*, with constant *null*. How to return 3 €? Which coins and which order?

Example: coin returning vending machine

Now we add equational reasoning to the mix. We use 1 € and 2 € coins. Represent these by functions *one* and *two*, with constant *null*. How to return 3 €? Which coins and which order?
Specification:

$$S_A = \{ \quad A : \multimap[]\rightarrow \mathrm{Out}_{Sys}(two(one(null))) \quad \}.$$

Example: coin returning vending machine

Now we add equational reasoning to the mix. We use 1 € and 2 € coins. Represent these by functions *one* and *two*, with constant *null*. How to return 3 €? Which coins and which order?
Specification:

$$S_A = \{ \quad A : \multimap[]\rightarrow \mathrm{Out}_{Sys}(two(one(null))) \quad \}.$$

Implementation:

$$\begin{aligned} S_B = \{ \quad & B_1 : \multimap[]\rightarrow \mathrm{Out}_{Sys}(two(one(null))), \\ & B_2 : \multimap[]\rightarrow \mathrm{Out}_{Sys}(one(one(one(null)))), \\ & B_3 : \multimap[]\rightarrow \mathrm{Out}_{Sys}(one(two(null))) \qquad\quad \}. \end{aligned}$$

Example: coin value

Environment to check coin value:

$$
\begin{aligned}
Env = \{ \quad & E_{null} : \multimap[]\rightarrow \mathrm{M}(null), \\
& E_{one} : \mathrm{M}(x) \multimap[]\rightarrow \mathrm{M}(one(x)), \\
& E_{two} : \mathrm{M}(x) \multimap[]\rightarrow \mathrm{M}(two(x)), \\
& E_{check} : \mathrm{M}(x), \mathrm{In}_{Env}(x) \multimap[]\rightarrow \mathrm{Out}_{Env}(true) \quad \} .
\end{aligned}
$$

Example: coin value

Environment to check coin value:

$$
\begin{aligned}
Env = \{ \quad & E_{null} : \multimap[]\rightarrow \mathrm{M}(null), \\
& E_{one} : \mathrm{M}(x) \multimap[]\rightarrow \mathrm{M}(one(x)), \\
& E_{two} : \mathrm{M}(x) \multimap[]\rightarrow \mathrm{M}(two(x)), \\
& E_{check} : \mathrm{M}(x), \mathrm{In}_{Env}(x) \multimap[]\rightarrow \mathrm{Out}_{Env}(true) \quad \} .
\end{aligned}
$$

This environment will distinguish based on the exact order of coins.

Example: coin value

Environment to check coin value:

$$\begin{aligned} Env = \{ \quad & E_{null} : \multimap[]\rightarrow \mathrm{M}(null), \\ & E_{one} : \mathrm{M}(x) \multimap[]\rightarrow \mathrm{M}(one(x)), \\ & E_{two} : \mathrm{M}(x) \multimap[]\rightarrow \mathrm{M}(two(x)), \\ & E_{check} : \mathrm{M}(x), \mathrm{In}_{Env}(x) \multimap[]\rightarrow \mathrm{Out}_{Env}(true) \quad \}. \end{aligned}$$

This environment will distinguish based on the exact order of coins.

We only want to check that the same value is returned.

Example: coin value

Environment to check coin value:

$$\begin{aligned} Env = \{ \quad & E_{null} : \multimap[]\rightarrow \mathrm{M}(null), \\ & E_{one} : \mathrm{M}(x) \multimap[]\rightarrow \mathrm{M}(one(x)), \\ & E_{two} : \mathrm{M}(x) \multimap[]\rightarrow \mathrm{M}(two(x)), \\ & E_{check} : \mathrm{M}(x), \mathrm{In}_{Env}(x) \multimap[]\rightarrow \mathrm{Out}_{Env}(true) \quad \}. \end{aligned}$$

This environment will distinguish based on the exact order of coins.

We only want to check that the same value is returned.

Add the equation $two(x) = one(one(x))$. Then the specification and implementation are observationally equivalent for the given environment.

How to prove ObsEq: Bi-Systems

- Requiring the tool to guess which rule on one side is simulated by what rule on the other is difficult
- Instead use diff-terms and thus same-format rules
- Additionally we require that each rule is simulated by itself
- This is sufficient to prove ObsEq, but not necessary

What is covered?

- Real-or-random test
- Privacy properties of voting and auctions
- Ciphertext indistinguishability
- Authenticated key-exchange
- All previous example in this lecture

Bi-System yields two systems

For bi-system S we get

- Left instance $L(S)$
- Right instance $R(S)$

where for all terms $\mathit{diff}[M, N]$ we get M in $L(S)$ and N in $R(S)$.

Example: pairs again

Single bi-system S combines S_A and S_B as

$$S = \{ \quad AB : \multimap[]\to \mathrm{Out}_{Sys}((x, diff[x, y])) \quad \},$$

where $L(S) = S_A$ and $R(S) = S_B$.

Example: coins again

$$
\begin{aligned}
S = \{ \quad & AB_1 : \multimap[]\rightarrow \mathrm{Out}_{Sys}(diff[\quad two(one(null)), \\
& \qquad\qquad\qquad\quad\ \ two(one(null))]), \\
& AB_2 : \multimap[]\rightarrow \mathrm{Out}_{Sys}(diff[\quad two(one(null)), \\
& \qquad\qquad\qquad\quad\ \ one(one(one(null)))]), \\
& AB_3 : \multimap[]\rightarrow \mathrm{Out}_{Sys}(diff[\quad two(one(null)), \\
& \qquad\qquad\qquad\quad\ \ one(two(null))])\}.
\end{aligned}
$$

Example: coins again

$$
\begin{aligned}
S = \{ \quad & AB_1 : \multimap[]\rightarrow \mathrm{Out}_{Sys}(diff[\quad two(one(null)), \\
& \qquad\qquad\qquad\quad\ \ two(one(null))]), \\
& AB_2 : \multimap[]\rightarrow \mathrm{Out}_{Sys}(diff[\quad two(one(null)), \\
& \qquad\qquad\qquad\quad\ \ one(one(one(null)))]), \\
& AB_3 : \multimap[]\rightarrow \mathrm{Out}_{Sys}(diff[\quad two(one(null)), \\
& \qquad\qquad\qquad\quad\ \ one(two(null))])\}.
\end{aligned}
$$

Keeping the environment *Env* from earlier results in the bi-system S not satisfying observational equivalence.

Example: coins again

$$
\begin{aligned}
S = \{ \quad & AB_1 : \multimap[]\rightarrow \mathrm{Out}_{Sys}(diff[\quad two(one(null)), \\
& \qquad\qquad\qquad\qquad\quad two(one(null))]), \\
& AB_2 : \multimap[]\rightarrow \mathrm{Out}_{Sys}(diff[\quad two(one(null)), \\
& \qquad\qquad\qquad\qquad\quad one(one(one(null)))]), \\
& AB_3 : \multimap[]\rightarrow \mathrm{Out}_{Sys}(diff[\quad two(one(null)), \\
& \qquad\qquad\qquad\qquad\quad one(two(null))])\}.
\end{aligned}
$$

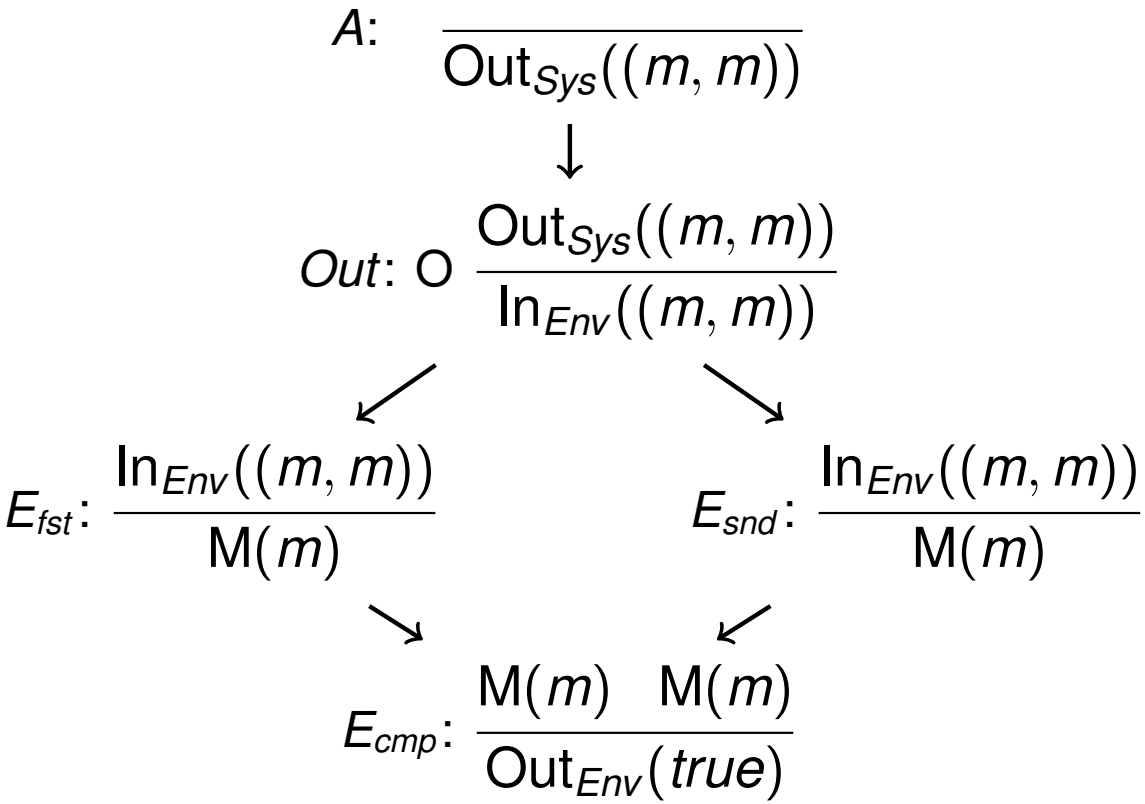
Keeping the environment *Env* from earlier results in the bi-system S not satisfying observational equivalence.

If we add the equation $two(x) = one(one(x))$, then S satisfies observational equivalence.

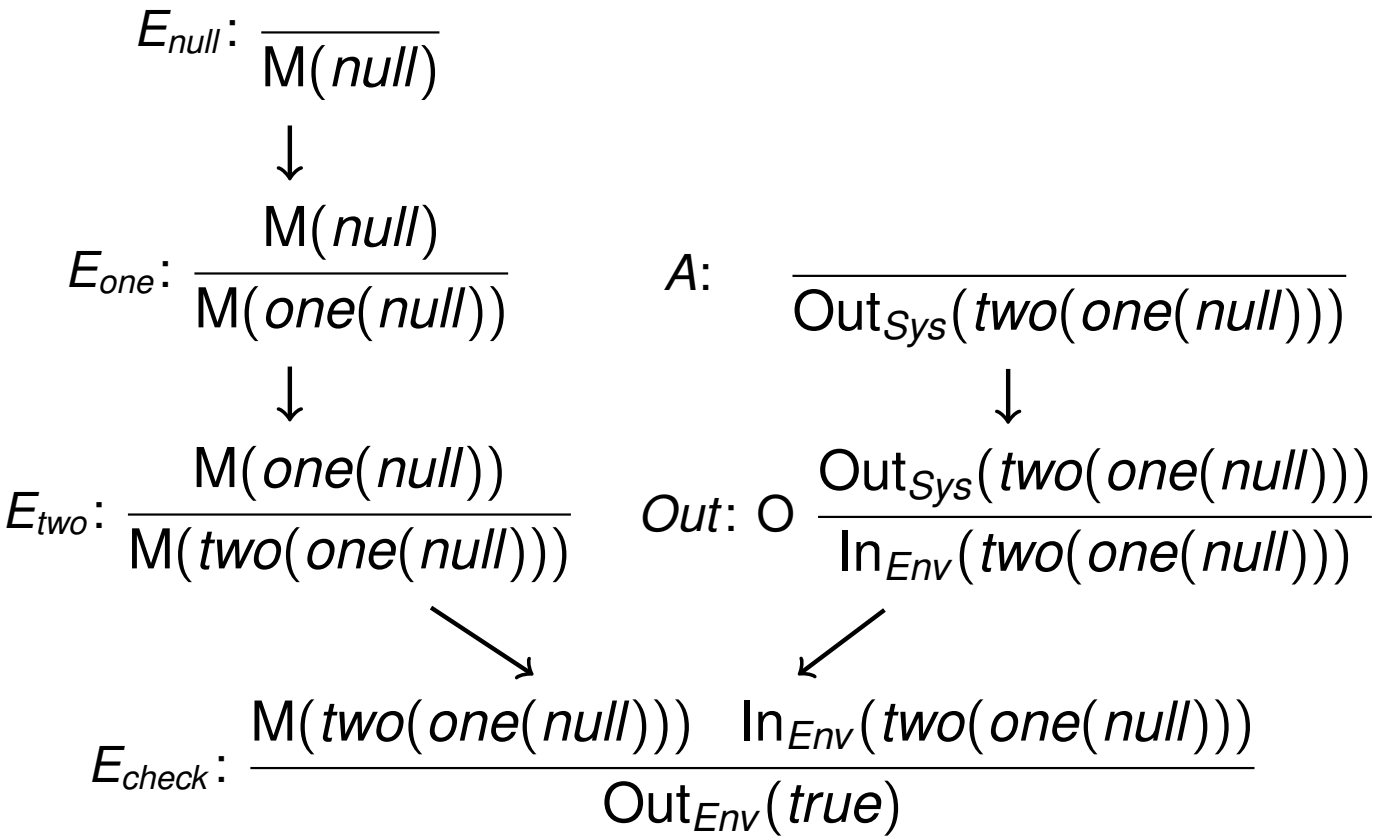
Dependency Graphs

- To simplify reasoning, our algorithm works with dependency graphs rather than with the labeled transition system.
- Dependency graphs are a data structure that formalize the entire structure of a system execution, including which facts originate from which rules, similar to recipes.
- They are well-suited for automated analysis using constraint solving, as they cover whole system state.
- Dependency graphs naturally give rise to an equivalence relation that implies observational equivalence; however, it is substantially simpler to verify.

Dependency graph example for pairs



Dependency graph example for coins



Mirroring dependency graphs

- mirrors of dependency graphs: mirrors contain all dependency graphs on the other side of the bi-system with the same structure,
  - ★ the same edges
  - ★ where the nodes are instances (potentially different due to the diff-terms) of the same rules.
- If the set of mirrors contains all “necessary instances” (for new diff variables), each transition is enabled on the other side as well.
- This guarantees observational equivalence.

Mirroring dependency graphs

- mirrors of dependency graphs: mirrors contain all dependency graphs on the other side of the bi-system with the same structure,
  - ★ the same edges
  - ★ where the nodes are instances (potentially different due to the diff-terms) of the same rules.
- If the set of mirrors contains all “necessary instances” (for new diff variables), each transition is enabled on the other side as well.
- This guarantees observational equivalence.
- Mirroring DG are sufficient, but not necessary for observational equivalence, see Theorem 1.

Reminder: Algorithm

1: **function** VERIFY(S)
2: $RU \leftarrow L(S) \cup R(S) \cup IF \cup Env$
3: **while** $RU \neq \emptyset$ **do**
4: choose $r \in RU$, $RU \leftarrow (RU \setminus \{r\})$
5: compute $DG \leftarrow dgraphs(r)$ by constraint solving
6: **if** $\exists dg \in DG$ s.t. $mirrors(dg)$ lacks ground instances
7: **then return** “potential attack found: ”, dg
8: **return** “verification successful”

Implementation

Execution trace of

$$S_0, (l_1 \xrightarrow[rec_1]{a_1} r_1), S_1, \ldots, S_{k-1}, (l_k \xrightarrow[rec_k]{a_k} r_k), S_k$$

is the sequence of the multisets of its action labels $[a_1, \ldots, a_k]$.

Implementation

Execution trace of

$$S_0, (l_1 \xrightarrow[rec_1]{a_1} r_1), S_1, \ldots, S_{k-1}, (l_k \xrightarrow[rec_k]{a_k} r_k), S_k$$

is the sequence of the multisets of its action labels $[a_1, \ldots, a_k]$.

The adversary's message deduction capabilities are captured by the following set of rules.

$$\begin{aligned} MD = \{\ & \mathsf{Out}(t) \text{—}[]\!\!\rightarrow \mathsf{K}(t),\ \mathsf{K}(t) \text{—}[\ \mathsf{K}(t)\ ]\!\!\rightarrow \mathsf{In}(t), \\ & \mathsf{Fr}(x : fr) \text{—}[]\!\!\rightarrow \mathsf{K}(x : fr),\ [] \text{—}[]\!\!\rightarrow \mathsf{K}(x : pub)\ \} \\ \cup \{\ & \mathsf{K}(t_1), \ldots, \mathsf{K}(t_n) \text{—}[]\!\!\rightarrow \mathsf{K}(f(t_1, \ldots, t_n)) \mid f \in \Sigma^n_{Fun}\ \} \end{aligned}$$

Implementation

Added new message deduction rule:

$$\mathit{IEquality} : \mathsf{K}^{\downarrow}(x), \mathsf{K}^{\uparrow}(x) \multimap [] \rightarrow .$$

Implementation

Added new message deduction rule:

$$\mathit{IEquality} : \mathsf{K}^{\downarrow}(x), \mathsf{K}^{\uparrow}(x) \multimap[] \rightarrow .$$

Compare different sources of x.

Implementation

Added new message deduction rule:

$$\mathit{IEquality} : \mathsf{K}^{\downarrow}(x), \mathsf{K}^{\uparrow}(x) \multimap[] .$$

Compare different sources of x.

- If a side can construct the same value twice this rule is applicable
- If the same source is used twice, the other side can trivially mirror it
- But, if one side can decrypt a value and compare it, but the other cannot, this will be exposed.

Outline

Case studies

- Probabilistic encryption - next
- DDH - see theory file and paper (if time permits)

Case studies

- Probabilistic encryption - next
- DDH - see theory file and paper (if time permits)
- TPM_Envelope - find attack in lab
- Signed Diffie-Hellman - analyze in lab

Probabilistic encryption

Equational theory:

$$pdec(penc(m, pk(k), r), k) \simeq m\,.$$

This equation gives rise to the following decryption rule for probabilistic encryption for the adversary, which is automatically generated by TAMARIN:

$$Dpenc : \mathrm{K}(penc(m, pk(k), r)), \mathrm{K}(k) \text{—}[]\!\!\rightarrow \mathrm{K}(m)\,.$$

Probabilistic encryption (ctd.)

We now express, as a bi-system, that a probabilistic encryption cannot be distinguished from a random value:

$$
\begin{array}{lll}
S = \{ & \mathit{GEN}: & \mathsf{Fr}(k) \multimap \mathsf{Key}(k), \mathsf{Out}(\mathit{pk}(k)) \\
& \mathit{ENC}: & \mathsf{Key}(k), \mathsf{Fr}(r_1), \mathsf{Fr}(r_2), \mathsf{In}(x) \multimap \\
& & \mathsf{Out}(\mathit{diff}[r_1, \mathit{penc}(x, \mathit{pk}(k), r_2)]) \quad \} .
\end{array}
$$

Probabilistic encryption (ctd.)

We summarize below how TAMARIN automatically proves this property. The algorithm VERIFY (line 2) first constructs the set *RU* of rules to be analyzed,

$$RU = \{ \quad L(GEN), R(GEN), L(ENC), R(ENC), \\ \text{FRESH}_{Sys}, \text{FRESH}_{Env}, IEquality, Dpenc \} ,$$

together with the remaining rules in *IF* and *Env*.

Algorithm VERIFY iterates over all rules (lines 3–4) until either an attack is found (line 7) or all rules have been checked and the verification is complete (line 8), which happens here.

Probabilistic encryption (ctd.)

VERIFY works like this:

- VERIFY first generates dependency graphs with the rule as the root (line 5).
- For each resulting dependency graph, it looks for a mirror (line 6) that contains all instances required by the definition of normal dependency graph equivalence.
- It always finds a mirror and verification therefore succeeds.
- TAMARIN analyzes left-diff and right-diff instantiations independently, we present them together.
- No explicit dependency graphs shown (run the tool to see those); however, we do explain how they are mirrored in each case so that the verification succeeds.

Probabilistic encryption (ctd.)

$$
\begin{aligned}
S = \{ \quad & \mathit{GEN}: && \mathsf{Fr}(k) -[]\!\!\rightarrow \mathsf{Key}(k), \mathsf{Out}(\mathit{pk}(k)) \\
& \mathit{ENC}: && \mathsf{Key}(k), \mathsf{Fr}(r_1), \mathsf{Fr}(r_2), \mathsf{In}(x) -[]\!\!\rightarrow \\
& && \mathsf{Out}(\mathit{diff}[r_1, \mathit{penc}(x, \mathit{pk}(k), r_2)]) \qquad \}\,.
\end{aligned}
$$

- As rule *GEN* does not contain a diff-term, the left diff-instantiation of this rule is identical to the right diff-instantiation.
- The rule has only a single fresh fact as its premise and thus any dependency graph with this rule at its root contains only those two rule instances and is trivially mirrored by itself.

Probabilistic encryption (ctd.)

$$S = \{ \quad \begin{array}{ll} \mathit{GEN}: & \mathsf{Fr}(k) -[]\!\!\rightarrow \mathsf{Key}(k), \mathsf{Out}(\mathit{pk}(k)) \\ \mathit{ENC}: & \mathsf{Key}(k), \mathsf{Fr}(r_1), \mathsf{Fr}(r_2), \mathsf{In}(x) -[]\!\!\rightarrow \\ & \mathsf{Out}(\mathit{diff}[r_1, \mathit{penc}(x, \mathit{pk}(k), r_2)]) \end{array} \quad \}.$$

- The rule *ENC* has the same premises in the left- and right-hand side system and is therefore identical for the purpose of dependency graph computation with the *ENC* rule as root. (Note that outputs will be considered using the equality rule below.)
- The two fresh premises will result in identical dependency graphs, while the key and message reception input are independent.
- Hence both of them will have identical dependency graphs as premises, and the resulting dependency graphs are identical (up to the outputs) and therefore mirror each other.

Probabilistic encryption (ctd.)

The fresh rules FRESH_{Sys} and FRESH_{Env} have no premises. Hence the dependency graphs with them as root are just their instances, which mirror each other in the left- and right-hand system.

Probabilistic encryption (ctd.)

- For an equality rule instance of *IEquality* as the root of a dependency graph, the two premises are the same instance of a variable x.
- If both of the premises are adversary generated, then the resulting dependency graphs are the same in the left- and right-hand system, and thus will mirror themselves trivially.

Probabilistic encryption (ctd.)

- For an equality rule instance of *IEquality* as the root of a dependency graph, the two premises are the same instance of a variable x.
- If both of the premises are adversary generated, then the resulting dependency graphs are the same in the left- and right-hand system, and thus will mirror themselves trivially.

- Alternatively, if one of the premises uses the output of an instance of either the *ENC* rule or the *GEN* rule, then there is no dependency graph with a matching second premise.
- This is because all system outputs, $pk(k)$ for *GEN* and $r1$ or $penc(x, pk(k), r2)$ for *ENC*, contain a fresh value, k, $r1$, respectively $r2$, that is never available to the intruder.
- As this will never allow a complete dependency graph to be derived, no mirroring dependency graph is needed.

Probabilistic encryption (ctd.)

- For the decryption rule generated for the probabilistic encryption, this rule is never applicable on either side as the adversary never receives the keys needed for decrypting system generated encryptions.
- As there is no dependency graph, no mirroring one is needed.

Probabilistic encryption (ctd.)

- For the decryption rule generated for the probabilistic encryption, this rule is never applicable on either side as the adversary never receives the keys needed for decrypting system generated encryptions.
- As there is no dependency graph, no mirroring one is needed.

(One might mistakenly think that this rule might apply to intruder-generated terms. However, this is not the case due to the restrictions on how the adversary may combine its knowledge ($K^{\downarrow}$ vs $K^{\uparrow}$) and, in any case, both sides would use the same dependency graphs as premise, so the result would be the same.)

Probabilistic encryption (ctd.)

For all other adversary rules, it is obvious that they result in identical dependency graphs on both sides.

More precisely:

- Construction rules have adversary knowledge input and thus the same dependency graphs as premises.
- For the deconstruction rules, the only relevant one is the previous decryption rule, as that is the only one that can use information coming out of the system; all other rules can only be used on adversary-generated terms and thus have the same dependency graphs as premises.

Probabilistic encryption (ctd.)

This completes our summary of TAMARIN's verification of observational equivalence for this example. TAMARIN automatically constructs the proof in under 0.2 seconds.

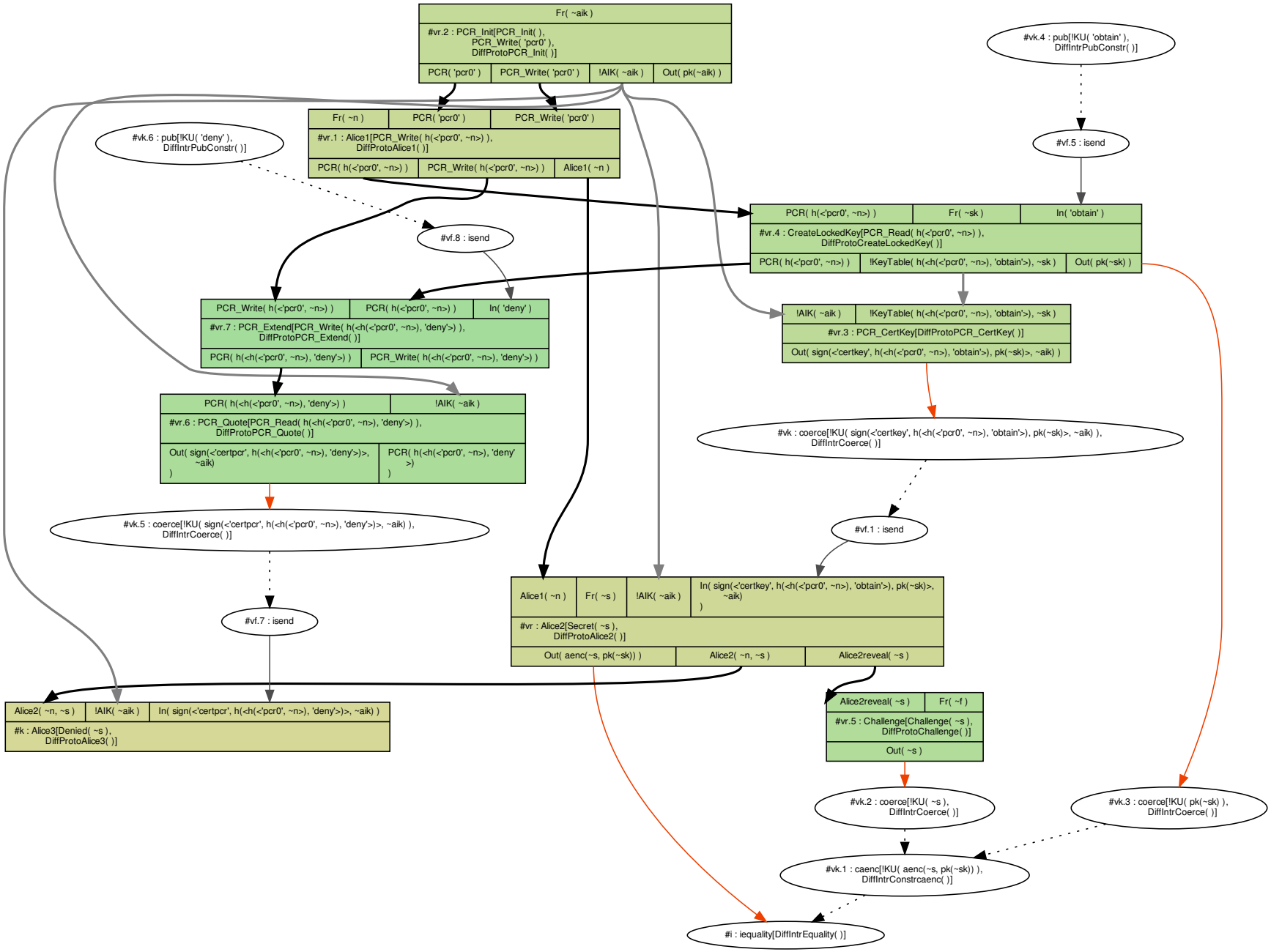
DDH

Demo if time permits

Simplified TPM_Envelope specification

$$
\begin{aligned}
\mathit{Init}: &\ \mathsf{Fr}(\mathit{aik}) -[]\rightarrow \mathsf{PCR}('\mathit{pcr0}'), \mathsf{AIK}(\mathit{aik}), \mathsf{Out}(\mathit{pk}(\mathit{aik}))\\
\mathit{Ext}: &\ \mathsf{PCR}(x), \mathsf{In}(y) -[]\rightarrow \mathsf{PCR}(h(x,y))\\
\mathit{CertK}: &\ \mathsf{AIK}(\mathit{aik}), \mathsf{KT}(\mathit{lock}, \mathit{sk}) -[]\rightarrow\\
&\ \mathsf{Out}(\mathit{sign}(\langle '\mathit{certk}', \mathit{lock}, \mathit{pk}(\mathit{sk})\rangle, \mathit{aik}))\\
\mathit{Quote}: &\ \mathsf{PCR}(x), \mathsf{AIK}(\mathit{aik}) -[]\rightarrow\\
&\ \mathsf{PCR}(x), \mathsf{Out}(\mathit{sign}(\langle '\mathit{certcpr}', x\rangle, \mathit{aik}))\\
\mathit{Unbind}: &\ \mathsf{PCR}(x), \mathsf{KT}(x, \mathit{sk}), \mathsf{In}(\mathit{aenc}(m, \mathit{pk}(\mathit{sk}))) -[]\rightarrow\\
&\ \mathsf{PCR}(x), \mathsf{Out}(m)\\
\mathit{A1}: &\ \mathsf{Fr}(n), \mathsf{PCR}(x) -[]\rightarrow \mathsf{PCR}(h(x,n)), \mathsf{A1}(n)\\
\mathit{A2}: &\ \mathsf{Fr}(s), \mathsf{A1}(n), \mathsf{AIK}(\mathit{aik}),\\
&\ \mathsf{In}(\mathit{sign}(\langle '\mathit{certk}', h(h('\mathit{pcr0}', n), '\mathit{obtain}'), \mathit{pk}\rangle, \mathit{aik}))\\
&\ -[]\rightarrow \mathsf{Out}(\mathit{aenc}(s, \mathit{pk})), \mathsf{A2}(n,s), \mathsf{A2ror}(s)\\
\mathit{A3}: &\ \mathsf{In}(\mathit{sign}(\langle '\mathit{certpcr}', h(h('\mathit{pcr0}', n), '\mathit{deny}')\rangle, \mathit{aik})),\\
&\ \mathsf{A2}(n,s), \mathsf{AIK}(\mathit{aik}) -[\ \mathsf{Denied}(s)\ ]\rightarrow\\
\mathit{CLKey}: &\ \mathsf{Fr}(\mathit{sk}), \mathsf{PCR}(x), \mathsf{In}(\mathit{lock}) -[]\rightarrow\\
&\ \mathsf{PCR}(x), \mathsf{KT}(h(x, \mathit{lock}), \mathit{sk}), \mathsf{Out}(\mathit{pk}(\mathit{sk}))\\
\mathit{ROR}: &\ \mathsf{A2ror}(s), \mathsf{Fr}(f) -[]\rightarrow \mathsf{Out}(\mathit{diff}[s, f])
\end{aligned}
$$

TPM_Envelope attack



Conclusions

- Observational equivalence can be automatically proven
- Diff-terms allow concise description
- Approach is sound (mirroring DG are sufficient, not necessary)
- Compares well to other approaches

Conclusions

- Observational equivalence can be automatically proven
- Diff-terms allow concise description
- Approach is sound (mirroring DG are sufficient, not necessary)
- Compares well to other approaches

(-: Happy Proving :-)